

رسالة البرمجة بايثون

م. وائل حسن محمد علي (أبو إياس)

بِسْمِ اللَّهِ، وَ الصَّلَاةُ وَ السَّلَامُ عَلَي رَسُولِ اللَّهِ، وَ عَلَي آلِهِ وَ صَحْبِهِ وَ مَنْ وَآلَاهُ،

رِسَالَةُ الْبِرِّمَجَّةِ بِإِبْدَاعِ

الإصدارة 1

كَتَبَهَا:

م. وائل حسن محمد علي (أبو إياس)

إِلَى إِسْمَاعِيلِ الدَّرِينِيَا:

عَلِيِّ بْنِ حَمَزِ النَّظَّاهِرِيِّ الْأَنْدَلُسِيِّ

خَفَرَ اللَّهُ لَهُ وَرَفَعَهُ دَرَجَاتِهِ فِي الْجَنَّةِ

وَالسَّبِيحِ الْعَلَّامَةِ:

أَبِي إِسْحَاقَ الْمُحَوِّثِيِّ الْأَنْدَلُسِيِّ

بَارَكَ اللَّهُ فِي عَمْرِهِ

شكرٌ خاصٌ

هذا الشكر الخاص أُوجِّهُهُ (بعد الله عز و جل) إلي برنامج (نجوم العلوم stars of science) و القائمين عليه من: مُمَوِّلِينَ، و مُنْتَجِينَ، و فريق عملٍ و مُحَكِّمِينَ، و كذلك رفاقي في الموسم الرابع؛ ذلك لأنه أجبرني علي تغيير أسلوب عملي في المشروع في فترة كنتُ أحوَجُ ما أكون فيها لذلك التغيير، كما أنه أتاح لي فرصة من الصعب أن تتكرَّر من التفرغ التام، بعيداً عن المشاغل اليومية و المنغصَّات الحياتية و تكاليف التوا صل الاجتماعي، بما كفَّل لي تفرغ طاقتي الذهنية كاملةً للمشروع، و هو ما أدي به إلي أن يقفز قفزاتٍ غير متوقَّعةٍ بالنسبة للفترة القصيرة التي شاركتُ بها في المُسَابَقة. و تكفَّل الرفاق و طاقم العمل بالجو الحميم الذي جعل العمل أكثر مُتعةً و سلاسةً.

فشكراً لهم جميعاً

رِخْطَةُ مَحْتَوِيَّاتِ هَذَا الْكِتَابِ

كُلُّ الْمَوَادِّ الْمَنْشُورَةِ فِي هَذَا الْكِتَابِ تَخْضَعُ لِبُنُودِ التَّرْخِيصِ التَّالِيَةِ، وَاسْتِمْرَارِكَ فِي قِرَاءَةِ مَحْتَوِيَّاتِ الْكِتَابِ تُعْتَبَرُ مُوَافَقَةً مِنْكَ عَلَيَّ تِلْكَ الْبُنُودِ:

- يُمَكِّنُ لِأَيِّ أَحَدٍ نَقْلَ أَيِّ مَحْتَوِيَّاتِ الْكِتَابِ أَوْ حَتَّى الْكِتَابِ كَامِلًا عَلَيَّ أَيَّ شَبَكَةٍ تَوَاصَلَ اجْتِمَاعِيٍّ أَوْ أَيِّ وَسِيلَةٍ رَقْمِيَّةٍ أَوْ غَيْرِ رَقْمِيَّةٍ، بِشَكْلِ غَيْرِ تِجَارِيٍّ مَعَ الْإِلْتِزَامِ بِ:
 - ذِكْرِ اسْمِ الْكَاتِبِ الثَّلَاثِي (بِشَكْلِ وَاضِحٍ فِي كُلِّ عَمَلِيَّةِ نَقْلِ)، وَ/أَوْ:
 - وَضْعِ اسْمِ الْكِتَابِ نَفْسَهُ (بِشَكْلِ وَاضِحٍ فِي كُلِّ عَمَلِيَّةِ نَقْلِ).
- لَا يُسَمَّحُ بِالتَّعْدِيلِ فِي أَيِّ مَحْتَوِيٍّ مِنْ مَحْتَوِيَّاتِ الْكِتَابِ بِأَيِّ شَكْلِ مِنْ الْأَشْكَالِ ثُمَّ إِعَادَةِ نَشْرِهِ بِاسْمِ الْمُؤَلِّفِ أَوْ اسْمِ أَيِّ شَخْصٍ آخَرَ، عَلَيَّ أَيِّ وَسِيطٍ رَقْمِيٍّ أَوْ غَيْرِ رَقْمِيٍّ، إِلَّا بَعْدَ اخْتِزَانِ الْإِذْنِ مِنَ الْمُؤَلِّفِ الْكِتَابِ.
- فِي حَالَةِ النَّقْلِ إِلَى مَحْتَوِيٍّ رَقْمِيٍّ أَوْ غَيْرِ رَقْمِيٍّ بِشَكْلِ تِجَارِيٍّ فَلَا يُسَمَّحُ بِهَذَا إِلَّا بَعْدَ الْحَصُولِ عَلَيَّ إِذْنٍ مِنَ الْمُؤَلِّفِ.
- بِالنِّسْبَةِ لِلْعَلَامَاتِ التِّجَارِيَّةِ الْمَنْشُورَةِ فِي هَذَا الْكِتَابِ فَإِنَّهَا تُعْتَبَرُ مِلْكِيَّةً فِكْرِيَّةً لِأَصْحَابِهَا.

الفهرست

13	القِسْمُ الأولُ: التَّوْطِئَةُ:
15	■ عن الواقع العلمي الإسلامي
23	■ عن المُستقبل
31	■ الإعتراضات المُوجَّهة لإنتاج لغة برمجةٍ عربيَّةٍ و الرد عليها:
33	• الإعتراض الأول
34	• الإعتراض الثاني
40	• الإعتراض الثالث
42	• الإعتراض الرابع
43	• الإعتراض الخامس
44	• الإعتراض السادس
46	• الإعتراض السابع
46	• الإعتراض الثامن
47	• الإعتراض التاسع
48	• الإعتراض العاشر
48	• الإعتراض الحادي عشر
49	• الإعتراض الثاني عشر
50	• الإعتراض الثالث عشر
50	• الإعتراض الرابع عشر
51	• الإعتراض الخامس عشر
52	• الإعتراض السادس عشر
53	• الإعتراض السابع عشر
54	• الإعتراض الثامن عشر
56	• الإعتراض التاسع عشر
58	• الإعتراض العشرون
59	• الإعتراض الحادي والعشرون
61	• الإعتراض الثاني والعشرون
62	• الإعتراض الثالث والعشرون
62	• الإعتراض الرابع والعشرون
62	• الإعتراض الخامس والعشرون

63	• الإعتراض السادس و العشرون
65	■ نقد لغات البرمجة العربية الحالية:
68	• قائمة بلغات البرمجة العربية التي أعلمها
70	• اللغات المجهولة
72	• اللغات المترجمة
77	• اللغات المنقولة
79	• اللغات المُبتكرة
99	■ منهاج العمل:
101	• خطة (ما قبل نجوم العلوم)
103	• خطة (ما بعد نجوم العلوم)
105	• الخطوط العريضة لإدارة المشروع
107	■ القاموس
110	■ المصادر

113	القِسْم الثاني: عن المُواصفات القياسية للغة البرمجة العربية إيداع:
115	■ شرح المُواصفات القياسية لإيداع:
117	• صفات اللغة
118	• رخصة مُواصفات لغة البرمجة إيداع في الإصدار (1.0)
119	• مبادئ التصميم التي تم اعتبارها أثناء تصميم إيداع
121	• قائمة اللغات التي تم الإستفادة منها عند تصميم إيداع
123	• هيكل البرنامج في اللغة
128	• المُتغيّرات و الثوابت
131	• المُعاملات المنطقية و الحسابية
133	• الإجراءات
147	• الأصناف
156	• خُيوط التنفيذ
158	• الجُملة الشرطية لو
160	• الجداول
164	• الجُملة التكرارية بينما
167	• الألقاب
169	• مُعالجة الأغلط
173	• استخدام الأكواد الخارجية في إيداع

175	■ روح الإبداع:
177	• التَّخْصُّص
182	• الْوَرَاثَةُ
185	• التَّهْجِين
188	• الشُّمُول
197	• الْبَسَاطَةُ وَ الْاِسْتِقْرَار
202	• الْأَمْن
204	• الْأُلْفَةُ
209	■ الْمَكُونَاتُ الَّتِي رُفِضَ ضَمُّهَا إِلَى اللُّغَةِ:
211	• الْوَاجِهَاتُ interfaces
213	• الْهَيْكَلُ structs
216	• جُمْلَةٌ goto
218	• الْمَوْسَّرَاتُ pointers
222	• الْإِجْرَاءَاتُ procedures
225	• الْوَسَائِطُ delegates
226	• الْمَهْمَاتُ Tasks
228	• الْمُعْلَفَاتُ Properties
230	• الْمَفْهَرِسَاتُ indexers
232	• نَمُوذَجُ الْpython لِلْأَصْنَافِ وَ الدَّوَالِ
236	• الدَّامِجَاتُ unions
239	■ أَيْنَ الْإِبْدَاعُ فِيهَا؟
254	■ الْقَامُوسُ
257	■ الْمَصَادِرُ
260	<hr/> السيرة الذاتية
263	<hr/> أدوات إنتاج هذا الكتاب



عن الواقع العلمى الإسلامى

ليس هناك عصرٌ كانت الأمة الإسلامية فيه فى أضعف حالاتها و أشدها تدهوراً من هذا العصر؛ فالواقع أن الأمة الإسلامية اليوم هى أشد أمم الأرض ضعفاً و خنوعاً، و يتجلى هذا فى كل مناحى الحياة الإسلامية بلا استثناءٍ واحد، بدءاً من التشرذم بين أقطار الإسلام و انفراد أهل كل قطرٍ بقُطْرِهِم، و تفشّى الدعوات الشُّعوبية و القُطرية بين أفراد الأمة لتُردم الهوية الإسلامية تحت رُكامٍ من صراخات أهل تلك الدعوى البغيضة.

إلى انتشار كل خُلُقٍ كَرِيهٍ بين عامة المسلمين، من سلبيةٍ و خُنوعٍ إلى التباغض و التنافر حتى بين أفراد العائلة الواحدة (بل و خصوصاً بين أفراد العائلة الواحدة!)، إضافةً إلى ذلك الفساد الذى انتشر كالسوس لينخر فى بلاد الإسلام على مرأىٍ و مَسْمَعٍ من كل مسئول، بل و بمُشاركةٍ و مُباركةٍ كثيرٍ من هؤلاء المسئولين، الأمر الذى كشفته ثورات الربيع العربى المُباركة و ستكشف الأيام التالية أغلب ما خَفِيَ منه اليوم بإذن الله عز و جل.

فراينا بأم أعيننا مدي فداحة الخسارة التى مُنينا بها من السرقات و الاختلاسات التى ولغ فيها من لم يُراعوا عهداً و لا أمانة، فسرقوا و نهبوا ما يكفي لجعلهم يعيشون هم و أحفاد أحفاد أحفادهم ملوكاً مُتَوَجِّين! و كأنه قد خِيَل إليهم أن نقودهم ستصحبهم إلى الآخرة فأرادوا الاستزادة بلا شع!

و لا تفسير لكل ذلك التَّهْم للسرقة إلا القول المنسوب لرسول الله صلوات الله و سلامه عليه: **(مَنْهُمَانِ لَا يَشْبَعَانِ : مَنْهُومٌ فِي عِلْمٍ لَا يَشْبَعُ، وَ مَنْهُومٌ فِي دُنْيَا لَا يَشْبَعُ)**¹

و ليس هذا بآخر السوء: فالزيادة فى الواقع العلمى المُخجل للأمة؛ فاليوم يعيش المسلمون عالةً على غيرهم فى العلوم و المُنتجات العلمية بكل المقاييس:

فلا ننتج بل نستهلك،

و لا نختار أجود المُستهلكات بل نختار الأسرع ربحاً و الأفضل مكسباً،

و حينما نستهلك ما نستهلكه لا نُسَمِّيه باسمه الأصلى بل باسم السلعة التى استهلكناها، فيُصبح اسم كل نظام تشغيلٍ: **windows**، و تُصبح كل لعبةٍ رقميةٍ: **atari**. و مثل هذا كثير.

بل و الأنكى و الأشد إثارة للغیظ أننا ننفق الكثير و الكثير على توافه الأمور، و لا ننفق على العلوم إلا أقل القليل! و حتى أقل القليل هذا لا ننفقه على العلم الحقيقى، بل على العلم من عينة "تعلم كيف تصنع أسطوانة ويندوز عليها صورك و برامجك"، و ما تبقى مما يُنفق يتم إنفاقه على مرتبات موظفى الحكومة من النوعية التى صارت لا تعباً لا بعلمٍ و لا بتقدمٍ و لا بأى شيءٍ يَمُتُّ لـ "وجع الرأس" هذا بصلة!

أما العلم الحقيقى فليس له مُعِينٌ من المسئولين و أهل الحَل و العَقْد، و ليس أمام المُبدع الحقيقى إلا أن يبحث عَمَّن هم على شاكلته لِيُشْكِلُوا مع بعضهم البعض ثلثة من المنبذين و يُحاولوا تجاوز واقعهم المريض، أو أن يرضى بالواقع و يحاول جاهداً التكيف معه، و دَفَع نفسه دفعا للإحساس أن كل الأمور علي ما يُرام، و أنه "ليس فى الإمكان أبدع مما كان"، و متي نَجح فى هذا عاش راضياً عن نفسه و عن الكون بما فيه بذات الطريقة التى يَرْضِي بها الدجاج عن عالمه.

و لم يكن مُستغرباً و الحال هكذا أن نرى أن العالم الإسلامى بأُسْرِهِ (و هو المُكوّن مما يُقارب الإثنين و خمسين دولة دينها الأول الإسلام) يشهد انحطاطاً مُفزعاً فى جامعاته و معاهدته و مدارسه و مواقعه التعليمية كلها تقريباً (إلا من رَحِمَ اللهُ تعالى)، و أن يصل حال الإنتاج العلمى فيه فى شتى المجالات إلى الحضيض.

و بينما يهتم أهل التربية و التعليم عندنا بتلقين الطلاب أكبر كميةٍ مُمكنةٍ من المعلومات التى ليس لها أى أهميةٍ لا فى حياتهم العادية و لا فى مستقبلهم العلمى، يقوم العالمُ المُتقدِّم بتحبيب طلابه فى العلم و التحصيل العلمى و الإستماتع بهما، و بينما يستمتع الطالب الغربى (أو على الأقل لا يتم تعذيبه بالتُسْرَهات) بفرق الطالب عندنا فيما يُشبه ما فى كتاب الجغرافيا للصف الأول الثانوى فى مصر: أطنانٍ من المعلومات و الأرقام عن ترتيب دول العالم من حيث إنتاج الماشية و الأرز و بقية المحاصيل، و نَسبهم المئوية من

¹ هذا الحديث المنسوب لرسول الله صلى الله عليه و سلم فيه كلامٌ يطول بين علماء الحديث عن صحته، فيُرْجَى التنبه إلي هذا.

ذلك الإنتاج العالمي ! و كأن الطالب سيخرج من المدرسة ليُصبح وزير الزراعة مباشرةً ! فتكون النتيجة المُتوقَّعة هي أن الطالب صار يكره العلم و التعلم كره الطاعون.

أما في البلاد التي تحترم آدمية البشر فبدلاً من التلقين و الإعتماد علي كمية المعلومات التي حفظها الطالب يعتمدون علي تنمية مهارات التفكير و البحث العلميين، و يعتبرونهما أهم من كمية المعلومات التي يسكبها الطالب علي ورقة الإمتحان ثم ينساها فور خروجه من اللجنة.

فتحوّل الطالب عندنا إلي ما يُشبه الحمار الذي يحمل أسفارا لا يفقه ما فيها و لا يهتم به، و يهمله أكثر ما يهمله مجموعهُ الذي سيحصل عليه في النهاية حتي يتأهل لكلية من كليات " القمة "، و التي بدورها تقضي علي البقية الباقية من ملكات النقد و الإستقلالية و حب البحث العلمي عنده، ليتحول إلي حمارٍ كاملٍ بدون أدني نقص.

و تحوّل الطالب الجامعي عند الآخرين إلي ما كينة علم لا ترحم؛ فما بين كُليته التي يتعايش فيها عملياً مع ما يدرسه و يري فيها مُدرِّسين ذوي كفاءةٍ و مِرَاسٍ، و بين مُدَوّنته/موقعه الشخصي/حسابه في المنتديات العلمية؛ يعيش جواً علمياً مُكتمل الأركان، و لا يتقصه سوي أن يفرِّغ نفسه لحضور بعض المُحاضرات و المؤتمرات التي تُنظّمها الشركات الرائدة تقنياً في العالم، و التي يُوجد منها في بلده و علي مسافةٍ قريبةٍ منه ما ليس بالقليل، و أن يُشارك في المُسابقات العالمية التي تُنظّمها تلك الشركات.

و الناتج النهائي أن الطلبة عندهم يُخرجون مُنتجاتٍ تقنيةٍ من أفضل ما يكون، و بعض مُنتجاتهم يكون من النوعية التي تقلب كافة الموازين، مثل نواة اللينوكس linux kernel التي بدأها الطالب لينوس تورفالدز linus torvalds ثم ساعده فيها جمعٌ غفيرٌ من المُبرمجين المُتحمسين في أنحاء العالم، و بيئة سطح

المكتب KDE التي بدأها الطالب ماتثياس إترش **matthias Ettrich** !

بينما رأيتُ و سمعتُ الكثير من المخازي عن طلبتنا، و من أفدحها حينما أراد أحد المُعيدين أن يمزح مع الطلبة فسألهم سؤالاً سخيفاً: "لو وقعت الفلاشة علي الأرض هل ستنتقل لها فيروسات؟"، ليُفاجأ المسكين أن الإجابات التي حَصَل عليها كانت من النوع النووي؛

فقد أجاب كثيرون أنهم "لا يعلمون" !

بينما قال البعض الآخر أن "الفيروسات ستنتقل للفلاشة بلا ريب" !

و لله الأمر من قبل و من بعد.

و لأنني أَعُدُّ نفسي من أهل البحث العلمي عامةً، و من أهل علوم الحاسب خاصةً: فإن هذا الواقع لا يُرضيني البتة، و أجد أنه ليس من الدين و التقوى في شيء أن أظل (بينما أنا قادرٌ مع مَنْ هم مثلي علي تغيير الواقع) مكتوف الأيدي أمام كل هذا العبث، فعلي الأقل يُمكننا فَتْح باب التغيير و لو في تخصصنا العلمي فقط.

فما كان مني إلا أن فكرتُ في النهضة العلمية للأمة:

كيف يُمكن أن تبدأ؟

و إلى أين يُمكن أن تسير؟
 و أى المسارات يُمكنها أن تتخذ؟
 و من من الناس يُمكن أن يُساعد على هذا؟
 و ما هي العقبات التي ستقف في وجه تلك المسيرة؟
 و كيف يُمكن التعامل معها و التغلب عليها؟

شغلتني هذا الأمر لفترةٍ طويلةٍ حتى تيقنتُ من تفكيرى و من خبرتى (الصغيرة فى مجال علوم الحاسب) أن واحداً من أيسر السبل التي يُمكن للأمة أن تبدأ بها النهضة العلمية الجديدة هو طريق علوم الحاسب البرمجية؛ فهذا النوع من العلوم لا يحتاج من الإنفاق المادى مقارنةً بباقي العلوم إلا أقل القليل، و لا يحتاج إلا الكثير من الفكر و الإبداع، و فى هذا فإن أمم الأرض و شعوبها يتساويان إذا ما تم تحصيل العلم على قدرٍ متساوي.

ثم إن هذا العلم مطلوبٌ فى كل دراسةٍ علميةٍ أخرى سواءً أكانت تتعلق بمجال الحوسبة ام لا، و بإنتاجنا للتطبيقات البرمجية الإسلامية فإننا نفتح الباب أمام طوفانٍ من الدراسات التي تستثمر هذه المُنتجات البرمجية فى باقى المجالات.

و الأهم أن خطوة ترجمة و إنتاج علم من العلوم باللغة العربية ستُحقق الكثير من الأهداف (إضافةً إلى ما سنحصل عليه من مُنتجاتٍ صنعناها بأيدينا و جهدنا)، مثل إثبات أنه ليس هناك تلك الأسطورة المُسمّاة بـ(لغة العلم الوحيدة) و بالتالى تصلح كل اللغات للإنتاج العلمى و منها اللغة العربية.

و سيُشجع هذا كل من كان مُتردداً فى التأليف و البحث العلمى باستخدام اللغة العربية على وُلوج ذلك السبيل، لينشأ بالتدريج مُجتمعٌ من العلماء المسلمين يُترجم و يبحث و يُنتج باللغة العربية، و كلما كبر حجم هذا المجتمع زاد أمل الشباب فى انتشار فكر النهضة و القيام من جديد، و كلما زادت جذوره قوةً و امتداداً فى الأرض الإسلامية، حتى يأتى اليوم الذى يُصبح التفكير العلمى فيه أمراً مُعتاداً كما كان فى عصر أجداد الأجداد من العلماء المسلمين، حينما كان التاجر و الحَمَّال بل و حارس الدواب عالماً جليلاً تقطع إليه الفِيافى و القفار لطلب علمه و التعلم على يديه.

و كلما كبر حجم هذا المُجتمع الطيب زادت اللُحمة بين أهله من المسلمين و ذابت الفوارق العرقية بينهم، فلن يكون هناك مصرىٌ أو باكستانىٌ أو أمريكىٌ أو روسىٌ أو يابانىٌ أو صينىٌ أو غيرهن من الجنسيات، بل ستكون هناك الهوية الإسلامية المُوحدة التي تربط بين كل هؤلاء، كما نرى بين معاشر الملتزمين فى كل عَصْرٍ و مِصْرٍ.

و بكثرة الحديث عن هذه النهضة الإسلامية و العلم الذى يُنتجه المسلمون سوف تتوحد الجماهير الإسلامية رويداً رويداً تحت راية الدين من جديد، و سوف يعود الإحساس بالدين كعنصرٍ فاعلٍ فى الحياة اليومية كما كان عليه فى الأيام الأولى، و بالتدريج سيسود الإلتزام بتعاليم الله تعالى و شرعه فى كل مناحى الحياة، و هو الهدف الأعظم الذى نسعى كلنا خلفه و يجب أن يندُر الجميع حياتهم له.

كما أن هناك الكثير من الخبراء و أهل الإبداع فى العالم الإسلامى فى هذا المجال، فهو منتشر الآن جداً على العكس من مجالاتٍ أخرى (مثلاً: تقنية النانو nano technolog، و تقنيات إنتاج عتاد الحواسيب computers hardware). أى أن عامِل الموارد البشرية متوفرٌ جداً و بدمجه بعامِل قلة التكلفة: نرى أن كل العوامل أصبحت متوافرةً لدينا للنهضة البرمجية التى نرغب بها.

على أن هذه النهضة لى يُكتَب لها البقاء و الحياة و إتيان ثمارها المرجوة منها: يجب علينا أن نخطِّط لها جيداً من كل النواحي، عملاً بالمبدأ الإسلامى "اعْقَلْهَا وَتَوَكَّلْ"²، فيجب بدايةً أن:

- نُحدِّد الأهداف المُستقبَلية المرغوبة بالتفصيل، ثم:
- نُحدِّد ما يجب علينا عمله بالضبط لى نحقق هذه الأهداف على أرض الواقع فى قائمة أعمالٍ كاملة، و علينا كذلك:
- تحديد الأولويات بين هذه الأعمال، بأن نرتبها زمنياً بحيث يُتاح لنا أن نرى تقدمنا أثناء العمل، و ما تبقى فعله فى المرحلة الحالية، و ما سيتوجب علينا فى المرحلة القادمة.

و لكن علينا فى بداية كل مرحلةٍ من المراحل (و قبل أن نبدأ بالعمل الفعلى فيها) أن ننظر إن كانت هناك مُحاولاتٍ سابقة فى هذا الشأن يُمكن أن نستفيد منها فى عملنا؛ حتى لا نُحَمِّل أنفسنا عبئاً زائداً فنخسر وقتاً و جهداً قيمين فى إعادة اختراع العجلة، فإن وجدنا أن هناك مُحاولاتٍ سابقة بالفعل: أصبح لزاماً علينا أن ننظر لها بعينٍ ناقدة، من حيث: قابليتها لإفادتنا، و كونها على المُستوى المرغوب أم لا، و هل يُمكن استعمالها حتى كمرحلةٍ و سطى للوصول إلى المُستوى الذى نرغب فيه.

و النقد هكذا أمرٌ ضرورى و ليس ترفاً فكرياً و مضيعةً للوقت كما قد يظن البعض، و الحق أن الكثيرين سَيرون أن هذا الأمر مضيعةٌ للوقت و الجهد، و أنه من الأفضل البدء من الصفر بدلاً من البحث عما فعله الآخرون و مناقشته و نقده. و لكنى أُنبههم أن بعض الجهد فى هذه المُناقشات قد يُوفِّر علينا الكثير من الوقت فى المُستقبل نُضِيعه فى إعادة صُنع المصنوع، فنستفيد منه نحن فى إنجاز خطواتٍ أخرى تُقربنا من أهدافنا أكثر، فالحق أنه لدينا الكثير من الأمور التى يجب علينا إنجازها كما سأفصّل فيما بعد و لذلك فإن أى توفيرٍ للجهد و الوقت يُعد أمراً مطلوباً جداً لنا، فإن لم نجد فإننا ندعوا الله تعالى بأن يُعيننا على مشوارنا الطويل.

2 عن أنس بن مالك قال: (قال رجل: "يا رسول الله: اعْقَلْهَا وَتَوَكَّلْ أَوْ أَطْلِقْهَا وَتَوَكَّلْ" قال: "اعْقَلْهَا وَتَوَكَّلْ"). قال أبو عيسى وهذا حديث غريب من حديث أنس لا نعرفه إلا من هذا الوجه وقد روى عن عمرو بن أمية الضمري عن النبي - صلى الله عليه وسلم - نحو هذا.

و هذا الحديث أخرجه "الترمذي" برقم (4415)، وحسنه "الألباني" في (تخریج مُشكلة الفقر و كيف عالَجها الإسلام) - (1 / 23).

فإذا ما وجدنا بالفعل شيئاً قد أُنتج من قبل و يُمكننا الاستفادة منه فإننا حينئذٍ سنحاول ضم صاحب المُكوّن إلى مجموعتنا العلمية؛ و ذلك لأنه أكثر الناس علماً بالمُكوّن المعنى و سيؤقّر علينا وقتاً كبيراً حينما ينضم إلينا ليعمل على تطوير مُكوّنه؛ فيتفرغ الباقون للعمل على باقى الأشياء التى تحتاج إلى الإنتاج فى المرحلة الحالية، فإن كان هناك ما يَمنع حدوث هذا فإننا سنقوم بتفريغ أنفسنا لتطوير المُنتج المقصود، و هى حالة أفضل من البدء من الصفر على كل حال.

فإن لم نجد فالبداية من الصفر ستكون هو الحل الوحيد أماناً، و لن يكون هذا دافعاً للإحباط لدينا على أية حال؛ فالأمر الطبعى أن المشوار الذى سنقطعه ليس طريقاً مفروشا بالورود تملأه الحماسة و الضياء كما قد يظن بعض الشباب ممّن يُعميهم الحماس عن رؤية الواقع من حولهم، فهناك من المُعرقلات الكثير و الكثير.

و لن يكون غض النظر عن هذه المُعرقلات و عن أخذها بعين الإعتبار أمراً حسناً، بل سيكون حجر عثرة يُوقف المسيرة و يدق عنق من لم ينتبه إليه، لذا فالحكمة كل الحكمة فى التنبه لكل الأخطار و النظر لها بجديّة و التحسب لها بكل ما يُمكن من أساليب؛ ففى مشوارٍ صعبٍ طويلٍ كمشوارنا لن يكون هناك الوقت أو الذهن الشاغرين عندنا لترك ما بأيدينا من مهامٍ لحل هذه المشكلة أو التحسب لعواقب تلك أثناء العمل الفعلي.

و الوقت الذى يُمكننا أن نرتّب فيه أوراقنا هو الوقت الأول فى عمر المسيرة؛ ففيه سيكون الجميع مازالوا على بر العمل و لم ينخرطوا فيه بعد، و سيكون من المُمكن التنبه للمشاكل و وضع العدة لحلها قبل أن تطفو على سطح الواقع، و فى نظرى فإن التجهيز الأوّل لتلك المشاكل ينقسم إلى النقاط التالية:

- التوضيح الكامل المُفضّل لكل الأمور التى تتعلق بالمسيرة من قريبٍ أو بشكلٍ مُباشر، بل لو كانت هناك بعض الأمور التى تتعلق بالمسيرة بطريقةٍ غير مباشرةٍ و كان هناك الوقت الكافى للتحدث عنها؛ فإن العقل السليم يحض على حسم أمرها فى البيانات الأوّل من زمن الإعلان عن الحركة العلمية.

- الرد الإستباقي على أية اعتراضاتٍ نرى أنها ستخطر على بال من يسمع عن المسيرة الجديدة فى أى جانبٍ من جوانبها، و إن كانت هناك اعتراضاتٌ موجودةٌ بالفعل من قبل؛ فالرد عليها بكل وضوح يجب أن يسبق الرد على الاعتراضات التى يُتوقع صدورها فيما بعد، و هنا يجب ألا نخشى التكرار فى الرد على الاعتراضات، أى أن بعض الاعتراضات ربما يكون قد تم الرد عليه بشكلٍ غير مباشرٍ أثناء الحديث عن المسيرة (فى مواقعٍ غير موقع الرد) و لكن هذا لا يعنى أنه يُمكن الإكتفاء بما قيل من قبل، بل يجب أن يتم الرد بشكلٍ مُباشرٍ على الاعتراض و لو كان فى هذا بعض التكرار و الإطناب؛

فهناك من الناس مَنْ لا يُمكنه التنبه إلى أن الردود على الإعتراضات تكمن فى الشروح السابقة.

و هناك من لا يَقتنع إلا بالكلام المُباشر الصريح و ليس الكلام الذى يَجمع بين أكثر من هدفٍ فى ثناياه، و غيرهم و غيرهم، فلكل هؤلاء يجب أن تكون الردود على الاعتراضات مُفردةً فى مكانٍ واحدٍ و غير مُتأثرة هنا و هناك حتى يرى الأمور واضحةً أمامه.

• التنبه إلى الجوانب القانونية و/أو الدينية و/أو الإجتماعية إن كانت موجودةً، و أخذ الحيطة لها و دراستها بتأنٍ؛ لأن مثل هذه الجوانب يكون لها أخطر الآثار فيما بعد عند عدم أخذ الإحتياط الواجب لها.

• التنبه إلى الجانب التمويلى و الإدارى من المسألة؛ فمشروعٌ نهضوىٌ كبيرٌ كهذا المشروع الذى نريدُه يجب أن يُدار بكل حكمةٍ و دقةٍ حتى لا يكبو لأخطاءٍ إداريةٍ كان يُمكن تلافيها، فنخسر كل شىء، و خاصةً و أن هناك من سترصد له و يقف فى وجهه عنوةً. و كذا فإن الجانب التمويلى من أخطر الجوانب فى المسألة و ينبغى حسمه منذ البداية بكل وضوح.

و سيراً على هذا النهج الذى نبهتُ إلى أهميته سيكون سيرنا فى مشوارنا، و ما هذه الرسالة المُباركة إلا المثال العملى لكل ما أوردته؛ فقد كتبت لتكون مرجعاً شاملاً شافياً يَغطى كل الجوانب الهامة التى تخص المشروع العلمى الجديد الذى أرغب فى طرحه و تنفيذه على أرض الواقع.

عن المستقبل

كانت الخلاصة فيما وصلنا إليه في مقدمة هذه الرسالة أن الأمة الإسلامية تحتاج إلى نهضة علمية حقيقية تكفل لها استرداد مكانتها بين أمم الأرض، على شرط أن تكون تلك النهضة قد تم الترتيب لها بشكل عميق يُغَطِّي كافة النواحي المتعلقة بها، مع وضع جدول زمني يُوَضِّح الخطوات التي يجب أن تسير عليها في تالي أيامها فلا تكون عرضة للتخبط و التشتت.

و أن يقوم على أمر هذه النهضة من يمتلك علماً جيداً بمجالها، فيضع لها جدولها الزمني و يبني الأساسيات الأولى لها ثم يدعوا الناس لها بعد الوصول فيها إلى مرحلة ليست بدائية، ثم يقوم ذلك الشخص بالتعاون مع من انضم إليه بتنسيق العمل فيما بينهم لتحقيق الجدول الزمني على أرض الواقع. هذا إن ظل الجدول الزمني كما هو بعد انضمام المنضمين الجدد إليه؛ حيث أن انضمام الكثيرين من أصحاب مجال معين كان قد وُضع في ترتيب متأخر في التنفيذ قد يُغري بتقديم تنفيذ ما يتعلق بذلك المجال زمنياً لتوفر القوة البشرية الخاصة به، و غير هذا كثير.

و ذكرنا فيما سبق أيضاً القول بأن أكثر العلوم قابلية للزرع في الأرض الإسلامية هي العلوم البرمجية؛ لما تتمتع به من مُتطلِّباتٍ ماليةٍ أقل من التي تتطلبها باقي العلوم الأخرى، و خاصة بعد نهوض حركة المصادر البرمجية الحرة التي وُقرت لكل من هم على شاكلتنا ما يلزمهم من أدوات النهضة البرمجية بدون أي مقابلٍ مادي.

فتوجد أنظمة تشغيل `operating systems` و مترجمات `compilers` و مفسرات `interpreters` و منقحات `debuggers` و غيرهن من الأدوات من المُندرجات تحت تصنيف

البرمجيات الحرة free software. و يُمكننا أن نستخدم كل ذلك للوصول إلى الهدف الذي نرغبه من تحقيق النهضة العلمية بحيث لا تكلفنا إلا أقل القليل مادياً حتى لا يُزعجنا أمر التمويل. وكذا لتوافر العنصر البشري من المُبرمجين والخبراء في مختلف المجالات البرمجية (و إن غلب عليها المجالات ذات الطابع التجارى).

و ما دام الأمر هكذا فإننى أجد أن الوقت مُناسبٌ للدعوة إلى مشروع أُريد من الجميع المشاركة فيه معنى لتحقيق ما نتمناه لأمتنا و ديننا علمياً، فالمشروع الجديد الذى أطلقت عليه اسم (البرمجة بإبداع) هو النموذج الذى وضعته لكل ما تحدثت عنه فى أمر النهضة البرمجية الإسلامية. و خلاصة المشروع و الغاية منه هما: تزويد الأمة الإسلامية بكل الأساسيات التى تخص علوم الحاسب البرمجية باللغة العربية و على كل المستويات، بدءاً من ترجمة أهم المنشورات التى تخص أساسيات مجال البرمجيات، و وصولاً إلى إنتاج لغات البرمجة العربية الحرة المُختلفة التى تُنافس اللغات الأجنبية فى قوتها و نضجها، و ليس انتهاءً بصنع نظم التشغيل العربية الحرة.

غايات المشروع:

و تفصيل غايات المشروع فيما يلي:

- **ترجمة و تأليف المؤلفات الخاصة بالعلوم البرمجية الأساسية إلى اللغة العربية.** و أقصد بالعلوم الأساسية: تلك العلوم المُتعلقة بـ: تصميم لغات البرمجة، و إنتاج المُترجمات و المُفسرات و المُنقّحات و كافة الأدوات البرمجية الخاصة بها، و وصولاً إلى إنتاج بيئات البرمجة المتكاملة integrated development environments. بل و وصولاً إلى ترجمة كل ما يتعلق بتصميم و إنتاج نظم التشغيل المُختلفة. و لا أعني هنا الترجمة من اللغة الإنكليزية فقط. بل يجب الإهتمام بترجمة كل المؤلفات الهامة التى تختص بالمجالات المذكورة بالأعلى (أو أي مجالٍ نهتم به فيما بعد) و إن لم تكن باللغة الإنكليزية، فلو كانت هناك رسالة دكتوراة أو ماجستير متميزة و كانت افتراضاً باللغة الألمانية فيجب أن يتم ترجمتها، إلا أنه نظراً لأنه من المُعتاد أن يقوم المُتخصصون في المجال التقني باستخدام اللغة الإنكليزية فلا أظن أن مثل هذه الحالة سوف تحدث كثيراً. كما أن الوصول إلي مرحلة تتبع الأبحاث المُتميزة علمياً بلغاتٍ غير الإنكليزية سوف يأتي في مراحلٍ غايةٍ في التقدم في المشروع، و حينما تترسخ أقدامه و يكون له مصادر تمويله الخاصة و كوادره العلمية المُتفرّعة.

- **إنتاج مُنتجاتٍ عربيةٍ تخص الأدوات الرئيسة فى المجال البرمجى،** بدءاً من لغات البرمجة العربية بمُختلف أنواعها و انتهاءً بنظم التشغيل العربية بمُختلف أنواعها.

الدوافع:

و الأسباب وراء ذلك هي:

- **عامل تقوية للغة العربية:**

و ذلك باستخدامها فى مجال العلوم البرمجية بتوسع بالغ. و الغاية من ذلك جعل العربية جزءاً لا يتجزأ من العلم البرمجى؛ لأن الإعتما د على الترجمة بدون نقل العلم نفسه إلى اللغة و الإسهام فيه باستخدامها ليس بالأمر الذى يحفظ اللغة من الإنهيار، بل يؤخر حدوث ذلك فقط، و ما نراه من تفضيل الكثيرين لقراءة الكتب الإنكليزية على الكتب العربية (و إن كانت تلك الكتب العربية أفضل من حيث الشرح و كمية المعلومات) يرجع أول ما يرجع إلى النظرة الإنبهارية التى ينظرون بها إلى الكتب الإنكليزية عامة، و النظرة الدونية التى ينظرون بها إلى الكتب العربية عامة، بغض النظر عن المحتوى و المستوى العلمى.

و يرجع هذا بدوره إلى الهزيمة النفسية التى يعانون منها؛ فهم يرون أن العلم ينبع حيث ينبع من أراضٍ غير عربية و بأيدٍ غير عربية و بلغة غير عربية، فمن الطبعى هكذا أن ينظروا للعربية على أنها لا تصلح وعاءاً للعلم، و أن ينظروا للغة الإنكليزية التى يكتب بها أهم العناصر المُمثلة للبرمجة (و هو الكود نفسه) على أنها لغة العلم الحقيقية، و الكتب التى تكتب بلغة العلم لابد من أن تكون هى الأفضل علمياً!

و رغم أن الكثيرون الآن يرون أن الترجمة و تأليف كتب الشروحات العلمية باللغة العربية أمرٌ جميل (و يشهد على انتشار هذه النظرة كمية الكتب التى يؤلفها العرب بالعربية، و التى أصبحت تغطى مجالات علمية برمجية كثيرة بالفعل و بكميات ليست بالقليلة)، أقول: رغم هذا فإنه كما أسلفنا القول لا يكفى للحفاظ على اللغة العربية كما يفعل إنتاج العلم من أساسه بها؛ فنحن حينما نترجم نحتاج دائماً إلى إخبار القارئ بالمرادفات التقنية الأصلية فى الإنكليزية حتى يستطيع معرفة معنى المصطلح الإنكليزى متى ما رآه، و بالتالى يكون من السهل عليه قراءة و فهم محتوى الكتب الإنكليزية، كما أنه لا بد له من مطالعة تلك الكتب الإنكليزية و لو بعد حين، بل و لا بد له من التوسع فى ذلك الإضطلاع إذا ما أراد متابعة الجديد أولاً بأول، و كذلك إذا ما أراد التعمق فى جزئية من البرمجة لم يهتم بها أحدٌ من المؤلفين بالعربية من قبل، أو كان الاهتمام بها ضعيفاً و لا يُعطيه ما يحتاجه من تعليم مُتعمق أو مُحدّث.

أما حينما تُنتج العلم بلغتك و تُطور فيه مُستخدماً إياها فإنك تكون حينئذ قد رسّخت أقدام لغتك فى المجال البرمجى بأقوى السبل؛ لأنك حينما تُنتج علماً بلغتك و تؤثقه بها لن يكون على الراغب فى تعلمه من بنى جلدتك استخدام لغةٍ أخرى فى مُطالعتة و/ أو نقده و/ أو تطويره، و بالتالى ستكون كل الألفاظ التى تُستخدم من ثوب اللغة نفسها لا رقعاً من لغةٍ أخرى تُصق بلغتك لصقاً.

- **تنشئة أجيالٍ مسلمةٍ و/أو عربيةٍ قادرةٍ على البرمجة منذ الصغر:**

لوجود لغة برمجة بلغة دينها و/أو لسانها، و الفارق كبيرٌ للغاية بين أن يتم تدريس البرمجة للأطفال بلغتهم مع البرمجة بلغة برمجة أجنبية (إنجليزية بطبيعة الحال) و بين أن يكون التعليم و اللغة البرمجية نفسها عربيين مئة بالمئة.

و أنا أقول هذا الكلام و أنا من الأجيال التي عانت و تعانى حتى الآن من جرّاء كون العلم البرمجي ليس علماً أصيلاً في الثقافة العربية، و كل ماله علاقة بالإحتراف في المجال البرمجي هو من إنتاج غير المسلمين/غير العرب و باللغة الإنكليزية (إلا فيما ندر)، و ليس للعربية نصيبٌ من التأليف العلمي إلا القليل، و ما كان من ذلك التأليف إضافة حقيقية إلى العلم هو أقل القليل. لذا فأنا و جيلي كله (كما الأجيال السابقة) مُجبرون على تحصيل العلم من مظانّه بلغة غير لغتنا، و لا أخشى أن أقول أن الأمر لم يكن سهلاً أو مُبهجاً على الإطلاق بالنسبة لي؛ فقد كان على أولاً أن أعتاد على اللغة الإنكليزية كلغة تحصيل علمي، و أن أحفظ من تعبيراتها العلمية و مُصطلحاتها التقنية ما أفدر عليه، و حتى بعد أن فعلتُ هذا كان على أن أعوّد نفسي على قراءة الكتب الإنكليزية لفتراتٍ طويلة حتى أحصل من العلم أقله؛

فالواقع أن الساعة الزمنية التي تكفيني لمُطالعة كتابٍ عربي كبير الحجم و فهمه فهماً شبيه كامل لا تكاد تكفيني لمُطالعة فصلٍ واحدٍ في كتابٍ إنكليزي سهل اللغة قريب الأسلوب، فما البال بالكتب التي ما ألّفها مؤلّفوها إلا و كان التحذلق اللغوي فوق أطراف أنوفهم؟!!

و كلما وجدتُ نفسي في موقفٍ سخيّف يفرض عليّ أن أطلع كتاباً إنكليزياً لغته (بالنسبة لي أنا المغلوب على أمرى) كطلاسم السحرة: كرهتُ الوضع الحالي أشد الكره، و كرهتُ أكثر منه مَنْ كانوا يستطيعون التغيير فيه و لم يفعلوا (بل و ربما حتي لم يُحاولوا)، و كرهتُ أكثر من كل هذا من تُبَطّوا همم من حاول التغيير و كانوا سبباً في فشلهم بشكلٍ مُباشرٍ أو غير مُباشرٍ.

و عن تجربةٍ حقيقية أقول أن العلم إذا أصبح بلغتنا كان بإمكاننا الإبحار فيه بمنتهى البساطة و الوضوح، و سيكون تركيزنا على التحصيل الحقيقي للعلم و لن نُضيع الجهد و الوقت الغاليين في تحصيل أدواتٍ (مثل اللغة الأجنبية) ليست مرغوبة في حد ذاتها، بل إن الهدف منها فقط أن تُمكننا من تحصيل العلم نفسه.

و سيكون الأمر بالنسبة للصغار أسهل من شرب الماء القراح؛ فنحن حينما نريد تعليم الطفل الصغير البرمجة بلغة برمجة أجنبية سيكون علينا أولاً: أن نجعله يفهم أن هذه لغة أخرى كلغتنا العربية، و لكن يُمكن للحاسوب فهمها إذا ما كتبت على نحوٍ معين و بألفاظٍ مُحدّدة، و بعد عناء إيصال ذلك المعنى له (و هو الأمر الذي يكاد يكون مُستحيلاً على الأطفال غير الإنكليز) فسيكون أمامنا أن نجعله يعي معنى كل كلمة إنكليزية (تُستخدم في لغة البرمجة) في الحياة العادية و معناها في البرمجة حتى يستوعب الفكرة جيداً (و هو الأمر الأقرب للمُستحيل فعلياً على الطفل الصغير).

و بالتالى لا يكون أمام من يُعلِّمه إلا قول (اكتبها كما هي) و (احفظها و انته) و (ستعرف معناها فيا بعد و ليس الآن).

أما حينما نستخدم لغة برمجةٍ عربيةٍ فإن الأمر لن يُكلِّفنا إلا قول جُملةٍ واحدةٍ: (يا صغيرى: الحاسب لا يفهم إلا تعبيراتٍ مُعيَّنة هي كذا و كذا و كذا).

و بالطبع لن نحتاج إلى شرح المعنى الحياتى لهذه التعبيرات؛ لأن الطفل: يستعملها بالفعل، أو يسمعها، أو هى قريبةٌ للغاية مما يستعمل أو يسمع، فالجُمَلُ الشرطية كجملة **لو** (التي سيلي شرحها في أثناء شرح قواعد اللغة) على سبيل المثال لا تحتاج إلى أى شرح كجملة **if** التي نحتاج لإفهام الطفل معها معنى كلمة **if** و **else** أوّلاً.

و هكذا يُمكننا التركيز على تعليم الأطفال فكرة البرمجة لا تحفيظهم الكلمات المحجوزة الخاصة بلُغة البرمجة، و سنرى أن الأطفال سيتسابقون (فيما بعد) فيما بينهم بالألغاز البرمجية و حلولها، على العكس من الحالة الثانية التي سيتسابقون فيها بحفظ التلخيصات و أمثلة البرامج و القواعد لسكبها على ورقة الامتحان ثم نسيانها تماماً بعد ذلك !

• إعطاء الفرصة لربط المسلمين غير العرب باللغة العربية بشكل أكبر:

من خلال تحويل العلوم البرمجية إلى اللغة العربية فتُصبح اللغة العربية بالنسبة لهم لغة دين و تخصص علمي في ذات الوقت، و هو المدخل المُتميز لربطهم بالدين بصِلَةٍ وثيقة؛ حيث أن المسلم غير العربي على كاهله من الهموم الكثير، مما يشغله عن تحصيل العلم الشرعى و تعلم العربية (لغة الإسلام) إذا كان من أهل العلوم الحياتية؛ حيث سيكون عليه أن يُحصَل اللغة الإنكليزية (اللغة الأكثر انتشاراً فى مجال العلوم الحياتية) و العلم الحياتى نفسه، إضافةً إلى جهده الذى يصرفه فى و صل رحمة و بقية أمور حياته الخاصة. فلا يتبقى له من الوقت شيءٌ للإهتمام بعلوم الشرع إلا على أخف الأوجه و أشدها ضرورة.

و لو أننا حَوَّلنا العلم البرمجى إلى علم أصيل فى اللغة العربية و جعلنا لها قدماً راسخةً فيه لكان بإمكان المسلم غير العربى استغلال ذلك أفضل الاستغلال بتعلم اللغة العربية واستخدامها كلغة تحصيل علمي للعلم الحياتي و كلغة دين، فلا يخسر من الجهد شيئاً و كما يقولون (يصطاد عصفورين بحجر واحد).

و هذا الهدف يُعد من أول الأهداف التي يجب أن يسعى لها كل راغب فى النهضة بالأمة من جديد؛ فليس المسلمون كلهم من العرب، بل إن أغلبهم من غير العرب، و على أكتافهم كما على أكتاف المسلمين العرب تقع مسؤولية النهضة بالأمة من جديد، و مادامت اللغة العربية تُشكّل أمراً ضرورياً لا يتجزأ من الحياة الإسلامية فيجب أن يُبذل أقصى الجهد فى سبيل تقويتها عند غير

العرب و جعلها لهم كما كانت بالنسبة لمُسلمي الزمان الأول من العجم كالإمام محمد بن إسماعيل البخاري (صاحب: صحيح البخاري) رحمة الله عليه قضية غير قابلة للنقاش أو الجدل حولها من الأصل ناهيك عن التخلي عنها.

• عاملٌ نفسى هامٌ للغاية:

يجعل من الحديث عن النهضة الشاملة و استرجاع مكانتنا الحضارية فى كل الميادين الحياتية أمراً قوياً الحجة إلى أقصى الحدود؛ فمادام الأمر قد نجح فى علم من العلوم (و خاصة إذا كان علماً فى غاية التطور مثل علوم الحوسبة) فهو قابل للنجاح فى كل العلوم، فقد أثبت أنه لا لغة للعلم و أن المهم هو الإنتاج العلمى لا اللغة التي يتم بها ذلك الإنتاج، على العكس مما يقوله دجالو هذا العصر من أن العلم: غربى الهوية، إنقليزى اللغة، و أن تغيير ذلك من المُحال. و بدلاً من المجال البرمجى فقط سنجد أن الإنتاج العلمى بالعربية قد توغل فى باقى الميادين العلمية بدون استثناء، و رُويداً رُويداً سيأتى الزمن الذى يستنكر طالب العلم فيه أن يطلب العلم بلغة غير اللغة العربية، و سيقول رداً على هذا كلمة سبذل الغالى و الرخيص يا ذن الله عز و جل لأسمعها من كل الناس "أنا أريد تعلم العلم الفلانى لا اللغة الإنقليزية!".

• البرهنة على سَفَسَطَةِ الْقَائِلِينَ بِأَنَّ الْإِلْتِزَامَ بِتَعَالِيمِ الدِّينِ ضَدَّ الْعِلْمِ وَ الْحَضَارَةِ:

حيث أن الأجيال المُحتلّة عقلياً (ممن خانوا دينهم و باعوا أنفسهم للثقافة الغربية بضمن بَخْسِ) يستغلون الجهل المنتشر بين أهل الأمة، و الدجل و الشعوذة اللذين انتشرا فى كل مكانٍ لِيُسيئوا للدعوة إلى الالتزام بتعاليم الدين الحنيف أبشع الإساءة.

و دائماً ما يَصَب حديثهم فى هذا المَصَب النجس، و لأنهم انتشروا فى كل مكانٍ و تجدهم بسهولة عند كل زاويةٍ و تحت كل حجر، و لأن الجهل انتشر بالفعل بين عامّة الأمة انتشار النار فى الهشيم: فقد لاقت مزاعمهم أرضاً خصبةً و عقولاً تصدّق و تؤمن. و أصبح الرد على كل من يأمر بمعروفٍ أو ينهى عن منكرٍ: أن "كُفَّ عن التخلف"، و هو ما أوّمن أنه قيل لكل مُلتزمٍ حتى و إن أمر بأوضح المعروف و نهى عن أشنع المنكر!

و أصبح شوكة فى حلق كل ملتزمٍ أنه يُوصم دون أى جنائيةٍ أو تحقُّقٍ بأنه ضد العلم و الثقافة و الحضارة، على الرغم من أن أغلب من فى معاشر المُلتزمين و أغلب رؤوسهم و مشايخهم هم من أهل العلم و الفضل فى العلوم الحياتية قبل العلوم الشرعية!، و لكن الكره لا يدع للعدالة جانباً و الجهل لا يدع للتحقيق و التأكد مكاناً.

بل و أصبح من سَمَت أهل العلم و الدراية بالعلوم و الحضارة (عند كثيرٍ من الناس) الإبتعاد فى الهيئة و القول عما يُشابه هيئة و قول أهل الالتزام، و أصبح الناس عندهم قِسَمين: مُلتحين مُتخلفين، و غير مُلتحين مُتحررين تقدميين. و القول بخلاف ذلك عندهم هرطقةً و جدلاً فارغان!

لذا فإن مثل تلك الدعوة للنهضة تُعدُّ بمثابة الرد العملي على تلك الدعاوى البغيضة، فإذا ما أمرنا بعدها بمعروفٍ أو نهينا عن مُنكرٍ و اتَّهَمنا بالتخلف: كان بإمكاننا ترك البرهان النظري و البرهنة بالواقع على العكس، و أن أنفع الناس للأمة هم المُلتزمون فيها بتعاليم ربهم جل و علا، و أن أشد الناس إكثاراً في الحديث بدون جدوى و تبيطاً لهمم أهل الحَلِّ و العَقْدِ هم مَنْ نا صبوا أهل الدين العدا و وقفوا في وجوههم.

• هدفًا شخصيًّا صرفًا:

وهو رغبتى الذاتية في تحصيل علوم الأساسيات الحاسوبية بشكل عملي إحتراقي، و أنا أضع هذا الهدف هنا لأنه من أهم الأهداف التي تُحرك هذه المسيرة علي الإطلاق، و من خلاله يُمكن تفسير كثير من القرارات التي تبدو غريبة للوهلة الأولى، مثل قرار بناء كل شيءٍ من الصفر و عدم الإستعانة بأي مُنتجاتٍ أخرى كما سيلي ذكره.

و كذلك العمل الدؤوب لعامين كاملين بدون إخبار أحدٍ بحقيقة المشروع (اللهم إلا قلة تُعد علي أصابع اليد الواحدة بحكم الظروف فقط)، و الإصرار علي المُبالغة في بعض الأحيان فيما يجب: دراسته و/أو عمله و/أو التخطيط له في مُختلف مراحل المشروع، و في الأوقات القادمة من حياة المشروع سوف يظهر هذا في بعض القرارات الغريبة (مع أنها لن تكون مُستنكرة).

و دعوني أسترسل هنا بعض الشيء لأنني أريد إيصال رسالة مُعيَّنة إلي أشخاص بعينهم: فبالنسبة لي فإن أبشع كوابيسي هو أن أكون علي فراش الموت في التَّزَع الأخير، و أنظر إلي حياتي الماضية فأجد أنني جئت إلي الدنيا كأبي حيوانٍ أعجم: أكلتُ، و شربتُ، و نمتُ، و قضيتُ حاجتي، ثم متُّ ! و بعد موتي أصبحتُ كأبي لم أكن !

إن كابوسي الأشنع هو أن أموت بدون أن يكون لحياتي أي معنيٍّ أو أثر، و لا أقصد الشهرة هنا؛ فكم من أناسٍ أُنزوا في حياتنا بشكلٍ كبيرٍ جدًّا و لا نعرفهم و لا نذكر أفضالهم، إنما أقصد بالكابوس أن أكون عاديًا في ولادتي و حياتي ثم مماتي، أن أكون رقمًا مُضافًا إلي تعداد الناس لا يُحدث فرقًا إن كان بالنقص أو بالزيادة.

ما أريده حقًّا (و الشيء الوحيد الذي أرتضيه لنفسي و أجده يليق بي) هو أن أكون علامةً فارقةً في المجال الذي أُنخِص فيه،

أن أصير مرَّجعًا علمياً يسير علي قدمين و يُعد كلامي من الأدلة التي تفصل بين المُتنازعين في المسائل العلمية،

بل أريدُ أن أكون مُقتحماً لأراضٍ في المجال العلمي لا يجرؤ علي اقتحامها الكثيرون،

بل أريدُ أن أكون فاتحاً لأراضٍ لا يعلم عنها أغلب الناس (إن لم يكن كلهم) شيئاً.

و من وجهة نظري: فهذا ليس بخارقٍ لطبائع الأمور؛ فما دام المرء بعيداً عن الغرور و الانتقاص من قدر غيره فيمكنه أن يحلم بما يريد، و أظن أنني أتعلم الدروس التي لا بد لي من تعلمها ما دمتُ قد سلكتُ هذا الدرب درسا تلو الآخر.

و ما يهمنا هنا: أن هذا المشروع هو أفضل السُّبل علي الإطلاق لإرضاء طموحي (هل أقول: جسعي؟). و يُمكنني من خلاله تحقيق كل تلك الأحلام التي تكاد تُفجِّر مخي و أعصابي تفجيراً، كما أنه أيضاً فرصة لكل من هم علي مثل شاكلتني لكي يُحاولوا معي تحقيق أحلامهم علي أرض الواقع.

الإعترضات الموجهة لإنتاج لغة برمجة عربية و الرد عليها

ما دام إنتاج لغة برمجة عربية هو الهدف الأول الذي يجب علينا أن نحققه لأجل نهضة علمية برمجية حقيقية كما أسلفنا الذكر، فإنه يجب علينا أن نركّز علي أية اعتراضات موجودة علي هذا الهدف؛ فكما أوضحنا من قبل فإنه يجب علي القائمين علي شأن أي مسيرة علمية ذات تخطيط جيد أن يقوموا بتوضيح كافة ما يتعلق بها منذ البداية و أن يقوموا بالرد علي أية اعتراضات تُوجد. و قد شرحتُ أسس مشروع (البرمجة بإبداع) بإيجاز من قبل، و حان الآن وقت الرد علي الاعتراضات التي تُواجه الهدف الأول و الأهم و هو إنتاج لغة برمجة عربية احترافية.

و قد جمعتُ هذه الإعترضات كمُعظم البيانات التي أعتمد عليها في هذا الكتاب من شبكة المعلومات الدولية، و على الأخص من المنتديات التقنية و المواقع التي كانت تُثار بها قضية إنتاج لغة برمجة عربية فيُسهم الجميع إما موافقة لهذا الأمر أو رفضاً له.

فكثيرون كانوا يطرحون هذا الأمر على باقي المشاركين في المنتدى أو الموقع إما لمناقشته من حيث المبدأ في حد ذاته، أو حتى على سبيل دفعهم للمشاركة معهم في إنتاج لغة برمجة عربية بدأوا أو سيبدأون في إنتاجها بأنفسهم، أو حتى لمناقشة محاولة تتم حالياً سمعوا عنها من غيرهم أو من خلال شبكة المعلومات.

المهم أنه في كل الأحوال فإن المشاركين في المناقشات لا يخرجون عن إحدى المجموعات التالية:

• الأولي:

وهي مجموعة المُتحمسين للأمر بطريقةٍ كبيرةٍ للغاية قد تصل إلى حد التعصب الأهوج، وهو ما قد يدفعهم إلى الإحتداد الكبير أثناء مناقشة الأمر مع الراضين له، و أسلوب خطابهم يشف عن عدم تحملهم لوجود من يرفض المسألة برمتها أو على الأقل سماع أقوال ذلك الراض. وأنا (و إن كنتُ أوافقهم في أصل المسألة) فإنني في معظم الأحيان أرفض الإحتداد الكبير الذي يُصاحب كل خطاباتهم، وأقول في معظم الأحيان لأن هناك أوقات لا يكون مُمكنًا بالنسبة لي (مثلهم تمامًا) أن أضبط أعصابي و عندها يكون تَهَيُّج الأعصاب و زراً يقع على كاهل المُتسبب فيه و لا يكون عليّ أو عليهم فيه أى ذنب.

• الثانية:

وهي مجموعة الراضين للأمر بطرقٍ ربما تصل بدورها إلى حد التعصب، و يُصاحب أقوال كثيرٍ منهم بدورهم حِدَّةٌ كبيرةٌ و سخريَّةٌ مريرةٌ أثناء مُناقشاتهم للمُوافقين و المُتحمسين للأمر، و رغم إصابة بعضهم في بعض إعتراضاتهم على الأمر إلا أن الأغلبية منهم لا يُعبرون عن إعتراضاتٍ قوية، بل عن إعتراضاتٍ في قمة الضعف و الهزال، و لا تُعبر إلا عن رفضٍ نفسي قاطعٍ منهم للمسألة من حيث المبدأ و محاولة إيجاد أي مُسوِّغٍ منطقيٍّ لذلك الرفض !.

• الثالثة:

و هم قليلٌ من الناس من المُتوسِّطين في رفضهم أو قبولهم للأمر و يُحاولون مُناقشة الفرضيات المُختلفة بعقلانية، و العقلانية هنا لا تعني التخلي عن التحمس لتأييد أو رفض الأمر بل تعني الإنصاف في التناقش و الإعتماد علي البراهين و الأدلة بحيادية تامَّة. و لا أعني بقلة العدد أن عددهم الحقيقي قليلٌ بل أعني أن عدد من يظهر منهم في مثل تلك المُناقشات الحامية هو القليل، كما أن توسطهم هذا لم يُبرز أقوالهم في النقاشات التي دارت و تدور في هذا الموضوع، بل كانت السيطرة على النقاشات (كالعادة) لأصحاب المجموعتين السابقتين لحماستهم الزائدة التي كانت تدفعهم للكتابة دفاعاً أو هجوماً، و لصوتهم العالي و الأسلوب الساخر الذي لا يهتم صاحبه لا بدليلٍ و لا ببرهانٍ و الذي أُطلق عليه اسم (أسلوب تطليع اللسان).

و المُلاحظ أن أغلب العرب في نقاشاتهم الدينية و/أو السياسية و/أو التقنية يكونون أقرب ما يكون لفريقين من الديكة المُتنافرة، و يكون التعصب عندهم لصالح فريقٍ ضد فريقٍ آخر و إستعمال التهكم و السباب أمراً طبعياً للغاية، و لتذهب البراهين و الحجج العقلية إلي حيث ألتقت !.

و فيما يلي أطرُحُ الإعتراضات الرئيسة التي أظن أن باقى الإعتراضات تنفرع منها، و قد قمتُ بالتنسيق فيها قدر الاستطاعة بحيث تخرج سليمة اللغة و في نفس الوقت تُعطى ذات المعنى و الإحساس للذنان

تُعْطِيهِمَا كَلِمَاتِ الْمَعْتَرِضِ الْأَصْلِيَّةِ، وَ لَمَّا كُنْتُ قَدْ فَشَلْتُ فِي مَعْظَمِ هَذِهِ الْمُحَاوَلَاتِ فَإِنِ التَّغْيِيرَاتِ الَّتِي سَيَجِدُهَا الْقَارِئُ سَتَكُونُ طَفِيفَةً لِلغَايَةِ إِنَّ وَجِدْتَ مِنَ الْأَصْلِ.

وَ قَدْ أَدْرَجْتُ إِعْتِرَاضَاتٍ تُمَثِّلُ الطَّيْفَ الْكَامِلَ لِكُلِّ الْإِعْتِرَاضَاتِ الَّتِي قَرَأْتُهَا فِي الْمُنْتَدِيَّاتِ الْمُخْتَلِفَةِ، وَ لَمْ أَلْتَزِمُ بِأَنْ تَكُونَ كُلُّهَا مِنَ الْإِعْتِرَاضَاتِ الْوَجِيهَةِ: بَلْ فِيهَا إِعْتِرَاضَاتٌ مِنَ النَّوْعِ الْهَزِيلِ الضَّعِيفِ الَّذِي لَا يُمَكِّنُنِي أَنْ أَهْمَلُ وَجُودَهُ وَ الْحَاجَةَ إِلَى الرَّدِّ عَلَيْهِ، وَ ذَلِكَ لِأَنَّهُ رُبَّمَا يَكُونُ (رَغْمَ ضَعْفِهِ) قَدْ صَدَرَ عَمَّنْ هُوَ مُقْتَنِعٌ بِهِ، وَ كَذَلِكَ لِمَنْعِ خُطُورِهِ عَلَى خَاطِرِ أَحَدٍ آخَرَ فِيمَا بَعْدَ فَتَنُوكُنْ قَدْ كَفِينَا أَنْفُسَنَا مُؤَنَّةَ الرَّدِّ عَلَيْهِ مُسْتَقْبَلًا.

وَ مِنَ الْبَدْهِىِ طَبَعًا أَنَّنِي لَمْ أَهْتَمَّ بِذِكْرِ أَسْمَاءِ أَصْحَابِ الْإِعْتِرَاضَاتِ الْمُخْتَلِفَةِ، وَ مَحَوْتُ مَا كَانَ مِنْهَا مَوْجُودًا فِي صَلْبِ الْإِعْتِرَاضَاتِ ذَاتِهَا؛ لِأَنَّ الْأَمْرَ غَيْرَ شَخْصِيٍّ وَ لَا يَحِقُّ لِي أَوْ لِغَيْرِي أَنْ يُحَوَّلَ إِلَى ذَلِكَ، وَ هَذِهِ نَقْطَةٌ فَائِضَةٌ الْأَهْمِيَّةِ أَرْجُو التَّنَبُّهَ لَهَا جَيِّدًا.

1- مَا مَعْنَى أَنْ نَبْدَأَ مِنَ الصَّفْرِ فِي الْوَقْتِ الْمَوْجُودِ فِيهِ لُغَاتٍ بِرْمَجَةٍ مَفْتُوحَةٍ الْمَصْدَرِ ذَاتِ قُدْرَاتٍ عَالِيَةٍ مِثْلَ **java**، لَمَّا ذَا لِانْشَارِكِ (نَحْنُ الْعَرَبِ) فِي تَطْوِيرِ تِلْكَ اللُّغَاتِ مِثْلَ آلَافِ الْعُقُولِ حَوْلَ الْعَالَمِ بَدَلًا مِنَ الْكَلَامِ الْكَثِيرِ.
مَا نَحْتَاجُ إِلَيْهِ فَعَلًا هُوَ عَمَلِيَّةٌ تَعْرِيبٌ لِلْمَرَاجِعِ الْخَاصَّةِ بِالْبِرْمَجَةِ.

الْحَقُّ أَنَّ هَذَا الْقَوْلَ يَعْنِي أَنَّ صَاحِبَهُ يَتَحَدَّثُ عَنِ أَمْرٍ مُخْتَلِفٍ بِالْكُلِّيَّةِ، فَكَمَا أَسْلَفْنَا الْقَوْلَ مِنْ قَبْلِ فَإِنِ النَّهْضَةُ الْبِرْمَجِيَّةُ الَّتِي نَرْغَبُ فِيهَا لَيْسَ الْهَدَفُ الْوَحِيدُ الَّذِي نُوَدُّ الْوَصُولَ إِلَيْهِ مِنْ خِلَالِهَا هُوَ رَفْعُ الْإِسْهَامِ الْعِلْمِيِّ الْإِسْلَامِيِّ فِي مَجَالِ الْعُلُومِ الْبِرْمَجِيَّةِ فَحَسَبَ؛ لِأَنَّهُ إِنْ كَانَ الْأَمْرُ كَذَلِكَ لَكَانَتِ الْمُسَاهَمَةُ مَعَ مُجْتَمَعِ الْمَصَادِرِ الْمَفْتُوحَةِ الْعَالَمِيِّ هُوَ الطَّرِيقُ الْأَمثلُ لَنَا بِالْفِعْلِ كَمَا أَسْلَفَ صَاحِبُ الْإِعْتِرَاضِ الْقَوْلِ، لَكِنِ النَّهْضَةُ لَهَا مِنَ الْأَهْدَافِ الْأَهْمُ الْكَثِيرُ وَ الَّتِي لَا تَحْقُقُهَا هَذِهِ الْوَسِيلَةُ.

وَ هِيَ كَمَا ذَكَرْنَاهَا مِنْ قَبْلُ: تَقْوِيَةُ اللُّغَةِ الْعَرَبِيَّةِ فِي مَجَالِ الْبَحْثِ الْعِلْمِيِّ، وَ رِبْطُ الْأَجْيَالِ الْإِسْلَامِيَّةِ الشَّابَةِ وَ الطِّفْلِ بِاللُّغَةِ الْعَرَبِيَّةِ لِزِيَادَةِ التَّمَسُّكِ بِالْدِينِ الْإِسْلَامِيِّ الْحَنِيفِ فِي كُلِّ أَقْطَارِ الْإِسْلَامِ، ثُمَّ إِشَاعَةُ جَوِ النَّهْضَةِ الْعِلْمِيَّةِ فِي كُلِّ أَرْجَاءِ الْمُجْتَمَعِ الْمُسْلِمِ بِحَيْثُ يَتَنَفَسُ الْكُلُّ صَغَارًا وَ كِبَارًا الْعِلْمَ بِلُغَتِهِمْ وَ يَتَحَوَّلُ تَفْكِيرُهُمْ إِلَى الْمَنْهَجِ الْعِلْمِيِّ الرَّصِينِ، وَ غَيْرِهَا مِنَ الْأَهْدَافِ الَّتِي تَتَرْتَّبُ عَلَى كُلِّ هَذَا.

وَ مُحَاوَلَةٌ تَحْقِيقِ هَذِهِ الْأَهْدَافِ لَا يَدْخُلُ تَحْتَ مَفْهُومِ (الْكَلَامِ الْكَثِيرِ) الَّذِي عَبَّرَ بِهِ الْمُعْتَرِضُ عَنِ شَعُورِهِ حِيَالَ الْكَلَامِ عَنِ تَطْوِيرِ لُغَةٍ بِرْمَجَةٍ عَرَبِيَّةٍ، وَ بِصِرَاحَةٍ فَإِنَّ لَهُ بَعْضَ الْحَقِّ (بَعْضٌ وَ لَيْسَ كُلٌّ) فِي إِعْتِرَاضِهِ هَذَا؛ بِسَبَبِ مَا رَأَى مِنْ قَبْلِ وَ مَا نَرَاهُ كُلِّ يَوْمٍ مِنْ مُهَاتَرَاتٍ وَ شَعَارَاتٍ كَاذِبَةٍ وَ دَعَاوٍ بَرَاقَةٍ لَيْسَ لَهَا مَضْمُونٌ حَقِيقِيٌّ.

فالإحباط الذى يُسببه كل هذا الزخم يجعل صاحبه غير قادرٍ على تصديق أى شئٍ يتعلق بذلك الموضوع، و يصير الأمر بالنسبة له مضيعةً للوقت و الجهد، و هكذا يصير الأفضل هو النظر إلى الأمور التى يُمكن تحقيقها علي أرض الواقع بدون كل أولئك الذين أصابوه بالإحباط و اليأس الشديدين. و لكنه يُخطئ حين يُعمّم هذه النظرة على الكل؛ فالحق أنه يجب أن يُقصرها على المُتحمسين الذين يتحمسون للشئ و نقيضه فى وقتٍ واحدٍ و بنفس الدرجة من الصدق!، من نوعية " صباح الخير يا مان، يلا نعمل لغة برمجة عربى " ثم ينسون الأمر فى لحظات، و إن لم ينسوه فإنهم يُنتجون شيئاً هزلياً مُضحكاً ثم يملأون الدنيا حديثاً عنه بشكلٍ يُثير الغثيان فى نفس من هم على شاكِلة صاحب الإعتراض و كذلك كاتب هذه الأسطر.

أما من يتحدث بعقلانيةٍ و منطقيةٍ و يضع لكل أمرٍ حسابه و عدته فيجب أن يُعامل بطريقةٍ مُختلفةٍ تماماً، فحينئذٍ يجب الإنضمام إلى حركته تلك و مساعدته بكل الإمكانيات المتاحة لنا، أو على الأقل إن لم يكن بإمكاننا المساعدة بأنفسنا: نُساعده عن طريق الدعم المادى و/أو المعنوى و/أو الدعاية لحركته و الترويج لها بين الشباب القادرين على المُساعدة.

2- لغة البرمجة نفسها ليست هدفاً و إنما وسيلة لعمل برامج، انظر إلى الدول الأولى عالمياً فى تصدير البرمجيات ستجد أنها لم تعمل لغات برمجة خاصة بها (الهند الأولى عالمياً فى صناعة البرمجيات لم نسمع فيها عن لغة برمجة هندية ولم يخرج واحد عبقرى طالب بعمل لغة برمجة هندية)، ثم إنه لعمل لغة برمجة محترمة أنت محتاج مجهود سنين وكوادر و تعاون فى العمل بين الأفراد و المؤسسات البحثية و كل هذا والحمد لله ليس موجوداً عندنا، إذاً نعمل بالموجود أم نقول "لأ يا نبدأ من الصفر يا بلاش".

صاحب هذا الإعتراض أصاب و أخطأ فى كلامه؛ لقد أصاب حينما قال أن لغة البرمجة و وسيلةً لا غاية؛ فالحق أن الهدف من إنتاج لغات البرمجة ليس هى بحد ذاتها؛ بل القدرة التى تُتيحها لنا لإنتاج البرامج الأخرى على إختلاف أنواعها و التى تُتيح لنا التعامل السهل مع البيئة المُحيطة بنا فى المنزل و/أو العمل و/أو شتى مجالات الحياة لتُصبح الحياة أيسر و أفضل.

و كذلك حينما قال أن إنتاج لغة برمجةٍ يحتاج إلى مجهود سنينٍ و كوادرٍ و تعاونٍ بين الأفراد و المؤسسات البحثية؛ فإن إنتاج اللغات البرمجية ليس بالأمر السهل كما قد يبدو للمُتحمسين من النوعيات التى تحدثنا عنها قبلاً، و لكى تُنتج لغة برمجةٍ قويةٍ لها مُميزات لغات البرمجة ذات الشهرة العالمية و التى تُتيح لك أن تبنى عليها و على منتجاتها نهضتك العلمية: فإن الأمر سيكون مشواراً طويلاً من التعلم، و الدراسة، و التصميم، ثم التنقيح، ثم البناء، ثم التحديث، و غيرهن من الأمور التى تبدو سهلة القول إلا أنها تُقاس بالأعمار و الأجيال!.

و لا يُمكن لفرْدٍ بمفرده أن يقوم بكل ذلك إلا إذا كان فى قمة الذكاء و العلم و التفرغ (و هى الصفات التى يستحيل توافرها فى شخصٍ واحدٍ فى زمننا هذا)، و حتى إن كان موجوداً فإن هذا الأمر

سيستحوذ على عمره كله ليصل إلى هدفه، و لكي يتم الوصول إلى الهدف في زمنٍ معقولٍ و بالإمكانات المعقولة فإن الأمر يستلزم بحق تعاوناً بين الأفراد و المؤسّسات البحثية للعمل عليه و جعله هدفاً أعلى لها.

هذا ما كان المُعترض مُحِقّاً فيه، أما ما كان مُخْطِئاً فيه فهو تلميحه بأنه يجب علينا أن نعمل بالموجود و ألا نبدأ العمل من الصفر؛ و وجه الخطأ هنا هو أننا بالعمل بما هو متاح لنا (أى بلغات البرمجة غير العربية و إن كانت مفتوحة المصدر) سنحقق هدف زيادة المُساهمة الإسلامية في مجال البحث العلمي، و لكننا سنسأهم في ترسيخ أقدام اللغة الإنكليزية في بلادنا أكثر، و بالتالي في إضعاف اللغة العربية في بلاد العرب و قتلها في بلاد المسلمين الأعاجم. و سنكون كالدب الذي قتل صاحبه (في الحكايات الشعبية) حينما أراد إبعاد ذبابةٍ وقفت على وجه الأخير و أزعجته و هو مُستغرقٌ في نومه! . و الحل كما أفضنا في الحديث هو: البدء بتعلم العلم الموجود حالياً و إتقانه إلى أقصى الحدود المُمكنة، ثم استخدام هذا العلم لبدء طريق النهضة، و بالتالي لن نبدأ من الصفر كما يقول صاحب الإعتراض بل باستخدام العلم (و ليس المُنتج) الذي أنتجه الآخرون، و من ثم نُنقذ و نُصحح لنتّج نحن بأنفسنا لأنفسنا، و هو الطريق الصحيح للإنتاج العلمي المُستقل و الشكل التعاوني مع الآخرين لأجل فائدة البشرية كلها، و الأكثر تناسبا مع احتياجاتنا في ذات الوقت.

و ثانياً هناك نقطة عدم وجود لغات برمجةٍ غير إنكليزية، و هذه النقطة تدل على أن صاحب الإعتراض لم يبحث الأمر كما يجب حينما قال ما قال، بل أطلق العنان للسانه بدون التحقق من حقيقة ما يقول من عددها، فهناك لغات برمجةٍ كثيرةٌ تستخدم مبنيةً على لغاتٍ أخرى غير اللغة الإنكليزية، بل و على أبجدياتٍ أخرى غير الأبجدية اللاتينية من الأصل! . و هاكم قائمةً لأمثلةٍ على كِلي النوعين منقولةً نقلاً من موسوعة ويكيبيديا:

- **Aheui** – An esoteric programming language similar to Befunge but using Hangul (Korean)
- **AMMORIA** – Open source object oriented Arabic programming language, designed especially for Arabs.
- **Analitik** – A Russian-based language for symbolic manipulations with algebraic expressions used in the Soviet series of Mir computers
- **ARLOGO** – The first open-source Arabic programming language, based on the UCB Logo interpreter

- [Chinese BASIC](#) - Chinese-localized BASIC dialects based on Applesoft BASIC; for Taiwanese Apple II clones and the Multitech Microprofessor II
- [எழில், Ezhil programming language](#) - A Tamil programming language developed for educational purposes.
- [Farsi.NET](#) - A Persian (Farsi, فارسی, پارسی) OO programming language for .NET Framework. It is similar to C# and Delphi.
- [Fjölnir](#) - An Icelandic imperative programming language of the 1980s
- [FOCAL](#) - Keywords were originally English, but DEC produced versions of FOCAL in several European languages
- [4th Dimension](#) - On local versions, its internal language uses French or German keywords
- [Jeem ج](#) - Arabic programming language, based on C++ with simple graphics implementation developed by Dr M A Al-Salka Jeem Website.
- [Glagol](#) - A Russian-based programming language similar to Oberon and Pascal
- [GOTO++](#) - A French esoteric programming language loosely based on French and English
- [Hindawi Programming System](#) - Indian language set of equivalents for C, C++, lex, yacc, assembly, BASIC, logo, Ada, and others for languages such as Hindi, Gujarati, Assamese, and Bangla (with the BangaBhasha version)
- [Hindi Programming Language](#) - A Hindi programming language for the .NET Framework
- [hForth](#) - A Forth system with an optional Korean keyword set

- [HPL](#) - Hebrew Programming Language
- [Kumir](#) - A Russian-based programming language similar to Pascal and IDE, mainly intended for educational usage in schools. The name is an acronym, which means *Комплект ученический 'Мир'* ('Mir' student's environment).
- [Lexico](#) - A Spanish OO language for teaching .NET programming
- [Linotte](#) - A French programming language
- [Logo](#) - In one of its Apple II editions was available in French.
- [Loughaty](#) (MyProLang) - A general-purpose natural Arabic programming language based on a proprietary syntax.
- [LSE](#) - Langage Symbolique d'Enseignement, a French, pedagogical, programming language designed in the 1970s at the École Supérieure d'Électricité. A kind of BASIC, but with procedures, functions, and local variables, like in Pascal.
- [Mama](#) - An educational programming language and development environment, designed to help young students start programming by building 3D animations and games. It is currently available in English, Hebrew, Yiddish, and Chinese.
- [MS Word](#) and [MS Excel](#) - Their [macro](#) languages used to be localized in non-English languages
- [ML4](#) - A language for client/server database programming, with keywords in English or German
- [Nadesiko](#) - A Japanese programming language.
- [Phoenix](#) - A C-like high-level imperative procedural Arabic programming language

- [Rapira](#) - A Russian-based interpreted procedural programming language with strong dynamic type system
- Robik - A simple Russian-based programming language for teaching basics of programming to children
- [SAKO](#) - A language created in the 1950s and nicknamed the "Polish FORTRAN"
- [Swaram](#) (Tamil) - A simple, general-purpose and procedural language designed for programming in Tamil
- [Superlogo](#) - A Dutch creation for computer-aided instruction, based on Logo
- [TI-Calculator BASIC](#) - The 68000 version is localized. Unfortunately, various configuration strings are localized too, preventing direct binary compatibility.
- [var'aq](#) - A language based on the constructed Klingon language of *Star Trek*
- [W-Language](#) - A French programming language used in the WinDev CASE Tool.
- [Yiddishe Mama](#) - An educational programming language with a Yiddish version.
- [YMB \(Yazyk mashin buchgalterskih\)](#) - ЯМБ (язык машин бухгалтерских) - A Russian programming language for Iskra-554, Iskra-555, and Neva computers.
- [1C: Enterprise](#) - A Russian framework and language for business applications. English keywords can also be used.

و في هذه القائمة ذُكر: خمس لغات برمجةٍ عربية، و لغتان هندية، و ست لغاتٍ روسية، و ستٌ فرنسية، و لغتان كوريتان، و لغةٌ يابانية، و غيرهن الكثير!

و هذه القائمة لا تُمَثِّل كل اللغات غير الإنكليزية؛ فهي قطرةٌ من غيِّثٍ ليس إلا (كما سأوضح عند الحديث عن لغات البرمجة العربية في تالي مُناقِشَاتنا بِإِذْنِ اللَّهِ -تعالى-)، و في هذا الكفاية للرد على الإعتراض بشأن ابتداعنا لمسألة إنتاج لغة برمجةٍ غير إنكليزية، و أظن أنني قد نجحت في ذلك إلى الحد الذي لا يستطيع معه قارئ الرسالة إلا الرد بنفسه على مثل هذا الإعتراض مُستقبلاً.

و هذه صورٌ لأسطر برمجةٍ مكتوبةٍ بلغات برمجةٍ غير إنكليزيةٍ للتمثيل:

لغة Glagol الروسية:

```

ОТДЕЛ Привет+;
ИСПОЛЬЗУЕТ Вывод ИЗ "... \Отделы\Обмен\ ";
УКАЗ
  Вывод.Цепь ("Здравствуй, мир!")
КОН Привет.

```

لغة Lexico الاسبانية:

```

incluya "System.Windows.Forms"
clase ventana derivada_de "System.Windows.Forms.Form"
{
  publicos:
  mensajes:
    ventana copie "Este es el título de mi primera ventana" en ventana.text
}

```

لغة Rapira الروسية:

```

Проц Старт ()
  Вывод 'Hello World !!!'
Кон Проц

```

ترجمة لغة BASIC إلى الصينية:

10 卜=0	10 Y=0
20 入 水, 火	20 INPUT E, F
30 從 日 = 水 到 火	30 FOR A = E TO F
40 卜 = 卜+對數(日)	40 Y = Y + LOG (A)
50 下一 日	50 NEXT A
60 印 卜	60 PRINT Y

ترجمة لغة python إلى الصينية:

```
#!/usr/local/bin/cpython
回答 = 读入('你认为中文程式语言有存在价值吗 ? (有/没有)')

如 回答 == '有':
    写 '好吧, 让我们一起努力!'
不然 回答 == '没有':
    写 '好吧, 中文并没有作为程式语言的价值.'
否则:
    写 '请认真考虑后再回答.'
```

3- مازلنا لم نحل مشاكلنا و معوقاتنا التكنولوجية. فكل ما هنا هو تقليد أكثر منه إبداع. لست ضد التنوع ولكننا لن نفرض أنفسنا كمنافسين بالمزيد من إغلاق أنفسنا بحدود معلوماتية نضعها و نحن حتى لا ندرى بمعاييرها ولا مقاييسها.

نحن حتى لم نحل مشكلة الخطوط العربية على الإنترنت ولم نصل لحل لوضع معايير خاصة باللغة العربية قد يستخدمها المصنعون العالميون ليتحدثوا إلينا بلغتنا، لا منتجات لدينا لننافس العالم. مازلنا في المعضلة العربية ألا وهي الناقل و المنقول، و الكم المهور من العقول التي ترفض حتى أخذ العلم بلغته الاصلية.

لست ضد اللغات ولا منشئها ولكنى أرى أنها يجب ألا تتجاوز المفهوم التعليمي في هذه المرحلة من تاريخنا التقني فتقافتنا العربية في المجال المعلوماتي الآن لا تتجاوز مفهوم الترجمة النصية الغير مكتملة.

هذه اللغات يجب أن تكون وسيلة لحل مشكلتنا أو مشاكلنا الخاصة باللغة العربية قبل أن تستعرض عضلاتها في عمل برمجيات سطح مكتبية، مرة أخرى لست ضد اللغات و لكن ما الجديد؟ وما الهدف؟ ما العائد؟ و ما الحلول؟ لا نستطيع المنافسة في الصعود الى القمر بعربات تجرها الخيول. الأفضل أن نكمل من حيث انتهى الآخرون لا البدء من الصفر وجعل هذا المشروع هو كل غايتنا.

يحتوى هذا الاعتراض على ثلاثة نقاطٍ رئيسيةٍ هي:

- وجود مشاكل تقنية حالية تواجهنا (نحن العرب) لم نحلها بعد، مثل: مُشكلة الخطوط العربية على الإنترنت، و وضع معايير خاصة باللغة العربية قد يستخدمها المُصنِّعون العالميون ليتحدثوا إلينا بلغتنا.
- لغات البرمجة العربية يجب ألا تتجاوز المفهوم التعليمي في هذه المرحلة من تاريخنا التقني؛ فتقافتنا العربية في المجال المعلوماتي الآن لا تتجاوز مفهوم الترجمة النصية الغير مُكتملة.
- الأفضل أن نكمل من حيث انتهى الآخرون لا البدء من الصفر، وجعل هذا المشروع هو كل غايتنا.

أما الاعتراض الثالث فقد ردنا عليه من قبل فلا حاجة لنا إلى تكرار الرد مرة أخرى، أما الاعتراض الثاني فإجابته بسيطة حقاً؛ فالقول بأن الدور الذي يجب أن تلعبه لغات البرمجة العربية في المرحلة الحالية هو الدور التعليمي فقط ليس بالأمر المنطقي بتاتا لسببين اثنين:

- حينما نصمّم لغة برمجة عربية للمجال التعليمي المُمتد (أي الذي لا يقتصر علي تعليم الأطفال فقط، بل يشمل تعليم بعض الأمور الإحترافية أيضاً لمن هم أكبر سناً) فإن الجهد المبذول في تصميمها يُمكن أن يكون قريباً من الجهد المبذول في تصميم لغات البرمجة التي تتيح لنا المنافسة على المستوى العالمي؛ لأن الفارق بين الإثنين هو الهدف الذي نضعه نصب أعيننا عند تصميم اللغة الجديدة و بعض الوقت الزائد في حالة اللغة الأكثر إحترافية.
- و الفارق الأكبر بين اللغتين سوف يكون في المكتبة التي تُبنى لكل منهما؛ فبينما ستكون في حالة اللغة التعليمية مكتبة بسيطة تكفي فقط لتعلم الأساسيات البرمجية في النقاط المُراد تعلمها فإنها ستكون مكتبة ضخمة في حالة اللغة الإحترافية بحيث يُمكن استخدامها لبرمجة كافة أنواع التطبيقات (و ليست التعليمية الطابع منها فقط).

و بالتالي فإنه يُمكننا أن نستثمر هذا الأمر لصالحنا، فنضع من ضمن أهداف اللغة البرمجية التي نطالب بها أن تكون سهلة التعلم بحيث نستخدمها منذ بداية خروجها للنور في العملية التعليمية، قبل أن تُكتب لها المكتبة الضخمة التي ستتكلّف بجعلها قابلة للتنافس العالمي في مجال إنتاج التطبيقات البرمجية الإحترافية، و نكون بهذا قد اصطدنا أكثر من عصفورٍ بحجرٍ واحد؛ فمن الناحية التعليمية ستكون لدينا لغة البرمجة التي تصلح للتعليم من البداية حتي الإحتراف، و على الصعيد الزمني الأبعد ستكون لدينا لغة البرمجة الإحترافية التي تكفل لنا المنافسة كما نريد و ذلك دون بذل جهدٍ إضافي لإنتاج اللغة التعليمية، بل دون تعريض من تعلّم لغة البرمجة التعليمية لمشاكل تغيير اللغة

البرمجية من النطاق العربى إلى النطاق الإنكليزى؛ لأنه سيلجأ إلى لغات البرمجة الإنكليزية إن لم يجد لغة البرمجة العربية الإحترافية التى يكمل بها مسيرته فيما بعد.

• إن اكتفينا بالدور التعليمى فقط و لم نقم بإنتاج لغة البرمجة العربية الإحترافية فكيف سيتأقلم من دَرَس العلم البرمجى بالعربية و بَرَمَج بالعربية على التحصيل العلمى و البرمجة بالإنكليزية؟!!

و متى سيدرس اللغة الإنكليزية الحديثة التى سَتُحِثُّ له تحصيل العلم أصلاً؟! وهذا أمرٌ يجب التفكير فيه بروية؛ لأن الأمر لن يكون مع الأجيال القادمة كما هو مع الأجيال الحالية التى تعودت على التعلم بالإنكليزية و ستواجه بعض الصعوبة فى العودة للتعلم بالعربية، و ستكون هذه الصعوبة قليلة لا لشيء سوى لأنها لغتهم الأم و لأنهم درسوا بها العلوم حينما كانوا أصغر سناً (على الأقل هذا حال القسم الأكبر بينهم). فالأجيال القادمة التى ستتعلم بالعربية فقط لن يكون التعلم بالإنكليزية مجرد تغيير فى الإصطلاحات المُستعملة بل سيعنى الانتقال إلى لغةٍ مختلفةٍ تماماً لم يتعودوا عليها عند دراسة العلم بتاتا، و إلى طرقٍ مُختلفةٍ فى التعامل مع المُصطلحات و تكوينها و اختصارها!.

و بالعودة إلى الإعتراض الأول للرد عليه نجد أن المدخل لذلك هو: التساؤل عن السبب وراء عدم وجود المعايير التى تحكم ما يتعلق باللغة العربية، فالحق أن السبب فى ذلك هو عدم وجود مُؤَسَّساتٍ عربيةٍ للتوحيديات القياسية توضع تلك المعايير، أو على الأقل ليست هذه المُؤَسَّسات بالقوة التى تُسَوِّقُ بها تلك المعايير و تجعل الآخرين يسيرون عليها بإرادتهم الكاملة.

و ما دام الحال هكذا فمن سيضع تلك المعايير؟ و إذا وضعها فما الذى سيعطيها القوة و الإنتشار اللازمين لإقناع الآخرين باستخدامها عند الرغبة فى التحدث إلينا؟

و إجابة ذلك متعلقةٌ بالنهضة التى نتحدث عنها كل التعلق؛ فحينما ينشأ المُجتمَع التقنى الإسلامى الذى نتحدث عنه و الذى يمتلك أدواته الخاصة و يُنتج الجديد من العلم فإنه سيكون قادراً على وضع المعايير القياسية، و ستتكفل سمعته و كذا تواجهه الإنتاجى الفعلى بدعمها بمنتهى القوة، و سيكونان هما عاملا الإقناع للكل فى الداخل و الخارج، و بالتالى يتم حل تلك المُشكلات التى نتحدث عنها صاحب الإعتراض حلاً جذرياً و بطريقةٍ تُتيح لنا تطوير حلولٍ لأى مشاكل تُقابلنا بالإعتماد على أنفسنا و بالتعامل مع الآخر بمُنتهى النديّة.

4- من يريد أن يصنع لغة لا بد من أن يكون قد عمل مبرمجاً لفترة 5 سنين على الأقل، وعنده خبرة باللغات ومواطن ضعفها، و ليس مثل أناس مثلكم (مع احترامى) فيهم شباب لم يعمل، و كل ما يفعلونه

أنهم يسرون مع الموجة و "نعمل و نسوى". و لا أحد لديه أى **qualifications** مطلقاً، والمشكلة حتى في أن يطور أحدهم في لغة موجودة فعلياً؛ و هذا صعب أيضاً لأنك يجب أن تكون أساساً مبرمجاً عنده خبرة.

هذا الكلام صحيحٌ تماماً؛ فلكي تقول أنه بإمكانك أن تُصمّم لغة برمجةٍ أفضل من الموجود حالياً فيجب أن تكون قد عمِلتَ في مجال البرمجة لفترةٍ طويلةٍ (علي الأقل ليست بالقليلة) و بلغاتٍ مُختلفة الأنواع، و أن تكون قد درّستَ على الأقل العلوم التالية:

- هندسة البرمجيات (علي الأقل الأجزاء التي تحتاجها في الواقع العملي)،
- تصميم لغات البرمجة أو ما يُسمّى بنظرية لغات البرمجة **programming language theory**.
- تاريخ تطور لغات البرمجة (علي الأقل تطور اللغات الأشهر و الأكثر استخداماً).

و أن تكون لديك درايةٌ بسيطةٌ بفروعٍ أخرى مثل:

- أنظمة التشغيل و واجهات المُبرمج **API** المختلفة فيها،
- الشبكات و الأساس البرمجي لها،

فإن حَصَلَ المُبرمج هذه العلوم جيداً كان بإمكانه نظرياً نقد لغات البرمجة الموجودة بصورةٍ علميةٍ حقيقية، فإذا ما اكتشف عيوباً و مساوئاً في اللغات الحالية و جَمَعَ الصفات التي يراها الأفضل في لغةٍ جديدةٍ أصبح بإمكانه أن يقول أنه قد صمّم لغة برمجةٍ جديدة.

5- سؤال بسيط: انت يا عم المستخدم ضامن ان اللغة دي مثلا مش هيبقى فيها **bugs** تفتح ثغرات أمنية؟ لا طبعا لان التطوير مقصور على العرب واحنا بسم الله ما شاء الله بنتعلم أحسن تعليم وعباقرة فعلا عباقرة ههههههه و عدد المشاركين فى تطوير اللغة 2 متحمسين مشتغلوش مثلا فى **sun** وعارفين **java** مثلا عملت ايه ولا ايه مزاياها ولا الناس بتستخدم ال **C++** ليه.

يحتوى هذا الإعتراض على ثلاثة نقاطٍ رئيسيةٍ هي:

- إمكانية وجود عيوبٍ برمجةٍ تفتح الباب أمام الثغرات الأمنية فى بناء لغة البرمجة، و ذلك لعدم وجود الخبرة و التعليم المطلوبين للسيطرة على هذه العملية عند العرب الذين يبنون اللغة.
- عدد المشاركين فى تطوير اللغات سيكون قليلاً.
- لن يكون لدى المُشاركين فى التطوير أى خبرةٍ فى عيوب و مُميزات لغات البرمجة و هو الأمر الذى ينتج عنه ما ذُكر فى النقطة الأولى.

و مرةً أخرى سأردُّ على النقطة الثالثة أولاً، فلن نبدأ المشوار الطويل للنهضة إلا إذا ما حَصَلْنَا العلوم التي ذَكَرْتُ جزءاً منها من قبل، و لن أسمح لنفسي بأن أنشُر الدعوة إلى حركة التطوير العلمي الجديدة هذه قبل أن أضحى ذا خبرةٍ في مجالى تكفل لى أن أقول فأحسن القول حتي لا تُصبح الأمور عبثية، و فى هذا الكفاية.

أما النقطة الاولى فالرد على النقطة الثالثة يُعْتَبَر رَدّاً عليها أيضاً، و هناك رَدٌّ آخرٌ عندي: فالعملية التطويرية لن تكون حِكراً على العرب فقط بل ستكون مفتوحةً أمام المُسْلِمِينَ جميعاً، و فى شباب المُسْلِمِينَ غير العرب الكثيرون جداً ممن دَرَسُوا أفضل الدراسات و تعلَّمُوا بأفضل الطرق، و فى هؤلاء سَنَدٌ لنا لأنهم يتساوون معنا فى الأحقية فى العمل فى هذا الأمر، فدعوتنا ليست للعرب فقط، بل هى لأجل الإسلام أولاً و آخراً و العرب يستفيدون من ذلك.

أما النقطة الثالثة فأنا أعياها جيداً، و أعلمُ أن المُشَارِكِينَ سيكونون فى البداية قليلين للغاية و ربما يقتصر الأمر لسنواتٍ و سنواتٍ علىّ أنا فقط، و لكن هذا الأمر يَهُونُ إذا ما نَحَيْنَا العَامِلَ الزمنى قليلاً، و لا يتأتى هذا إلا إذا ما نذرتُ عمري كله لهذا الأمر و هو ما قررته بالفعل.

ثم إنه أثناء هذا المشوار الطويل و بعد عدة نجاحاتٍ فى مراحل العمل المُخْتَلِفَةِ بإذن الله عز و جل سيزيد اهتمام الآخريين بالأمر و سيرغبون فى الإضمام إلىّ فيه، و رُوَيْدًا رُوَيْدًا سوف تتكون قاعدةٌ من المؤمنين بالقضية يُسَاعِدُونَ على الوصول للهدف، و ما هى إلا بضع سنين و نكون قد وصلنا إلى جزءٍ كبير من أهدافنا بفضل الله تعالى و حينها سنجد الأكثرين و الأكثرين يرغبون فى المُشَارَكَةِ؛ لأن النجاح يُؤَدِّدُ النجاح و الفشل (إن استسلمنا له) يُؤَدِّدُ الفشل.

6- غلط إنك تجبر طفل فى صغره و فى طفولته على حاجة معينة؛ انت ايه عرفك أنه لما يكبر هيجب المجال ده، ده اسمه قتل الإبداع، و اوعى تقوللى أنه هستخدم البرمجة فى حاجات مفيدة فى مجاله. بدل كل ده ممكن تعمل لعبة للأطفال توصل لهم قيمة معينة بحاجة كويسة.. و بالمناسبة: المفروض نشجع الأطفال الصغيرين على تعلم الإنقليزية منذ الصغر و إن أمكن لغات أخرى لأن ده يفتح عقولهم و يخليهم يقدرُوا لما يكبرُوا يلاقوا مصادر معلومات تفيدهم و ممكن يفيدوا بيها المجتمع بعد كدة، إنما لو خليت كل حاجة عربى يبقى انت بترجع للخلف لأنه مفيش احتكاك بينك و بين الثقافات التانية و انت الأضعف.

هذا الإعتراض يُعْتَبَر من أسوأ الإعتراضات إن لم يكن أسوأها بالفعل؛ فأسوأ شيءٍ يُمكنك أن تُعَارِضَ به أمراً ما: هو الشيء الذى يَنْسِفُ حتى ما تُدافعُ عنه!، فالْمُعْتَرِضُ قال أن تعليم الأطفال البرمجة منذ الصغر يُعْتَبَر قتلًا للإبداع، ثم عاد و قال أن علينا أن نُعلِّمَ الأطفال منذ الصغر اللغة الإنقليزية و هذا تضادٌ و تضاربٌ فى كلامه، فلو سلّمنا جدلاً بصحة كلامه عن خطأ تعليم الأطفال منذ صغرهم علوماً

قد لا يحتاجونها حينما يكبرون؛ فإنه حينئذٍ يصير من العبث تعليمهم اللغة الإنجليزية لأنهم ربما لا يحتاجونها عند الكبر، فلو صار أحدهم نحوياً أو عالمٍ شَرَعَ فإنه لن يحتاج إلى تعلم اللغة الإنجليزية و سَيُعْتَبَرُ تعليمنا إياها له (بمقاييس المُعْتَرِضِ) قتلاً للإبداع.
و رُوَيْدًا رُوَيْدًا لن نُعَلِّمَ الأطفالَ أي شيءٍ لأنهم ربما لن يحتاجوه فيما بعد و سَيُعْتَبَرُ تعليمنا لهم أي شيءٍ قتلاً للإبداع !.

هذا لو سلّمنا بصحة كلامه، و لكن كلامه يُعْتَبَرُ من السخافة بمكان؛ فكل الناس تُدرك أهمية العلوم البرمجية في حياتنا المُعاصرة، و الإتجاه الحديث للتعليم في كل بلاد العالم يسير ناحية التوسع في تعليم البرمجة و علوم الحاسب للأطفال منذ الصغر، و كذا تطبيق علوم و منهجيات الحوسبة علي كافة مناحي الحياة فيما يُسَمَّى بـ (التفكير الحوسبي (computational thinking).
فالعصر الحالي هو عصر الحواسيب و الشبكات، و العصر القادم سيكون لهما أيضاً بكل تأكيد، و ما رأيتُه كطالب في الهندسة من دخول الحاسوب و برمجياته في كل المجالات يجعلني لا أصدّق أن أحداً يقول بقول صاحب الإعتراض.

و لكن الأمر يصير واضحاً لو فَكَّرْنَا أن صاحب الإعتراض لم يقل هذا الكلام عن صحيح اعتقادٍ فيه؛ بل قاله لأنه حاول مُجادلة أصحاب الدعوى لإنتاج لغة برمجةٍ عربيةٍ و لكنهم حاجُّوه فغلبوه (أو رأي ذلك يحدث لغيره، أو علي الأقل خاف من حدوثه معه)، فلم يَجِدْ بُدًّا من ترك الأمر الرئيس و ضرب الأهداف التي يقولون أن إنتاج لغة برمجةٍ عربيةٍ سيُحَقِّقها، و بالطبع فإن على رأس هذه الأهداف هدف تيسير العلم على أطفال العرب. و لكن لأن الآراء التي يقولها الفرد بدون اقتناعٍ فعلي بها تتضارب مع باقي أقواله؛ فقد تضاربت أقوال المُعْتَرِضِ مع بعضها البعض كما أوضحنا سلفاً.

أما قول المُعْتَرِضِ بأن جعل كل شيءٍ باللغة العربية يُعدّ تخلفاً لأننا سنفقد حينئذٍ الإحتكاك مع الآخر، و بالتالي نفقد القدرة على التعلّم لأننا الأضعف فهو مُغالطةٌ واضحة؛ فهذا القول كان سيكون صحيحاً لو أننا اكتفينا بما سنترجمه في المرحلة الأولى من المشروع ثم قلنا أننا قد اكتفينا بما أخذناه حتى الآن من الآخرين و لن نأخذ شيئاً آخر، و لكن ما ننويه غير ذلك تماماً؛ حيث سيظل الإحتكاك بيننا و بين الآخر قائماً على أساس من النديّة و التعاون المُتبادل لخدمة الأمة الإسلامية أولاً و التقدم البشري عامةً ثانياً.

أما تعليم الأطفال لغةً أخرى غير لغتهم الأم فأمراً لا نُنكر أهميته بالفعل، و يكفينا مبدأ " **مَنْ عَرَفَ لُغَةَ قَوْمٍ أَمِنَ شَرَّهُمْ**"³، و بالفعل فإن من أهم اللغات بالنسبة لنا (كمسلمين و كعرب) اللغتان:

3 هذا ليس بحديثٍ نبوي كما قد يظن البعض، بل هو قول دَارِجٍ يَنْسِبُهُ من لا يعلم إلي رسول الله. صلوات الله و سلامه عليه بدون تحقيقٍ أو تدقيقٍ! فِيرْجِي التنبه لمثل هذا لأن رسول الله. صلي الله.

الإنجليزية و العربية، و لو استطعتُ أن أُعلِّم كل أطفال المسلمين و شبابهم هاتين اللغتين لفعلت، و لكن ليس بيدي إلا التنبيه و الدعوة.

7- لا أعتقد أن فكرة لغة برمجة عربية هي فكرة جيدة، لأننا لن نستفيد شيئاً من هذه اللغة إلا الكتابة بالرموز العربية وهذا كما أعتقد لسنا بحاجة إليه، لذلك أنا أعتقد أنه يجب تطوير أفكار برمجية جديدة بغض النظر عن اللغة من أجل الرقي بالعرب.

هذا الاعتراض يأخذ مكانه بجانب أخيه السابق له على لائحة أسوأ الاعتراضات؛ فأولاً قال المُعْتَرِضُ أن فكرة إنتاج لغة برمجة عربية ليست جيدةً لأننا لن نُحَقِّقَ من ورائها إلا فائدة الكتابة بالحروف العربية، ثم قال ثانياً أنه حتى هذا الهدف لسنا بحاجة إليه، ثم ثالثاً أوضح فكرته بكيفية الرُقَى بالعرب عن طريق الأفكار الجيدة بغض النظر عن اللغة.

أما أولاً فقد أفضنا في شرح الفوائد التي ستعود علينا (نحن المسلمين) من وراء إنتاج لغة برمجة عربية، و لكن الأمر المهم أنها أكثر من مُجَرَّد كتابة الأكواد البرمجية باستخدام اللغة العربية، أما قوله بأننا لا نحتاج إلى الكتابة بالحروف العربية فقولٌ ينبئ عن جهل تام بالواقع و غزو الثقافات لبعضها البعض، و يكفينا هنا أيضاً قائمة الفوائد التي تنتج عن البرمجة باللغة العربية للرد على هذا الغلط.

أما القول الثالث فلا أنكر أن نصفه الأول (و هو حاجتنا إلى الأفكار البرمجية الجديدة) صحيح، و لكن العجيب أن المُعْتَرِضُ و من يحذو حذوه يتناسون أن بناء لغات البرمجة و أدواتها المُسَاعِدَة يُعَدُّ مجالاً من أهم المجالات البرمجية على الإطلاق، و لا يُنَافِسه إلا مجال بناء أنظمة التشغيل، و نهضة برمجية بدون هذين العِلْمَيْنِ هي نهضة كَسِيحَةٌ أَقْصَى طَمُوحٍ لها هو الشحاذة العلمية على أبواب المُنتَجِجِين الآخرين!

8- لسنا بحاجة إلى لغة برمجة عربية؛ فلن تجد من يعمل بها سوى الأطفال أو الأشخاص الذين يستصعبون البرمجة بلغات عليا ومعقدة مثل الـ C والـ java وغيرها.. لذلك فهؤلاء ليسوا بالأهمية كما المبرمجين العالميين، ولكن لو تم تطوير مبدأ برمجي جديد يغير من المفاهيم الحالية كما ظهرت في السابق البرمجة الشيئية التي قلبت الموازين، فإن ظهور شيء مماثل من القوة هو الأفضل لنا لأنه من المستحيل أن نرحل أي من اللغات عن عرشها.

عليه و سلم قال: (إِنَّ كَذِبًا عَلَيَّ لَيْسَ كَكَذِبِ عَلَيَّ أَحَدٍ. مَنْ كَذَبَ عَلَيَّ مُتَعَمِّدًا فَلْيَنْبَوُا مَقْعَدَهُ مِنَ النَّارِ) رواه البخاري.

رداً على هذا الإعتراض فإنني أطرح سؤالاً: إذا ما تم إنتاج لغة برمجة عربية قوية تتفوق على باقى لغات البرمجة فى كثير من النواحي فهل سيكون مُستخدِموها هم فقط من الأطفال و الأشخاص الذين يَسْتَصْعِبُونَ البرمجة بِلُغَاتٍ عُلْيَا و مُعَقَّدَةٍ مثل الـ C و الـ java ، أم أن المُبرمجين العرب سيستخدمونها بدورهم ؟

فإذا كان الرد بأن المُستخدِمين سيكونون من المُبرمجين المُحترفين فإن هذا هو الأمر الذى نطمح إليه، و سيكونون قد أُكْتُوا و اجبهم ناحية أمتهم و دينهم، أما لو كان المُستخدِمون الوحيدون من الأطفال و المبتدئين: فإن الأطفال يكفوننا عند تعليمهم و جعلهم مُبرمجين مُحترفين فى المُستقبل لتكملة النهضة العلمية البرمجية، و حينها سيكون الخاسر الوحيد هو أولئك المُتقاعسين الذين كان بأيديهم المُساعدة على إعلاء راية الأمة و لكنهم نكسوا رؤوسهم !.

ثم لما ذا يهتم المُعتزض بالمُبرمجين العالميين ؟!

و ما دخلهم بهذا الموضوع من الأصل ؟!

أسننا نحاول إنتاج لغة برمجة عربية لتحقيق الإكتفاء الذاتى من العلم و التعليم ؟ فما الداعى للنظر إلى جعل غير المُسلمين من المُبرمجين العالميين يُبرمجون بلغتنا ؟ فى حين ربما يُعتبر هو الأمر المُستحيل مع لغات البرمجة غير العربية؛ لأن المُبرمج المُحترف الذى يُتقن لغته كل الإتيقان لا يميل فى المُعتاد إلى تغيير تلك اللغة إلا فى حالات استثنائية و بعد تفكيرٍ طويل؛ و ذلك لأنه يكون قد اعتاد على لغته الإحترافية كل الاعتياد، و تكيّف مع مُميّزاتها و عيوبها كل التكيّف، فأصبحت بالنسبة له منزلاً مُريحاً جميلاً، و من أصعب الأشياء على الإنسان شراء منزلٍ آخرٍ يكلفه مَالاً و جهداً إضافيين بدون فائدة، فى حين أنه لديه منزله الفخم.

9- أعتقد أن المشكلة الأساسية والوحيدة أمام إنشاء لغة برمجة عربية هو عدم وجود أو قلة من يتحدثون

العربية.

إعتراضٌ آخرٌ يحتل مكانه فى قائمة أسوأ الإعتراضات؛ فتعداد العرب لا يقل عن ثلاثمائة مليون إنسانٍ أغلبهم من فِتي الشباب و الأطفال، و هذا العدد سيكون هو الهدف الرئيس لأى لغة برمجة عربية يتم إنتاجها، أما لغة **إبداع** فلأنها إسلامية الهوية أصلاً فهى مُوجّهةٌ لأكثر من مليارٍ و نصف المليار من البشر الذين يستخدمون العربية فى حياتهم أو على الأقل الجانب الدينى منها، و يستخدم كثيرٌ منهم الحروف العربية لكتابة لغاتهم مثل: الفارسية، و الأوردية، و الأمهرية، و غيرهن من اللغات.

فعلى أقل الاحتمالات تفاؤلاً فإن هناك الملايين يقفون وراء لغة البرمجة العربية نظرياً و يُكوّنون قاعدة هائلة من المرشّحين لا استخدامها، و بذلك تكون مُشكلة عدم وجود أو قلة عدد من يتحدثون العربية مُشكلة وهمية لا وجود لها بتاتاً.

10- لغات البرمجة القوية والتي لا تزال حية، لم تصنع لأن صانعها أراد صنع لغة برمجة فقط لتكون لغة برمجة، بل كل لغة كان لها سبب:

C: صنعت بشكل أساسي لبرمجة نظام UNIX على حد علمي، وانتشرت أيضاً لدعمها الكبير للبرمجة الهيكلية.

C++: صنعت لتسهيل C وإدخال البرمجة غرضية التوجه عليها.

Visual Basic: صنعت لتسهيل البرمجة على نظام ويندوز أو لأغراض تجارية.

وبهذا نرى أن لغات البرمجة المشهورة انتشرت لأنه كان لها أغراض معينة، أو لأنها أدخلت مفاهيم جديدة على البرمجة، مثل لغة Small talk التي أدخلت مفهوم البرمجة غرضية التوجه OOP. إذاً لتصنع لغة برمجة ناجحة، يجب أن يكون لهذه اللغة مميز ما، ولا تكون مجرد لغة برمجة أخرى.

بدايةً لو كانت اللغة البرمجية العربية التي نطمح إليها لا تُقدّم في عالم البرمجة شيئاً إلا إعطاءنا إمكانية البرمجة بمُصطلحات و حروفٍ عربية فإن هذا أمرٌ رائعٌ جداً، و لا ينتقص من قدرها شيئاً؛ و ذلك لأن هذا يُحقّق لنا الكثير من الأهداف التي وضعناها أمام أبصارنا حينما فكّرنا في أمر النهضة البرمجية الإسلامية، و لكن الرائع بما لا يُقاس هو أن تُوجد تلك اللغة البرمجية العربية التي تُقدّم الجديد إلى العلم البرمجي، و هذا هو ما نود حدوثه و ما سيكون نُصبَ أعيننا كما سنُوضّح فيما يلي.

و من الخطأ مقارنة لغات البرمجة الأخرى و خروجها إلى الوجود و ظروف ذلك الخروج بلغة البرمجة العربية و ظروف خروجها إلى الوجود؛ و ذلك لأن الغربيين يُمكنهم الإكتفاء بلغات البرمجة التي لديهم و التي أنتجوها بأنفسهم، و عدم إنتاج لغاتٍ أخرى إلا حين وجود الطفرة البرمجية التي تُغيّر من مسار علم البرمجة، و لكن الواقع يقول أنهم لم و لن يفعلوا هذا؛ فالواقع يقول أن لغاتٍ أخرى جديدة تخرج إلى الوجود و ذلك لوجود الداعي التجاري لها رغم أنها لا تُحقّق الطفرة البرمجية الضخمة الملموسة التي تُبرّر إنتاج لغةٍ كاملةٍ جديدة. و يكون الداعي الوحيد (أو الأكبر) لهذا هو التنافس التجاري بين عمالقة السوق البرمجي في العالم.

11- طيب عندي لك سؤال: الهنود تعدادهم كبير جداً و متقدمون علينا كثيراً في مجال تقنية المعلومات

و معتزون بثقافتهم؛ فلما ذا لا أرى نصف لغات البرمجة هندية؟

الألمان متعصبون و يعتزون بعرقهم ولغتهم؛ فلم لا أرى أكواماً للغات برمجة ألمانية؟

أنا لا أتعصب ضد اللغة الإنكليزية لأنها أصبحت لغة قياسية مثل المعايير القياسية، إذا استخدمت لغات البرمجة المشهورة أكون أستخدم نتاج عدد غير محدود من المستخدمين، طرحي للغة الإنكليزية يعد رافد قوي لها، لماذا نتوقع تحت مسمى اللغة العربية بدل من الإنطلاق باللغات المتوفرة؟ أنت تدعوا لمشروع رائد والدافع الوحيد هو التمني تحت مظلة أن العقلية العربية فذة ولا يضاهيها عقلية!.

أما القول بعدم وجود لغات برمجةٍ بغير اللغة الإنكليزية فقد ردنا عليه قبلاً.

أما القول الثانى فهو مُغالطةٌ واضحة؛ فلغات البرمجة منها ما هو مفتوح المصدر و يُشارك فى تطويره الآلاف حول العالم، ومنها ما هو مُغلق المصدر و لا يملك غير مُصمِّميه قرارات التصميم و مُستقبل اللغة، و تكون مُشاركة الآخرين فى تطوير مثل هذا النوع من اللغات عن طريق الإقتراحات التى يُقدِّمونها لمُصمِّمى تلك اللغات لكى يضموها إن استطاعوا إلى اللغة فتزيد من قوتها و كفاءتها، و بالنسبة للغة البرمجة العربية التى نودها فستكون مفتوحة المصدر يُشرف على تطويرها فريقٌ خاصٌ يستقبل اقتراحات المُتحرِّين و يُدير النقاشات معهم لتطوير اللغة طوال الوقت، كما يُتابع مسيرة تطوير باقى اللغات البرمجية ليستفيد منها، و بالتالى لن نخسر من العلم شيئاً بإذن الله تعالى بدون الحاجة لجعل لغة البرمجة نفسها باللغة الإنكليزية.

أما القول بأننا نتمنى ما نتمنى تحت مظلة القول بأن العقلية العربية لا تُضاهيها عقليةً أخرى فقول نرفضه قلباً و قالباً في مشروعنا هذا؛ فنحن لا نفاضل بين الناس إلا بتقوى الله عز و جل، و نقول قوله تعالى **(يَا أَيُّهَا النَّاسُ إِنَّا خَلَقْنَاكُمْ مِنْ ذَكَرٍ وَأُنْثَىٰ وَ جَعَلْنَاكُمْ شُعُوبًا وَقَبَائِلَ لِتَعَارَفُوا إِنَّ أَكْرَمَكُمْ عِنْدَ اللَّهِ أَتْقَاكُمْ إِنَّ اللَّهَ عَلِيمٌ خَبِيرٌ)** [الحُجرات: 13].
و أي شخصٍ يُريد المُساهمة معنا في هذا المشروع يُمكنه فعل هذا بدون أي عائقٍ كان.

12- أعتقد أنه من غير المعقول أن نبدأ من الصفر في الوقت الحالي والثورة الكبيرة في التقنية الحديثة. فكل لحظة يتم فيها اكتشاف جديد، بل يجب أن نبدأ من آخر ما وصل له العلم ونحاول أن نطور أنفسنا ومنافسة الغرب في لغتهم و بإذن الله نتفوق عليهم.

سَبَقَ و أن أوضحنا أننا لن نبدأ من الصفر على الإطلاق، و أننا سنبدأ بفهم العلم الحديث و آخر ما وصلت إليه العقول البشرية فى مجالنا، ثم نحاول تطويره أثناء إنتاجنا للغتنا، و بالتالى ستُعتبر إضافةً إلى العلم البشرى عامة.

13- صراحة ليس لدي الكثير، ولكن بقليل من المنطق ماذا نستفيد عندما نقوم كل يوم بعمل نفس الفكرة ولكن باللغة العربية؟. من الأفضل أن يكون هناك لغة برمجة عربية أي من صنع عربي ولكن يتم كتابة أكوادها بالإنجليزية لأن اللغة الانجليزية هي القياسية في كل شيء.

في البدء حينما نصنع لغة برمجة عربية تقوم بعمل نفس الأفكار الموجودة مسبقاً فإن هذا علي الأقل يُعد درساً عملياً عن كيفية تصميم لغات البرمجة وإنتاجها، وهو ما يحدث حول العالم في الجامعات العريقة عند تدريس مادة المترجمات، حيث يقوم الأستاذ بحث الطلبة على صنع مترجمات (أو أجزاء من مترجمات) للتعلم العملي وكسر جمود الدراسة النظرية، ولكن لأننا هنا لدينا من العباقرة الكثيرون فإنه حتى مثل هذا الأمر يُعد موضع تساؤلٍ ونقد، بل واستهجانٍ و سخرية!.

ثم إن المُعترض ناقض نفسه حينما طالب بعمل لغة برمجة عربية باللغة الإنجليزية؛ حيث أن قوله الأول يُطبّق على قوله الثاني، فلنا نحن أيضاً أن نسأل بقليل من المنطق: ماذا نستفيد عندما نقوم كل يوم بعمل نفس الفكرة و بنفس اللغة!؟

أما كَوْنُ اللغة الإنجليزية هي اللغة القياسية: فإننا نحن من جعلها لغة قياسية بتخاذلنا و خنوعنا، و حينما يشتد ساعد الأمة العلمي (ياذن الله تعالي) ستكون اللغة العربية لغة قياسية بدورها مُعتمدة على كمية هائلة من الإنجازات و الابتكارات التقنية.

14- الوصول الى الأفضل ليست فكرته اللغة، الوصول الى الأفضل فكرته صناعة التميز، و صناعة التميز لا يهم لغتها، اصنع التميز بلغة العالم تصل الى العالم، اصنع التميز بلغتك تصل الى نفسك ومن حولك فقط .

والأهم: أخي الكريم، اللغة العربية من أصعب اللغات، فمثلاً في الإنجليزية (print) هي نفسها للجمع والمؤنث والمذكر، أما العربية فهناك (اطبع واطبعوا و لنطبع) إلى آخره، بالنسبة لك أنت تعرفها، لكن ماذا بالنسبة لشخص يريد تعلم لغة البرمجة هذه؟.

لهذا أخي الكريم الفكرة المطروحة عن لغة برمجية بالعربية هي فكرة عقيمة، وليست ذات أهميه للمنافسة، أصلاً صناعة لغة برمجة ليست بتلك الأهمية؛ لماذا تريد أن تصنع لغة برمجة؟ ماذا تريد أن تضيف على اللغات المصنوعة أيضاً، ما الفكرة الجديدة المميزة التي تمتلكها ولم ينفذوها!؟

اصنع بلغات البرمجة تلك مشاريع مهمة، facebook لم تصنع لغة برمجة لكنها صنعت موقعاً على الانترنت (و صانعاها طالب في سنته الثالثة في جامعة هارفرد) أصبحت قيمة شركته 15 مليار دولار أمريكي، حتى أن مايكروسوفت اشترت بعضاً من أسهمها، ومالكها عمره لم يتجاوز الـ 23 سنة وأصبح رقمها السابعة على العالم وهي في تقدم ..لماذا لا نصنع هذا!؟

أَكْرَزَ مرةً أُخْرَى أَنَّهُ لَوْ كُنَّا نَطْمَحُ إِلَى الْوَصُولِ إِلَى أَنْ نَكُونَ الْأَفْضَلَ عِلْمِيًّا فَقَطْ لَكَانَ هَذَا الْقَوْلُ صَحِيحًا، وَ لَكِنَّا ذُووْ أَهْدَافٍ أُخْرَى لَا يُمَكِّنُ أَنْ تُحَقَّقَ بِهَذِهِ الطَّرِيقَةِ، وَ لَا يُمَكِّنُ تَحْقِيقَهَا إِلَّا بِالطَّرِيقِ الَّذِي اخْتَرْنَا سُلُوكَهُ وَ الْمَضَى فِيهِ حَتَّى آخِرِهِ.

وَ لَقَدْ ضَايَقْتَنِي بِحَقِّ فِي هَذَا الْإِعْتِرَاضِ مُحَاوَلَةُ الْمُعْتَرِضِ تَصْوِيرِ اللُّغَةِ الْعَرَبِيَّةِ عَلَى أَنَّهَا لُغَةٌ صَعْبَةٌ عِلْمِيًّا، ضَارِبًا لِذَلِكَ مِثَالًا سَخِيفًا هُوَ أَنَّ كَلِمَةَ **print** فِي الْإِنْغَلِيزِيَّةِ هِيَ نَفْسُهَا لِلْجَمْعِ وَ الْمُؤَثِّثِ وَ الْمَذَكَّرِ، أَمَّا الْعَرَبِيَّةُ فَهَنَّاكَ (اطبع و اطبعي و اطبعوا و لنطبع) إِلَى آخِرِهِ، وَ رَدًا عَلَى هَذَا أَقُولُ لِلْمُعْتَرِضِ أَنَّهُ يُمَكِّنُنَا بِمُنْتَهَى الْبَسَاطَةِ تَثْبِيتِ صِيغَةٍ مُعَيَّنَةٍ لِلْأَوْامِرِ مِثْلِ الصِّيغَةِ الَّتِي ثَبَّتْهَا لُغَةُ (ج) وَ هِيَ صِيغَةُ (أَفْعَلْ) مِثْلِ (اَكْتُبْ) وَ (ارْسُمْ)، أَيْ أَنَّ هَذَا أَمْرٌ بَسِيطٌ لِلْغَايَةِ.

وَ إِنْ كَانَ هَذَا قَدْ ضَايَقَنِي كَثِيرًا فَإِنَّ مَا جَعَلَنِي أَضْرِبُ أَخْمَاسًا فِي أَسْدَاسِ قَوْلِ الْمُعْتَرِضِ أَنْ يُنْتِجَ لُغَاتُ الْبَرْمَجَةِ لَيْسَ بِهَذِهِ الْأَهْمِيَّةِ: فَحَيْثُ أَنَّهُ لَا شَيْءٌ يُضَافُ إِلَى الْمَوْجُودِ فَعَلِيًّا فَإِنَّ صِنَاعَةَ لُغَةٍ بِرْمَجَةٍ أُخْرَى يُعْتَبَرُ أَمْرًا عَقِيمًا!، وَ الْحَقُّ أَنَّنِي لَنْ أَرُدَّ عَلَى هَذَا الْقَوْلِ وَ سَأَكْفَى بِالْقَوْلِ بِأَنَّهُ لَوْ أَدَارَ مِثْلُ هَذَا الْمُعْتَرِضِ شَرِكَةً بِرْمَجِيَّةً لَكَانَتْ نَهَايَتُهَا؛ مَا دَامَ يُفَكِّرُ بِطَرِيقَةٍ أَنَّهُ "لَيْسَ بِالْإِمْكَانِ أَفْضَلُ مِمَّا كَانَ"، عَلَيَّ الرَّغْمُ مِنْ أَنَّ هَذَا الْمَجَالُ لَا حَدَّ لَهُ إِلَّا حُدُودَ الْإِبْدَاعِ الْبَشَرِيِّ ذَاتِهِ.

وَ عِوَضًا عَنْ إِضَاعَةِ الْوَقْتِ فِي الْكَلَامِ الْفَارِغِ الْمُسَمَّى تَصْمِيمِ لُغَاتِ الْبَرْمَجَةِ يَقُولُ الْمُعْتَرِضُ أَنْ عَلَيْنَا أَنْ نَلْتَفِتَ إِلَى إِنتِاجِ أَفْكَارٍ أُخْرَى مِثْلِ **Facebook**، وَ الَّذِي أَصْبَحَتْ قِيَمَةُ شَرِكَتِهِ 15 مِلْيَارِ دُولَارٍ أَمْرِيكِي، حَتَّى أَنْ **Microsoft** اشْتَرَتْ بَعْضًا مِنْ أَسْهُمِهَا، وَ مَا لَكِهَا عَمْرُهُ لَمْ يَتَجَاوِزَ الـ 23 سَنَةً وَأَصْبَحَ رَقْمُهَا السَّابِعَةَ عَلَى الْعَالَمِ!، وَ إِنِّي لِأَتَسَاءَلُ عَنِ الْإِبْدَاعِ الْجَدِيدِ الَّذِي قَدَّمَهُ **Facebook** مَعَ كُلِّ احْتِرَامِي لِفِكْرَتِهِ الْجَمِيلَةِ؟!، فَلَيْسَ هُنَاكَ إِبْدَاعٌ خَارِقٌ وَرَاءَهُ إِلَّا أَنَّهُ لَبَّى رَغْبَةَ الْكَثِيرِينَ بِطَرِيقَةٍ أَفْضَلُ مِنَ الْمَوْجُودَةِ فَعَلًا. وَ لَكِنْ يَظَلُّ الْأَمْرُ أَنَّهُ لَمْ يُنْتِجْ عِلْمًا جَدِيدًا يَسِيرُ بِقَافِلَةِ الْعِلْمِ الْبَشَرِيِّ إِلَى الْأَمَامِ، وَ أَنَا أَقُولُ هَذَا رَغْمَ نَشَاطِي الشَّدِيدِ عَلَيَّ تِلْكَ الشَّبَكَةِ التَّوَالِفِيَّةِ، وَ إِدْرَاكِي لِأَهْمِيَّةِ وَجُودِهَا وَ الْفَارِقِ الَّذِي تَصْنَعُهُ فِي حَيَاةِ النَّاسِ بِالْسَّلْبِ أَوْ بِالْإِيجَابِ.

15- لِمَاذَا نَضِيعُ وَقْتًا عَلَى إِتْيَاقِ لُغَةٍ بِرْمَجَةٍ عَرَبِيَّةٍ؟! مَا الْفَائِدَةُ مِنْهَا؟!

إِذَا عَلِمْتَ الشَّبَابُ فِي الْجَامِعَاتِ الْعَرَبِيَّةِ لُغَةَ بِرْمَجَةٍ عَرَبِيَّةٍ وَتَوَجَّهُوا إِلَى سُوْقِ الْعَمَلِ، كَيْفَ سَيَبْرُمَجُونَ لِمَبْرَمَجِينَ أَجَانِبَ بَلْغَةَ عَرَبِيَّةٍ، لِمَاذَا عَلَى الْأَجْنَبِيِّ أَنْ يَتَعَلَّمَ الْعَرَبِيَّةَ وَهِيَ لَيْسَ إِلَّا الْعَاشِرَةُ عَلَى تَرْتِيبِ اللُّغَاتِ الْعَالَمِيَّةِ، إِضَافَةً إِلَى أَنَّهُ (أَخِي الْكَرِيمُ) مَا الْهَدَفُ أَصْلًا مِنْ لُغَةٍ بِرْمَجَةٍ جَدِيدَةٍ، مَاذَا تَرِيدُ مِنْ مِيزَاتٍ لَتَضْعَهَا وَهِيَ غَيْرُ مَوْجُودَةٍ.

أَمَّا التَّسَاوُلُ عَنِ الْفَائِدَةِ الْمَجْنِيَّةِ مِنْ وَرَاءِ إِتْيَاقِ لُغَةٍ بِرْمَجَةٍ عَرَبِيَّةٍ فَأَمْرٌ تَمَّ إِضَاحُهُ قَبْلًا.

أما التساؤل عن كيفية تأقلم الشباب المُبرمجين بلغة البرمجة العربية مع المُبرمجين الأجانب فتساؤل فيه مُغالطة واضحة؛ لأن الأجنبي حينما سيُنشئ فريقاً من المُبرمجين سيُراعى أن يكون مُختبروا مُنتجات الفريق و مُصلحوا عِلله bugs قادرون على استخدام نفس اللغة البرمجية التي يستخدمها الباقون، و يَصُدِّقُ هذا على الأجانب عامة، فلو أدارت microsoft مشروعاً علمياً يستخدم مُبرمجوه لغة الـpython مثلاً فيجب أن يكون مُختبرو التطبيقات المُنتجة قادرون على البرمجة باستخدام الـpython أيضاً (و لو علي نحو بسيط).

و بالمثل فإنه لو استأجرت شركة من الشركات البرمجية مُبرمجين عرباً يُرمجون بلغة برمجة عربية فسيكون عليها جعل مُختبري الفريق من اللذين يستطيعون أن يُرمجوا بتلك اللغة العربية. أما إذا اشترطوا استخدام لغة برمجة غير عربية (كأن يطلبوا مُبرمجين يحترفون الـRuby أو الـDelphi مثلاً) فلن تكون هناك مُشكلة؛ لأنني أظن أنه يجب على المُبرمج العربي أن يتعلم (و لو بنسبة صغيرة) الكثير من لغات البرمجة بجانب اللغة البرمجية العربية، و ما قال أحد أن إنتاج لغة برمجة عربية يمنع من احتراف لغات برمجة غير عربية، على الأقل لتفيد المُبرمج في مجالات العمل المُختلفة، و كذلك ليرى بنفسه الفروق التقنية بين لغات البرمجة و الأنواع المُختلفة منها. أى أننا سنُعَلِّمُ الشباب العربي لغة البرمجة العربية ليحترف البرمجة بها، ثم سيكون عليه (بعد الوصول للاحتراف) أن يُحصِّلَ بنفسه لغاتٍ أخرى غير عربية، و ستكون المهمة يسيرةً عليه لأنه يُمسِكُ فعلاً بالأساسيات من تعلمه للغة العربية، و سيكون عليه فقط أن يعرف الفروق و يضعها في حُسابانه و هو الأمر الذي جربته بنفسى و وجدته سهلاً.

أما بالنسبة للتساؤل عن السبب الذي سيدفع المُبرمج الأجنبي لتعلم اللغة البرمجية العربية فلا محل له من الإعراب في مشروعنا؛ و قد قلنا قبلاً أننا لا نهدف إلي جعل الأعاجم من غير المُسلمين يُبرمجون باللغة العربية، و إنما نستهدف المُسلمين و العرب.

و أخيراً يتساءل المُعتَرِضُ عن الجدوى من إنتاج لغة برمجة جديدة من الأصل، و عن الجديد الذي ستُقدِّمه و لم يتم إنتاجه من قبل على طريقة "ليس في الإمكان أبدع مما كان"، و قد ردنا على ذلك أيضاً في سابق كلامنا.

16- صراحة هذه مشكلتنا، أننا لا نمد أرجلنا قدر لحافنا و أضغاث أحلامنا كثيرة جداً جداً، و مشروعك أقول لك مستحييييييييييل و ح تشوف لو في واحد عربي ولا 50 عربي عملوا لغة زي الـ C++ و كلامي مش تنقيص للعرب ولكن كل الظروف عكس التيار.

لو كان مثل هذا القول صحيحاً لَمَا استطاع (لينوس تورفالدز) بدايةً من المرحلة التي كان فيها طالباً قيادة فريقٍ من مُختلف أنحاء العالم لِيُنْتِجُوا نواة نظام تشغيلٍ يُهْدِدُ الآن عرش نظام الويندوز windows الذي تُنتِجه microsoft.

و لَمَا استطاعت مجموعةٌ صغيرةٌ من المُبرمجين المُحترفين أسَّست مشروع الـ GNU إنتاج أدواتٍ برمجيةٍ تُعتبر الآن الأدوات المُفضَّلة لكثيرٍ من المُبرمجين الفطاحل و العماد الأساسي لأضخم المشاريع مفتوحة المصدر.

بل و لما استطاع الطالب (ماتثياس إترش) تأسيس مشروع واجهة الـ KDE حينما لم يُعجبه حال الواجهات المرئية لأنظمة التشغيل شبيهة اليُونِكس unix-like، و من ثم صارت من أنجح و أقوى الواجهات البرمجية علي مُستوي كافة أنظمة التشغيل الموجودة.

و الحق أن مُشكلة المُعترض ليست في سوء نظرتَه إلى العرب، بل إن مُشكيلته الكُبرى في تعميمه لهذه النظرة على جميع المُبرمجين العرب بلا استثناء؛ فمن الحق أن كثيراً من المُبرمجين العرب و خريجي الجامعات العربية لا يفقهون في العلوم البرمجية إلا بمقدار ما يفهم كاتب هذه الأسطر في الكيمياء (و أعترف أنني لا أفقه فيها شيئاً، سوى أنك لو وَضَعْتَ شيئاً ما مع شيءٍ ما ثابٍ لانتجَ هذا لنا شيئاً ما ثالثاً)، و لكن من الحق أيضاً أن البعض منهم يمتلك قدراتٍ علميةٍ رفيعةٍ تؤهله للمنافسة العالمية.

17- سيظل الأساس اللغوي الإنكليزية لأن لغة الحاسب هي الأسمبلي، أي أنك تكتب المترجم **Complier** الذي يترجم أوامر لغة البرمجة سواءً كانت بلغة عربية أو روسية أو غير ذلك فإنه بالنهاية يترجمها إلى لغة الأسمبلي والتي بالأساس هي لغة الحاسوب، والتي هي بدورها تقوم بمحاكاة الحاسوب ليعمل الأوامر وهي لغة مبنية على الإنكليزية.

ثانياً ليس هنالك منطوق أن تقوم بذلك؛ لأنك ستأخذ الوقت الكثير لبناء لغة أسمبلي ومن ثم لغة برمجة عربية، في حين أن الغرب يتقدم علينا بالتكنولوجيا، يمكننا أن نستخدم ما وصلوا إليه ونحن نبني شيئاً جديد فهاكذا انتقلت حضارتنا إليهم، أخذوا علومنا وقاموا بالبناء عليها. فمثلاً أخذوا علوم ابن سينا وأكملوا عليها واكتشفوا اكتشافات جديدة بناءً على ما قام به ابن سينا في الطب، وغيره الكثير من علماء المسلمين.

هذا الكلام لا يحوي من العلم إلا مقدار ما يحويه السراب من الماء؛ فالمُترجم **Complier** لا يُترجم الأوامر البرمجية العليا في نهاية الأمر للغة التجميع **assembly language**. بل يُترجمها إلى لغة التجميع الخاصة بالمعالج **processor** المقصود أولاً، ثم إلى أصفارٍ و أحادي في آخر الأمر.

و لغة التجميع ليست هي لغة الحاسوب الأصلية، بل هي كغيرها من لغات البرمجة الأخرى مُجَرَّد واجهَةٌ لما يفهمه الحاسوب و لذلك فإنها تحتاج بدورها إلى مُترجمٍ من نوعٍ خاصٍ يُسمَّى مُجمِّع **assembler** يقوم بترجمتها إلى الأصفار و الأحاد، و الإختلاف بينها و بين اللغات عالية المُستوى أن لغات التجميع تكون أوامرها مُناظرةً تقريباً لأوامر الآلة.

و مثالٌ وهميٌّ على ذلك قولنا أن الأمر:

00101010111010100100101010011001

يُكتَب:

ADD R1, R2, R3

في لغة التجميع الإنكليزية، و يُمكن أن نصنع برنامجاً يُنتج نفس الأمر الآلي حينما يقرأ الجملة العربية التالية:

أجمع م1، م2، م3

فهل سيصرخ الحاسوب قائلاً أنه لا يفهم سوى لغة التجميع الإنكليزية، أم أنه لا يهتم إلا بالأصفار و الأحاد التي يُمكن إنتاجها من جُملةٍ عربيةٍ أو إسبانيةٍ أو هيروغليفيةٍ؟!.

و بالمناسبة: فإن هذه الأرقام التي يستخدمونها هي مُجَرَّد واجهَةٌ أخرى لما يحدث حقيقةً في عتاد الحاسوب؛ حيث أن الصفر ما هو إلا رمزٌ لـ(لا شُحنة)، و الواحد رمزٌ لـ(شحنة)، فحتى الأصفار و الأحاد مُجَرَّد واجهَةٌ لما يفهمه العتاد الصلب المُكوّن لدارات الحاسوب !
فببساطةٍ حينما نحتاج إلى لغة تجميع نُخاطب بها الحاسوب عند بناء المُترجم الكامل: يُمكننا بمُنتهى البساطة إنتاج لغة تجميعٍ عربيةٍ. و سهولة الأمر في أن لغة التجميع تكاد لا تكون إلا إنعكاساً جلياً للأوامر الأولية المبنية داخل المُعالج.

18- لماذا نحتاج للغة برمجة تستخدم كلمات و عبارات لغة عربية و لغات مثل **C** و **C++** موجودة؟! هذا الأمر سيزيد من الفجوة الرقمية بيننا و بين الغرب، في الحقيقة فلنفترض بأننا نستخدم مترجم لغة **C** يتعامل باللغة العربية، و أردنا برمجة تطبيق به على نظام ويندوز فهذا الأمر سيتطلب تحويل جميع ملفات الترويسة **header files** و المكتبات إلى مكتبات تستخدم شيفرة مترجم لغة **C** العربية، عندها سيتطلب هذا الأمر إعادة كتابتها جميعها الخ... هذا بالإضافة للمكتبات المفتوحة المصدر و غيرها مثل **OGRE , OpenGL** و.. و.. إلخ.

و الشيء الأكثر أهمية هو أن اللغة العربية يصعب إيجاد اختصارات بها على عكس اللغة الإنجليزية، هذا و نحن لم نذكر بأن مثل هذه اللغات لن تجد من يبرمج بها لأن اللغة العربية يصعب تعلمها و هذا قد يشكل عقبة كبيرة.

القول بأن الفجوة الرقمية بيننا و بين الغرب ستزداد حينما نستخدم لغات برمجة عربية قول غير صحيح؛ فالفجوة الرقمية تزداد و ستزداد بسبب جهلنا بالتقنيات البرمجية التي يُنتجها الغرب و يُطوِّرها كل يوم، فاللغة وعاءٌ للتقنية المطلوبة و ليست تلك التقنية نفسها، فلو أنتجنا لغات برمجة عربية تستخدم التقنيات و الأساليب البرمجية الحديثة فلن تتأخر عن الغرب في شيء، و سيكون الأمر كأن نستخدم سِماداً من صُنِع محلي له نفس فائدة السِماد المصنوع عالمياً، فالمهم هنا هو مكونات السِماد و الطريقة و التقنيات التي استخدمت في إنتاجه، و ليس مكان صنعه أو لغة الكلام المكتوب على الكيس الحاوي له.

و لو أن لغتنا البرمجية العربية كانت تستخدم الخوارزميات algorithms الأفضل و المنهجيات البرمجية programming paradigms الأحدث و الأقوى لساوت اللغات الأخرى في قوتها، و لو استطعنا تطوير خوارزميات أفضل و منهجيات أقوى و طبعتها على لغتنا البرمجية لصارت أقوى من باقي اللغات.

و بخصوص المشكلة التي يتوهم المُعترض إمكانية حدوثها عند استخدام مُترجم لغة C يستخدم اللغة العربية فهي سرابٌ بَقِيعة؛ لأننا نستطيع إعطاء الماسح scanner و السَّابِر parser القدرة على التعامل مع الكلام الإنجليزي و العربي بنفس الطريقة و ترجمة التعبيرات المُتناظرة فيهما إلى نفس المُخرجات، و بالتالي يكون المُترجم قادراً على ربط الأصفار و الأحاد الناتجة من ترجمة الكلام العربي و الأصفار و الأحاد الناتجة من الكلام الإنجليزي (أو العبري أو الديوپيقى) ليُنتج لنا أصفاراً و أحاداً لا يعلم إلا الله عز وجل مصدر كل منها.

ثم من قال بأن هذا الهزل سيكون هو الهدف الذي نرجوه: إن هدفنا هو إنتاج لغة برمجة عربية لها مكتبتها الخاصة التي تتفوق على الجميع، و لو احتجنا إلى مكاتب مكتوبة بغير اللغة البرمجية العربية فمن أسهل ما يكون صُنِع ما يُشبه الجسر البرمجي لتلك المكتبة يُمكننا من استخدامها، كما يحدث حينما يُريد مُبرمجو مكتبة مُعيَّنة مثل الـ OpenCV المكتوبة بلغة الـ C إتاحة الفرصة لمُبرمجي لغة مثل الـ python لاستخدام مكتبتهم، و الفارق الوحيد هو أن جسرنا سيأخذ بعين الإعتبار تَغْيِير اللغة العربية إلى اللغة الإنجليزية فقط.

و هذا الأمر ليس كما يتوهم المُعترض أمراً جَللاً بل هو أمرٌ قريب المَنال، و مثالٌ على ذلك لغة السوبرنوا (سيلي ذكُرها فيما بعد) و التي يُمكن فيها مزج اللغتين العربية و الإنجليزية في الكود البرمجي بُمُنْتَهَى الحرية.

19- ليس هذا الذكاء، الذكاء أن تطرح لغة برمجة للعالم، هل تعلم من هم القائمين على لغة الـ php ؟

شركة برمجة اسرائيلية ومقرها " اسرائيل " ولكنها قامت على لغة البرمجة باللغة الإنجليزية. تخيل فقط أن لغات البرمجة كلُّ بلغته؛ ما إذا ستدرس في الجامعة، وإذا درّست كل جامعة في كل بلد لغة البرمجة الخاصة بلغة تلك البلد في مايكرو سوفت مثلاً عندما يريدون توظيفك وأنت متخرج تعرف تبرمج بالعربية فقط، لما إذا سيوظفونك أصلاً، أو مثلاً عندما تقوم بعمل كود وتطرحه كود مفتوح من سيعدل على كودك وهو لا يعرف لغتك، تخيل أن مثلاً أكواد المصادر المفتوحة كانت بالألمانية أو الصينية هل كنت لتستفيد منها، تخيل لو أن شخص جاء اليك وأنت أكبر خبير في البرمجة وقال لك هذا الكود به مشكلة ووجدته باليابانية ما ذا ستفعل؟!!

اللغة ليست عائق، تعلم البرمجة تعلم لغة بحد ذاتها وعلى العالم كله أن يتعلم لغة واحدة ليتفاهم، الذكاء أن تصنع لغة برمجة في حال كنت في ذاك القدر من الخبرة بالانجليزية تنافس بها العالم

بل الذكاء هو أن تُحدّد الأهداف التي ترغب في تحقيقها حق التحديد، ثم تضع لتلك الأهداف طُرُقاً و وسائل يُمكنك من خلالها أن تصل إليها، و أن تضع تلك الطُرُق و الوسائل في جدولٍ زمني جيدٍ يكفل لك ترتيب جهديك و عدم إضاعة طاقتك و يكفيك شر البلبلة و التخبط، و أظن أننا قد فعلنا ذلك حينما قلنا أن إنتاج لغة برمجةٍ عربيةٍ يُعد هو حجر الأساس في أى نهضةٍ برمجيةٍ نرغب فيها، و لا يُمكننا أن نُكمل باقى مشوار النهضة إلا إذا ما حقّقنا هذا الهدف الأول.

و ليس الذكاء هو طرح لغة برمجةٍ للعالم؛ فما الهدف من وراء ذلك؟، و ما الذى سنستفيدة غير رفع إسهامنا العلمى فى المجال البرمجى فقط مُصْخِين بلغتنا و باقى الأهداف التى ببعض الجهد الإضافى سنحصل عليها عند إنتاج لغة برمجةٍ عربيةٍ؟

و المثل الذى ضربه المُعترض بلغة الـ php لا يعنينى فى شئ؛ فلو لم تكن هناك لغات برمجةٍ بلغةٍ غير الإنجليزية على الإطلاق لما شكّل هذا فارقاً بالنسبة لنا، لأنه من الواجب الدينى علينا أن نُقوِّى لغتنا العربية و أن نضع الأمة الإسلامية فى مقدمة الأمم حتى لو خنعت باقى الأمم للغرب و أفرّت له بالريادة فى كل مجال.

و لكن الواقع يرُكِّد هذا الإِدعاء و يقول أن هناك لغات برمجةٍ غير إنجليزية و غير عربيةٍ مثلما أفضنا القول من قبل، و من الأقدار أن من ضمنها لغة برمجةٍ بالعبرية و هى لغة HPL – Hebrew Programming Language، فما بالننا نرى الآخرين يقومون بدافع من الحس القومى أو الحماسة لدينٍ غير صحيحٍ بإنتاج لغات برمجةٍ بلغاتهم و لا نقوم نحن أصحاب دين الله تعالى الخاتم بالدفاع عن لغة ذلك الدين و عن ريادة أمتنا حتى النفس الأخير؟!!

و جزئية سوق العمل قد أجبنا عنها سابقاً، أما جزئية المصادر المفتوحة و العائق اللغوى الذى سينشأ حينما نجعل أكوادنا العربية مفتوحة المصدر فأقول: أنه حقيقةً سوف يمنع حاجز اللغة

الغربيين (و خاصةً غير المسلمين بطبيعة الحال) من المُشاركة معنا في أكواد مصادرنا المفتوحة. ولكن أليس في مليار و نصف مليار مُسلمٍ من روسيا و تركيا و الصين و اليابان و فرنسا و جنوب أفريقيا و البرازيل و الهند و أمريكا و مصر و باقى أنحاء الأرض الكفاية؟! ألا يقدر كل هؤلاء باختلاف ألوانهم و أعراقهم و تعليمهم و بيئاتهم، و اشتراكهم فقط فى الإخلاص لدين الله تعالى و الأخوة الرابطة بينهم على جعلنا نستغنى عن أى جهدٍ آخرٍ من الخارج إن ضن به ذلك الخارج علينا؟

فإن كانوا يستطيعون (و هم يستطيعون و الحمد لله - تعالى) فلا مُشكلة هناك، أما إن لم يكونوا قادرين على ذلك فإن هذه أمةٌ لا تستحق الحياة و من الشرف لها أن تُدفن تحت الأرض ما دامت لا تقوى بكل هذا العدد من أبنائها بكل الاختلافات و التمايزات التى بينهم على ستر عورتها العلمية بين أمم الأرض!

أما قول المُعترض أن على العالم أن يتعلم لغةً واحدةً ليتفاهم فقد يقصدُ به أن عليهم تعلم لغةٍ برمجيةٍ واحدة، و هذا أمرٌ يكذبُه الواقع قبل النظر العقلى؛ فما يحدث الآن أن هناك الآلاف من لغات البرمجة باختلاف أنواعها تم إنتاجها فى أزمانٍ مُختلفةٍ لتلبي احتياجاتٍ مُختلفةٍ فى أماكن مُختلفة، بل و هناك لكل زمنٍ واحدٍ و كل حاجةٍ واحدةٍ و كل مكانٍ واحدٍ أكثر من لغةٍ تلبي طلباته و لا داعى لكثرتها إلا التنافس التجارى بين الشركات، أو الغيرة العلمية بين البشر و مُحاوله كل منهم الإدلاء بدلوه فى هذا البحر الخضم لعله يضيف إلى العلم البشرى و لو أقل القليل.

أما النظر العقلى فيقول بأن التنافس العلمى من الأمور المحموده؛ لأن الفائز الوحيد فى النهاية حق الفوز هو مقدار العلم الذى و صل إليه البشر، و قدرتهم على تسهيل حياتهم و إنجاز ما يرغبون فيه بأقل جهدٍ مُمكنٍ بالإعتماد على ما اكتشفته و ما أنتجته قرائحهم من علم.

أو قد يقصد المُعترض باللغة الواحدة اللغة الإنكليزية ذاتها كلسانٍ مُوحدٍ للنشر العلمى، و هذا من الأمور التى فيها مُغالطةٌ واضحةٌ و يكذبُه الواقع أيضاً. فليس لزاماً على العالم كله أن يتكلم لغةً واحدةً لكي يتفاهم تقنياً؛ فللترجمة دورها الأساسى إن لم يحدث توحيدٌ للغة البحث العلمى، و لو افترضنا أن اللغة العربية صارت لغةً عالميةً فى مجال البرمجة و أصبح من المعتاد أن تُطرح بها أبحاثٌ علميةٌ برمجيةٌ علي مستوى عالٍ من التميز فإن المهتمين الغربيين بالتقنية (مثل الشركات الضخمة أو الجامعات الكبرى) سوف يُضطرُّون فى نهاية الأمر إلى الإهتمام بالترجمة من العربية إلى الإنكليزية.

فإن لم يهتموا بالترجمة: فسيقع الضرر عليهم و ليس علينا؛ ففي النهاية سوف نحصلُ علي أبحاثهم لأننا نجيد لغتهم و سيخسرون هم أبحاثنا لعدم اهتمامهم بما هو خارج دائرة لغتهم و هذا خطأٌ كبير، و قد قلنا قبلاً أننا سنهتم بكل الأبحاث العلمية المُتفرِّدة التى نستطيع الوصول إليها و نترجمها حتى و إن كانت بلغاتٍ أخرى غير الإنكليزية.

أما المُنَافَسَةُ فهي هدفٌ من أهدافنا حقاً و لكنها تَلِي أهدافاً أخرى من حيث الأهمية كما أوضحنا آنفاً، ثم إن التنافس يكون بالمنتج الأفضل تحقيقاً لأهدافه، و لغة البرمجة العربية تُحَقِّق لنا أهدافاً أكثر بكثيرٍ من لغات البرمجة غير العربية، و باقي التطبيقات البرمجية التي سنُناقِشُ بها هي الأكثر أهميةً بالنسبة لغير المسلمين و اللذين ليسوا بعَرَبٍ في المُنَافَسَةِ، و هذه لا يهتم فيها أ صَنَعَتَهَا بلُغَةً برمجةً عربيةً أم بلُغَةً برمجةً إنجليزيةً؛ فالمهم أن تكون تلك اللغة قادرةً على إنتاج برامج كفاءةً لها سرعة تنفيذٍ عاليةٍ و مقروئيةً **readability** و مكتوبيةً **writability** عاليتين، و كذا إعتما ديةً كبيرةً.

20- تبقى اللغة الإنجليزية أفضل لأن حروفها منفصلة و ليس لها أكثر من شكل كما هو الحال في اللغة العربية: أحرف متصلة و أخرى منفصلة، و شكل في أول الكلمة و شكل في منتصف الكلمة و شكل آخر في آخرها، لذلك الترجمة للبرنامج المكتوب باللغة الإنجليزية سيكون أسهل و أكثر سرعةً.

هذا كلامٌ لا يقوله إلا شخصٌ:

1) مهزومٌ معنويًا إلى أقصى الحدود المُمَكِّنة فأصبح لا يرى فيما لديه أي شيء حسن، و أصبح يرى ما عند عدوه الذي هزمه أفضل الأشياء و أحسنها، و تصدق فيه مقولة ابن خلدون **"المَغْلُوبُ مُوَلِّعٌ بِتَقْلِيدِ الْغَالِبِ"**⁴ أيما صدق؛ ففي نظر هذا التَّعَسِ فإن اللغة الإنجليزية أفضل في الإِستخدام في إنتاج لغات البرمجة من اللغة العربية لأن حروفها مُنفصلةٌ و ليست مُتصلةٌ!، و لها شكلٌ واحدٌ و ليست مُتعددة الأشكال ككثيرٍ من حروف اللغة العربية! و العجب أن هذا هو السبب الوحيد الذي أوردته و كأنه أورد حُجَّةً من حُجَجِ ابن حزم ضد مُناظريه أفحمهم بها فلم يستطيعوا لها رداً، و الحق أنها جعجةٌ بدون طحنٍ و مُتهافتةٌ قلباً و قالبا.

و يتبين من القول كذلك أن المعترض:

2) لا حَظَّ له من العلوم البرمجية الخاصة بإنتاج المُترجمات أو حتى تصميمها؛ فقد وَقَعَ في تقرير أخطاءٍ لا يقع فيها مُبرمجٌ يفهم تلك الأمور جيداً بحالٍ من الأحوال، فلا دَخَلَ لاتصال الحروف ببعضها البعض أو انفصالها عن بعضها البعض، و تغير أشكالها أو ثباتها أئُ دخل في سهولة و سرعة ترجمة البرامج المكتوبة بلُغَةً البرمجة! . و الأمور التي تحكم هذه العملية تختلف تماماً عن تلك المُورَدَة، و منها:

4 في (المُقَدِّمَة) لابن خلدون.

- طريقة إنتاج مُكوّنات المُترجم المُختلفة من سَابِرِ parser و مُحَقِّقٍ منطقيّ semantic analyzer و غيرهما من باقى المُكوّنات، فهل تم إنتاجها باستخدام أدوات برمجيّة مُتخصّصة تأخذ قواعد لغة البرمجة الجديدة المكتوبة بصيغ و صفيّة قياسية مثل الـ EBNF و تُخرج برنامجاً يؤدى دور السّابِرِ أو المُحَقِّقِ المنطقيّ للغة، و ذلك مثل برامج الـ yacc و الـ ANTLR الشهيرة؟. أم تم إنتاجها بشكلٍ مباشرٍ يدوياً للحصول على أفضل أداءٍ مُمكن باستخدام خوارزماتٍ خاصة بالمُبرمج نفسه؟.
- خوارزمُ السّيرِ parsing algorithm المُستخدَم في مُترجم اللغة.
- خوارزمات التحقيق المنطقي semantic analyzing algorithms المُستخدمة.
- قواعد اللغة البرمجية و مدى سهولة بنائها و التحقق منها برمجياً، و عدم وجود تضاربٍ منطقي فيها.
- شكل الصّيغة التمثيلية الوسيطة intermediate representation فى المُترجم.
- خوارزمات إنتاج الكود الهدف object code.
- خوارزمات تحسين شجرة السّير للبرنامج parsing tree optimization.
- خوارزمات تحسين الكود الوسيط.

و هذه الأمور لا تُؤثّر فقط على سرعة المُترجم، و إنما تُؤثّر كذلك على كل ما يتعلق به من أمورٍ مثل الكفاءة و الإعتمادية و النشاط المُستمر و غيرها من الصفات الأخرى.

و لكن دعونا نفترض افتراضاً جديلاً أن اللغة الإنكليزية أفضل فى بناء لغات البرمجة من اللغة العربية: فهل سيدفعنا هذا إلى التخلّى عن أمل إنتاج لغة برمجةٍ عربية اللسان؟ بالطبع لا؛ لأن الأهداف التى نسعى إليها ستجعلنا نهمل هذا الأمر تماماً لنُحققها على أرض الواقع، فمثل تلك الاعتراضات حتى و إن صَحّت لن تُقدّم أو تُؤخّر فى أمرنا شيئاً و سيظل الحال كما هو، و لكننا ردنا عليها فقط لنزيح الحيرة عن المُحتارين و لتطمئن قلوب الناس بما نُؤمن به ليس إلا.

21- لن نجد أب متمرّس سوف يجعل ولده يبدأ بلغة عربية بسيطة، بل سيجعله يبدأ من pascal مثلاً.

أو حتى php، أو أي لغة أخرى لعدة أسباب:

- اللغة لن يوجد لها مطورين كثر،
- لن يوجد لها دعم كافي،
- لن يجد لها تطوير و إصلاح للأخطاء،

- لن تجد الكتب.
- لن تجد الأمثلة.
- لن تكون اللغة متقدمة بالشكل الكافي.

يعني في حال تعلم الطفل الأساسيات سوف يترك هذه اللغة، لماذا؟ لأنها لن تكون مُنتجة **Productive**. ناهيك عن صعوبة البرمجة من غير وجود واجهات و خاصة لطفل.

المُعْتَرِضُ يفترض في كلامه أن أي لغة برمجةٍ عربيةٍ لا بد و أنها ستكون لها الصفات التالية:

- من الطرُز البرمجية القديمة و ليست على القدر الكافي من التطور،
- عدد مُطَوِّرِهَا قليل،
- ليس لها دعمٌ كبير،
- سيتوقف تطورها و إصلاح أخطائها بعد فترة.
- ستعاني من قِلَّةِ الكتب التعليمية و الأمثلة الواقعية التي يتم بها تعليمها للطلاب و الصغار.

و بناءً على كل ما فات يَسْتَنْتِجُ أن هذه اللغة ليس لها مُسْتَقْبَلٌ في تعليم البرمجة للأطفال، و بالتالي فإن الآباء المُتَمَرِّسين لن يبدأوا تعليم أطفالهم البرمجة بلُغَةً مثلها!، و الحقيقة أنه لو أصبح الوضع كما افترض المُعْتَرِضُ: فبالفعل لن يُخَاطِرَ أي أب بهذا، حتى أنا نفسي إن رزقني **الله** تعالى بالولد و أردتُ تعليمه البرمجة فلن أعلمه بلُغَةً مثل تلك اللغة التي تحدّث المُعْتَرِضُ عن مساوئها. لكنني أتساءل عن السبب الذي سيجعل لغتنا التي نسعى إليها تعاني من كل تلك العيوب، فأما العيوب من الثاني و حتى الخامس فهي ستنتج عن تخاذل المُجْتَمَعِ البرمجي الإسلامي عن دعم اللغة الجديدة و الوقوف وراءها كلُّ حسب قُدْرَتِهِ و في مجال تخصصه، و بالتالي يكون لي أن أطلب من المُعْتَرِضِ التخلّي عن موقفه الراض لأنّه سيكون من أكبر أسباب ذلك التخاذل الذي سيؤدّي إلى وقوع ما يكره من الأمور!

و أما السبب الأول فلا أعلم ما الذي يدفع المُعْتَرِضَ لافتراض أن أي لغة برمجةٍ عربيةٍ ستكون قديمةً و غير مُتقدِّمة، فالمُفْتَرِضُ أننا إذا ما حصلنا العلم البرمجي و تشرّبناه جيداً كانت لنا القدرة على استخلاص جوانب القوة و الضعف في لغات البرمجة العالمية الأقوى حالياً و قديماً، و بالتالي يكون من المُفْتَرِضِ أنه بإمكاننا تصميم لغة برمجةٍ تتغلب على الصعوبات الموجودة في اللغات الحالية و تُضيف الجديد المُفيد إليها (على الأقل من الناحية النظرية). و هذا هو الناتج الطّبيعي لمرحلة التعلّم و التّلقّي من المُنتجين للعلم قبل مُنافستهم في إنتاج الجديد في هذا العلم.

و مثل هذه النظرة الغير مُنصِفةٍ إلى عقولنا هي من أكبر الأسباب التي تُؤدّي إلى وأد الأفكار الجيدة التي قد تُغيّر دفة الأمور في عالمنا في مجالاتٍ كثيرة، و قد تحل من مشاكلنا التي نواجهها في مُجتمعاتنا الكثير.

و هي نظرةٌ لها خاصية تغذية الذات بأثر رجعي، حيث أنه ما دُمنا ننظر إلى أنفسنا نظرةً دُونِيَّةً (حتى في إمكانية المُنافسة بعد تحصيلنا للعلم المطلوب) فإننا سنُهمل الأفكار التي كانت ستُفيدنا في حل مشاكلنا إن فَعَدْنَاها، و بإهمال هذه الأفكار سينداد الواقع سُوءاً يوماً بعد يوم، و نتيجةً لتزايد السُوء و تراكمه ستزداد نظرتنا لأنفسنا دُونِيَّةً و احتقاراً !
و هكذا ندور في حلقةٍ مُفرّغةٍ جهنمية.

22- حتى نجعل أطفالنا مؤهلين لإستعادة مجدنا السابق الذي لم نستطع إستعادته فيجب أن نعلمهم أشياء كثيرة منها تعلم اللغة الإنكليزية من الصغر بإتقان؛ عندما يتعلم اللغة الإنكليزية ويتقنها من الصغر فستفتح أمامه مجالات كثيرة سيبدع فيها، أما إن ضيقنا المجال أمامه بلغة برمجة عربية فلن تفتح له هذه الأفاق الحاسوبية الكبيرة.

إذاً الخلاصة: الغرب لما أخذوا العلم من العرب أتوا للعرب و تعلموا لغتهم و أخذوا علمهم و طوروه و نحن كعرب الآن لسنا مؤهلين لتطوير علم الحاسب عربياً لأننا لم نصل إلى ما وصل إليه الغرب في هذا العلم، فالصحيح أن ننشئ أطفالنا على تعلم اللغة الإنكليزية أفضل بكثير من أن ننشأهم على لغة برمجة عربية.

يَحْوِي هذا الإعتراض النقاط الرئيسة التالية:

- لا بد من تعليم الأطفال اللغة الإنكليزية حتى تُفْتَحَ لهم أبواب تحصيل العلم البرمجي على مصاريعها؛ نظراً لأن العلم البرمجي الآن يُكْتَبُ مُعْظَمُ إنتاجه إن لم يكن كُلُّه باللغة الإنكليزية.
- تعليم الأطفال لغة برمجةٍ عربيةٍ يُعْتَبَرُ تَضْيِيقاً لآفاق التعلّم أمام الطفل العربي.
- الصحيح: تَنْشِئَةُ الأطفال على لغة برمجةٍ إنكليزيةٍ لا على لغة برمجةٍ عربية، و ذلك حتى نفعل مثلما فعلوا هم حينما كنا الأكثر تقدماً في العلم و أخذوا هم من علمنا بلغتنا ثم طوروا فيه ليصيروا إلى ما صاروا إليه.

أما النقطة الأولى فلا جَرَمَ أنها صحيحةٌ كما قلنا في السابق، لكن النقطتين الثانية و الثالثة فيهما من التَّجَبُّي الكثير و الكثير؛ لأن إنتاج لغة برمجةٍ عربيةٍ لا يعنى أبداً عدم تعليم الأطفال اللغة الإنكليزية بل و العبرية و الفرنسية و غيرهن أيضاً؛ لأن هذه اللغات تُلزِمُهُم في باقى أمور حياتهم و ليس بالضرورة الأمور العلمية منها فقط، و قد قلتُ أيضاً من قبل أن تَعَلَّمُ اللغات الأخرى ضرورةٌ للأمة، فلا تَعَارِضُ بين الأمرين على الإطلاق.

و تحتوى النقطة الثالثة على مقولة صحيحة هي أخذ العلم من مظانّه بلغته، وإيحاءٍ خاطئٍ بأنه يجب على الأطفال منذ الصغر تعلّم الإنكليزية بالشكل الذى يكفل لهم تحصيل العلم البرمجي، وهو الأمر الذى لا يُعقل إذا ما نظرنا إلى ضرورة تدريس الأطفال ما يكفل المحافظة على الفكر الإسلامى فى عقولهم و عدم تحميلهم ما لا يُطيقون فى صغرهم.

وكذلك فهناك الإيحاء بأن وجود لغة البرمجة العربية فى حد ذاته يعوق تحصيل العلم من مظانّه، على الرغم من أن وجود اللغة البرمجية العربية التى تُطبّق النظريات التى تعلمها العرب (من تحصيلهم للعلم البرمجي بالإنكليزية) يُعد تطبيقاً للأحلام التى تحملها كلمات المُعترض عن استعادة المجد السابق!

23- أراها مضيعة للوقت؛ فاللغة الإنكليزية هي الأفضل في كل شيء يتعلق بالكمبيوتر وليس فقط بالبرمجة. لست أريد إحباطك؛ و لكن أظن أن تركيز على فهم تقنيات أخرى كتقنية **wpf** أفضل من إضاعة الوقت على إنشاء لغة برمجة عربية.

و هذا مثالٌ آخرٌ علي الهزيمة النفسية؛ فصاحب الرد يُقرّر أن اللغة الإنكليزية هي الأفضل في كل ما يتعلق بعالم الحوسبة و ليس فقط المجال البرمجي، و مع ذلك لا يقول لنا لِمَ هذا التفوق؟ و هل هو أصيلاً في اللغة أم هو نتيجةً للتقدم التقني هناك؟ ثم كانت نصيحته بترك ما "يضيع الوقت" لتفريغ لإجادة التقنيات الجديدة، و هي الدعوة الصريحة للحياة في دور المُستهلكين إلي أبد الأبد!

24- أراها مضيعة للوقت و ذلك لأن اللغة العربية ليست موحدة اللهجة و إنما هي بلهجات عديدة و اللغة الإنكليزية موحدة اللهجة فهي أقوى من حيث البرمجة.

سببٌ آخرٌ مُتَهافتٌ للغاية؛ فالبدهي أنك حينما تُصمّم لغة برمجةٍ عربيّةٍ فستجعلها بالفصحى، لكن صاحب الرد افترض أنها ستكون بلهجةٍ عاميةٍ و هو الأمر الغريب جداً! أما القول بأن اللغة الإنكليزية موحدة اللهجة فقولٌ خاطئٌ تماماً؛ فهناك الإنكليزية البريطانية و الإنكليزية الأمريكية و اللهجة الأُسكتلندية و اللهجة الأيرلندية و غيرهن الكثير. و ذلك لاتساع رقعة المُتحدّثين بالإنكليزية في العالم.

25- أخي أنا مع لغة برمجة عربية لغرض التعليم و ليس لغرض البرمجة؛ (لأن اللغات الأساسية مثل لغات **vb.net .c sharp. c++** بحر) و لأن المبتدئ و ليس لديه لغة إنكليزية قوية سوف يمل و يترك البرمجة، في حين إذا كانت لغة البرمجة عربية ربما يستمتع و يتطور و يميل إلى اللغات الأساسية.

و هذا موقف أفضل من سابقه، لكنه يتصادم مع بقية الأهداف التي نرغب في تحقيقها.

26- أعتقد أنها ستكون بلا فائدة؛ لأن اللغات التي تراها قضت عشرات السنين في التطور كما أن اللغة الإنجليزية أسهل للذين يستخدمون الحاسوب.

لست أدري لماذا يفترض الجميع أن لغة البرمجة العربية الجديدة ستكون غير حديثة، بينما لا يفترضون هذا الافتراض مع لغات البرمجة الإنجليزية التي تخرج منها كل حين واحدة جديدة!، يا أهل الخير: ليس معني أننا سنصنع لغة برمجة جديدة أنها ستكون متخلفة عن المستوي الحالي الذي وصلت إليه اللغات الأخرى؛ كل ما هنالك هو أننا سنحتاج بعض الوقت لإتمام الأدوات التي سيحتاج لها من سيرغب في البرمجة بتلك اللغة، و علي رأس تلك البرمجيات: المترجم **compiler** و المُفسر **interpreter** و المُنقِّح **debugger** و غيرهن من الأدوات الأخرى التي لا يُستغني عنها، و كذلك المكتبة القياسية للغة الجديدة. و مسألة سهولة اللغة الإنجليزية مردودٌ عليها من أكثر من وجهٍ فيما ذُكر سابقاً.

نقد لغات البرمجة العربية الحالية

حينما أخذتُ أجمعُ المعلومات عن لغات البرمجة العربية المُتواجدة بالفعل، أو حتى تلك التي كانت موجودةً ثم اختفت: فُوجئتُ بكميةٍ كبيرةٍ منها فافت ما كنتُ أظنه موجوداً بمراحل؛ فقد كنتُ في بداية الأمر أظن أن مُحاولات إنتاج لغة برمجةٍ عربيةٍ لم تبدأ إلا في السنين الأخيرة، على أساس أن معظم البلاد العربية كانت تحت الإحتلال في العقود الماضية.

ثم بعد التحرر كانت النهضة الإقتصادية هي الأهم بالنسبة للكل و كانت النهضة العلمية تتعلق بمجالات التسليح العسكري لا أكثر، و لم يكن هناك مكانٌ للحديث عن النهضة العلمية بكامل فروعها إلا في العصور الحديثة،

و لأن علوم الحاسب كانت أقل العلوم انتشاراً في بداياتها الأولى عندنا لأنها كانت تتطلب الكثير من الإنفاق، و كانت الأجهزة الحاسوبية الشخصية لم تظهر بعد للنور فكان الشكل الوحيد للحاسوب هو الحاسوب الضخم الذي يحتل مساحةً كبيرةً و لا يستطيع شراءه إلا المؤسسات الضخمة، و بالتالي لا يملك حق استعماله إلا من قَدِر له أن يعيش في بلدٍ يُنفق على البحث العلمي الكثير، و هو الأمر الذي لم يكن يدور في خلد العرب آنذاك إما بسبب الأنظمة الدكتاتورية الغاشمة التي كانت تحتل مراكز القيادة فيه احتلالاً، و/أو لانشغالهم بقضايا التحرر و هي الأهم و/أو لضعف التعليم أو لغيرها من الأسباب.

المهم أنني كنتُ أظن أن كل ذلك سَيَقِفُ في وجه ظهور مُحاولاتٍ لإنتاج لغة برمجةٍ عربيةٍ فيما مضى، و أنه نظراً لاختفاء أو قِلَّة هذه الموانع في هذا العصر فسيكون العرب أكثر إقبالاً على خوض غمار هذا الميدان، إلا أنني فُوجئتُ (و حق لى أن أفاجأ بعد الذي ظننت) أن المُحاولات العربية لإنتاج لغة برمجةٍ للعرب قديمةٌ

أكثر مما كنتُ أظن، و أنها بدأت في العصور التي تلت عصور التحرر من الإحتلال العسكري الغربي مباشرة !

وقد فاجأني كذلك العدد الكبير الذي رأيتُه لهذه اللغات؛ فحتى الآن جَمَعْتُ معلوماتٍ عن أربعين لغة برمجةٍ عربية، و هي اللغات التي توافر لي عنها معلوماتٌ على الشبكة الدولية للمعلومات (الإنترنت)، و ربما كان هناك الأكثر من ذلك و منعني عدم وجود معلوماتٍ عنها على الشبكة من التعرف عليها، لكنني أظن أن هذه اللغات التي جمعتُ عنها معلوماتٍ تكفي لتكوين نظرةٍ شاملةٍ إلى اللغات البرمجية العربية عامة و عن المسار التطوري الذي اتخذته في كل مرحلةٍ من مراحل انتشار الوعي بأهمية فرع العلم البرمجي.

وتفاوتت المعلومات التي جمعتها عن كل لغةٍ من حيث الكم و الكيف عن تلك التي جمعتها عن الأخريات، و ذلك حسب كمية المعلومات التي وجدتُها على شبكة المعلومات عن تلك اللغة، فهناك لغاتٌ برمجيةٌ كان لها النصيب الأكبر من المواقع الشبكية و الكتب التي تشرحها، و ربما كذلك شروحاتٌ صوتيةٌ و مرئية، فبالتالي كانت لي الفرصة للتعرف عليها كل التعرف، في حين أن هناك من اللغات ما لا أعلم عنه إلا اسمه و جُمْلَةٌ أو جُمْلَتَيْنِ تصفانه باختصارٍ أو حتى أقل من ذلك !

و بما جمعته من معلوماتٍ حرصتُ أول ما حرصتُ على تصنيف تلك اللغات البرمجية العربية إلى أربع مجموعاتٍ رئيسة:

الأولى هي مجموعة اللغات التي لم أحصل على قدرٍ كافٍ من المعلومات فأسميتها اللغات المجهولة، أما **الثانية** فهي فئة لغات البرمجة التي نقلها المُبرمجون العرب نقلاً عن لغات برمجةٍ غير عربيةٍ و بالأدق إنجليزية المبنى، و لم يكن لهم فضلٌ إلا ترجمة الألفاظ التقنية من اللغة الإنجليزية إلى اللغة العربية، أما الهيكل النحوي الخاص باللغة و كذا كل ما يتعلق بها تقنياً فكان مطابقاً للأصل الإنجليزي تمام المطابقة.

كما أنه في هذا النوع لم يتم إنتاج مُترجمٍ أو مُفسرٍ خاصٍ للغة، بل كل ما هناك أنه تم إنتاج برنامجٍ يُترجم البرامج المكتوبة بالصياغة العربية إلى ملفاتٍ مصدريةٍ مكتوبةٍ باللغة الأجنبية الأصلية، لذا فهذه اللغات لا تُعدُّ إلا لغات واجهةٍ لا أكثر و ليست لغات برمجةٍ عربيةٍ حقيقية.

و النوع **الثالث** هو اللغات المنقولة؛ أي التي كانت تسير على هَدْيٍ من اللغات الأجنبية أو حتى تحاكيها بصورةٍ تكاد تكون كاملة، لكنها في ذات الوقت تختلف عن اللغات المُترجمة في أنها تمتلك المُترجم و/أو المُفسر الخاص بها، و لا تعتمد على المُترجم و/أو المُفسر الخاص بلغة البرمجة الأجنبية التي تُمثّل لها الأصل الذي انحدرت منه.

أما النوع **الرابع** فهو فئة اللغات الإبداعية، و هي تلك اللغات التي لم تأخذ من أصلٍ واحدٍ فقط، بل أخذت من أكثر من أصلٍ و ربما تكون قد نقدت تلك الأصول لكي تبلغ حداً أفضل منها، مع تطوير الحلول المناسبة للعقبات التي تقف أمام اللغة العربية و التي تختلف عن العقبات التي تواجه اللغة الإنجليزية عند استخدامها في النص البرمجي. و بالطبع في هذا النوع من اللغات فإن لكل لغةٍ مُترجمها أو مُفسرها الخاص بها و الذي بُني خصيصاً لأجلها.

و هذا التصنيف لا يأخذ بعين الإعتبار الأجيال البرمجية المُختلفة للغات البرمجة، فقد نجد في فئة اللغات المنقولة لغات من الأجيال الحديثة، في حين نجد في اللغات المُبتكرة لغات من الأجيال الأقدم، فيرجى الإنتباه لهذا حتى لا تكون الأمور مُربكة و يكون استيعاب التقسيم أسهل و أفضل.

و فيما يلي فإننى أعرض كل قسم من هذه الأقسام باستفاضة لكي أوضّح المعلومات الأساسية عن كل لغة مع الحفاظ على الإبتعاد عن الصيغات النحوية لها كل البعد لأن هذا الأمر لا يهمنا هنا من قريب أو بعيد، و سأوضّح العيوب و المميزات التي أجدها في كل نوع على حدا، و إن كانت هناك لغة داخل أي فئة تتميز عن باقي لغات فئتها فسأضع لها نقدا منفردا خاصا بها.

إلا أنى كما سيتضح في تالى الكلام لست براضٍ عن تلك المُحاولات كل الرضا، أى أننى لم أجِد من بين تلك اللغات ما يُمكننى القول بكل ثقة أنه يصلح نقطة بداية لنا في مشروعنا، و الأسباب تختلف من فئة إلى أخرى و من لغة إلى أخرى، و سأوضّح فيما يلى الأسباب الكاملة لذلك في كل حالة من الحالات. و سيتضح من هذا الكلام أن هناك لغات لا تصلح لأنها ضعيفة البناء، و أن هناك لغات أخرى لا تصلح رغم قوتها و قابليتها للإستخدام بسبب عدم القدرة على تطويرها؛ لإغلاق المُطوّر لها لمصدرها و رفضه لجعل المصدر مفتوحاً رغم مُحاولات المُحاولين إثناؤه عن ذلك.

قائمة بلغات البرمجة العربية التي أعلمها:

• اللغات المجهولة:

	القول	(1)
	المحترف 380	(2)
1978	الخوارزمي	(3)
	(Arabic Programming Language (APL	(4)
	الحل (أول لغة حاسوب للعرب)	(5)
	لغتي (MyProgLang)	(6)
	دنيا	(7)
	سماء	(8)
1979	نجلاء (نظام جبيري للحاسب الآلي)	(9)
1984	كاتب CATIB	(10)
1994	السنبلة	(11)
1995	Visual Prog	(12)
1995	Arablan	(13)
1995	Arab Language	(14)
2006	عموريا	(15)
	First Arabic Visual Language (FAVL)	(16)
2007		

• اللغات المترجمة:

	laith ليث	(1)
	Phoenix	(2)
	فيجوال بازيك العربي	(3)
1978	خوارزمي	(4)
1978	غريب	(5)
1979	سلطانة	(6)
1981-1980	صخر بيسك	(7)
1986	ARABW	(8)
1986	سيناء	(9)
1988	باسكال العربي	(10)

1990	ARBI (11
1999	لوقو العربية (12
2000	باسكال المتوازي (13
2006	Arlogo(14

• اللغات المنقولة:

1986	(1 ل.ب.أ (لغة برمجة أخرى)
1998	(2 زاي

• اللغات المُبتكَرة:

1984	(1 ض
	(2 لغة برمجة منشئ المواقع SMPL (Sites Maker Programming Language)
1997-1993	(3 خبير
2001	(4 الرسالة
2001	(5 ج
2005	(6 Arabic Assembly Language
2009	(7 سوپرنوفا
2010	(8 كلمات

اللغات المجهولة

(1) لغة كَاتِب :

المُطَوَّر : د. صبرى محمود ، محمد مندورة

سنة البناء: 1984

الجهة الداعمة: جامعة الملك سعود

البلد : السعودية

نظام التشغيل : CPM

مواصفات أخرى :

ثنائية اللغة، أى أنها تدعم البرمجة بالعربية و الإنكليزية.
بسيطة البناء لتسهيل استخدامها لغير المبرمجين، كالمدرسين مثلاً بغرض تعلمها و استخدامها
بكفاية لكتابة و تنقيح الدروس التعليمية بمساعدة الحاسوب.
كانت هناك ورقة علمية تُناقش تطوير نسخة من اللغة لتعمل على نظام تشغيل الـDOS و لكنى لا
أعلم إن تم هذا الأمر أم لا.

(2) لغة دُنْيَا :

المُطَوَّر : د. زكريا صالح قاسم

سنة البناء: 1978 م

الجهة الداعمة: المؤسسة العامة للمشاريع النفطية بالعراق

البلد : العراق

لغة البرمجة الشبيهة : assembly

مواصفات أخرى :

و صف نظري للتدريس ولم تُطبَّق على أى جهاز.

(3) لغة نَجْلَاء :

المُطَوَّر : د. رضا سراج الثقة

سنة البناء: 1979 م

الجهة الداعمة: جامعة الملك فهد

البلد : السعودية

لغة البرمجة الشبيهة : لغة البيك BASIC و لكنها أقوى في الهيكلية

بنية المُعالج : أجهزة الفارابي

مُوا صَفَاتٌ أُخْرَى :

تدعم Array و متغيرات Local & Global .

(4) السنبلة (Arabic Natural Language Processing) :

المُطَوِّر : د. الأفندي

سنة البناء: 1994 م

البلد : السعودية

(5) :Arab Language

سنة البناء: 1995 م

البلد : البحرين

مُوا صَفَاتٌ أُخْرَى :

لطلاب المدارس.

(6) :Visual Prog

المُطَوِّر : د. خالد سليمان

سنة البناء: 1995 م

البلد : بولدر-أمريكا

مُوا صَفَاتٌ أُخْرَى :

رسالة دكتوراه للدكتور خالد سليمان.

(7) سماء:

المُطَوِّر : خليل الأمين عبد الجواد

بريد المطور: Khal_i_l@yahoo.com

(8) المحترف 380:

المُطَوِّر : شركة DIGITAL

بِنْيَةُ الْمُعَالِج : أجهزة PDP11 و VAX-11

مُوا صَفَاتٌ أُخْرَى :

لغة مُعَرَّبَةٌ من الحجم الوَسَط.

اللغات المترجمة

(1) لغة باسكال المتوازي :

المُطَوَّر : الطالبان: خالد المصبيح و عبد الله الدكان، تحت إشراف الدكتور عبد الملك السلطان

سنة البناء: 2000 م

الجهة الداعمة: مدينة الملك فهد التقنية السعودية

لغة البرمجة الشبيهة : باسكال

نظام التشغيل : WINDOWS

مُواصَفَاتُ أُخْرَى :

مشروع تخرج.

(2) لغة غريب :

سنة البناء: 1978 م

الجهة الداعمة: جامعة الموصل

البلد : العراق

لغة البرمجة الشبيهة : لغة البيك

بِنْيَةُ الْمُعَالِجِ : IBM Mainframe

مُواصَفَاتُ أُخْرَى :

مُوجَّهَةٌ لِلْمُبْتَدئين.

لا تستخدم الحروف العربية و تقوم بالعمليات الحسابية فقط.

المؤشّر فيها من اليمين إلى اليسار.

تستخدم مترجم بلغة فورتران.

(3) لغة ليث LAITH:

مُواصَفَاتُ أُخْرَى :

شبيهة بلغة الكوبول ومحدودة جدا.

(4) لغة سلطانة :

المُطَوَّر : شركة اوترام

سنة البناء: 1979 م

البلد : السعودية

لغة البرمجة الشبيهة : لغة البيزك
بنية المُعالِج : جهاز ZX81.

(5) لغة خوارزمي :

سنة البناء: 1979 م
البلد : كاليفورنيا بأمريكا
لغة البرمجة الشبيهة : لغة البيزك لكن بدون تكرار
نظام التشغيل : CP/M

(9) لغة صخر بيسك :

المُطَوِّر : شركة صخر
سنة البناء: بين عامي 1980-1981 م
البلد : الكويت
لغة البرمجة الشبيهة : بيزك
مُوا صَفَاتٌ أُخْرَى :
يأذن من Microsoft الكويت.

(10) لغة سينا :

المُطَوِّر : د. الأفندي
سنة البناء: 1986 م
الجهة الداعمة: جامعة الخرطوم
البلد : السودان
لغة البرمجة الشبيهة : لغة باسكال
مُوا صَفَاتٌ أُخْرَى :
ترجمةً نظرية.

(11) لغة باسكال العربي :

المُطَوِّر : د. حسن مذكور و د. أحمد محبوب
سنة البناء: 1986 م و قيل 1988 م
البلد : السعودية
لغة البرمجة الشبيهة : باسكال
بنية المعالج : VAX-11

مُوا صَفَاتٌ أُخْرَى :

قِيلَ أَنَّهُ قَدْ تَمَّ الْإِنْتِهَاءُ مِنْ عَمَلِ الْمُتَرْجِمِ، وَقِيلَ أَنَّهُ قَدْ كُتِبَ مِنْهَا الـ scanner فقط !.

(12) لُغَةُ ARABW :

سنة البناء: 1986 م

البلد: البحرين

لغة البرمجة الشبيهة: الكوبول ولكنها مختصرة.

(13) ARBI أي Arabic Basic :

سنة البناء: 1990 م

لغة البرمجة الشبيهة: GWB BASIC

نظام التشغيل: DOS

(14) لُغَةُ بَاسْكَالِ الْعَرَبِيِّ (96) :

المُطَوَّر: د.عبدالمك السلمان

سنة البناء: 1996 م

الجهة الداعمة: بمدينة الملك فهد التقنية

البلد: السعودية

لغة البرمجة الشبيهة: باسكال

نظام التشغيل: windows

مُوا صَفَاتٌ أُخْرَى :

بيئة برمجية متكاملة.

(15) لُغَةُ لُوغُو الْعَرَبِيَّةِ :

المُطَوَّر: الطالبان: الجهني و الحربي، تحت إشراف د. عبد الملك السلمان

سنة البناء: 1999 م

الجهة الداعمة: مدينة الملك فهد التقنية

البلد: السعودية

نظام التشغيل: WINDOWS

مُوا صَفَاتٌ أُخْرَى :

مشروع تخرج.

(16) لغة باسكال المتوازي :

المُطَوِّر : الطالبان: خالد المصبيح و عبد الله الدكان، تحت إشراف الدكتور عبد الملك السلطان

سنة البناء: 2000 م

الجهة الداعمة: مدينة الملك فهد التقنية السعودية

لغة البرمجة الشبيهة : باسكال

نظام التشغيل : WINDOWS

مواصفات أخرى :

مشروع تخرج.

(17) arlogo :

سنة البناء: 2006

نظام التشغيل : windows

مواصفات أخرى :

مشروع تعريب لغة لوقو arlogo.sourceforge.net

نقد اللغات المُترجمة :

- 1- ليست لغاتٍ بحق، إنما هي مجرد واجهاتٍ فقط.
 - 2- لها كل عيوب ومثالب اللغات الأصلية.
 - 3- لا تُقدِّم جديداً إلى الفكر الإسلامي باللغة العربية؛ لأنها مجرد إعادة إنتاج للقديم بشكلٍ جديدٍ فقط. ولهذا السبب فلن يمكننا أن نقوم بتطويرها إلى الأفضل لأنها ليست ملكاً لنا أصلاً، كما أن أي تغييرٍ جديٍّ لها ليجعلها تلحق بركب التقنيات البرمجية الحديثة سيقوم بتغيير شكلها، تماماً أي أننا سنقوم بابتكار لغة برمجةٍ جديدة تماماً.
- مثلما حدث مع الـ Visual Basic .NET التي يعلم كل الناس أنها ليست إلا حيلةً من microsoft حتى لا تخسر عملاءها من مبرمجي VB6، و ذلك كما يقول الأخ: تركي العسيري في كتابه الماتع (Visual Basic للجميع) حيث قال:

ماذا عن Visual Basic.NET؟

يبدو ان الحروف .NET. تشد انتباه المبرمجين بعدما وزعت Microsoft النسخ التجريبية Beta من جميع اعضاء Visual Studio.NET. حسنا، الاصدار الاخير من Visual Basic هو الاصدار السادس VB6 والذي يمثل نهاية الاسطورة Visual Basic. اما Visual Basic.NET فهي لغة برمجة جديدة لا يكمن الشبه بينها وبين اسطورتنا إلا الاسم Visual Basic وصيغ Syntax بعض الاوامر. فالاسم Visual Basic.NET ليس سوى لعبة تسويقية قامت بها Microsoft حتى لا تخسر جميع زبائننا من مبرمجي VB1 حتى VB6. فقبل ان تشد الرحال الى Visual Basic.NET، فكر بالموضوع جيدا لان شد الرحال سيكون الانتقال الى لغة برمجة جديدة، كالانتقال من VB6 الى C#.

و إذا ما أصررنا على إيجاد علاقة بين اللغة القديمة (مثل اللوقو) و التطوير الجديد: فإن الناتج لن يكون إلا مسخاً لا ينفع في شيء و لا يُحَقِّقُ أي هدف، أي أن الأفضل هو البدء من الصفر أو كما يقول المثل المُعَبَّرُ (على مياهٍ بيضاء).

اللغات المنقولة

1) لغة (ل.ب.أ) أى (لغة برمجة أخرى):

المُطَوِّر : د. فؤاد دهلوي و د. محمد مندورة

سنة البناء: 1986 م

الجهة الداعمة: جامعة الملك عبد العزيز و جامعة الملك سعود

البلد : السعودية

لغة البرمجة الشبيهة : البيزك والباسكال

مواصفات أخرى :

كُتِبَ منها Scanner فقط و لم يكتمل بناء المُترجم ثم إختفت.

2) لغة زاي:

المُطَوِّر : د. جمال الدين زقور

سنة البناء: 1998 م

الجهة الداعمة: المعهد الوطني للإعلام الآلي

البلد : الجزائر

لغة البرمجة الشبيهة: PASCAL

نظام التشغيل : DOS/WINDOWS

موقع اللغة : <http://zegour.uuuq.com>

بريد المبرمج : d_zegour@esi.dz

مواصفات أخرى :



هى لغة سهلة و بسيطة لكتابة خوارزميات بسيطة كحساب مجموع عدد ما من الأعداد الصحيحة الأولى، و البحث عن الأعداد الأولية الأصغر من عدد ما، وغيرها.

و للتمرُّن على لوحة المفاتيح بكتابة خوارزميات عديدة مثل حساب تكرُّر حرفٍ مُعيَّن، أو البحث عن الكلمات التي تبدأ بحرفٍ ما و الكلمات المُتكوِّنة من كلماتٍ أخرى، وغيرها.

التمرُّن على آلة الأعداد بكتابة خوارزميات عديدة، كالبحث عن عددٍ مُعيَّن أو أكبر عددٍ موجودٍ في مُحتوى الآلة، و الأعداد التي مُربَّعها يُساوي مجموع العددين السابقين لها في الآلة، وغيرها.

التمرُّن على بعض طرائق خَزْنِ الْمُعْطِيَّاتِ البسيطة كالجداول و القوائم الخطية المتسلسلة و البُنْي، و ذلك بكتابة خوارزمياتٍ لجرد مجموعة المُعْطِيَّاتِ المخزونة أو حذف عنصرٍ ما أو زيادته في موضعٍ معيّن.

إمكانية استعمال المُفْرَدَاتِ في صِيغٍ مُناسِبَةٍ بكتابة الكلمات في صيغة المفرد والجمع (صحيح: صحاح)، كما يُمكن اختصار المُفْرَدَاتِ (اكتب_ مباشرة: اك_مب).
و يُمكن أيضاً كتابة خوارزميات الفرز و معالجة عدة جداول و قوائم في الوقت نفسه، بفضل آلياتها المُجَرَّدة. و أيضاً كتابة خوارزمياتٍ مُتنوّعةٍ لاستعمال الملقّات أو تصميم تراكيبٍ جديدةٍ لها .

النقد المُوجّه لهذا النوع من اللغات:

- هي نقلٌ واضحٌ للغة الأصل مثل الPascal، لذا فهي لا تحمل في طيّاتها إبداعاً خالصاً. كما أنها نتيجةٌ لذلك تحمل كل عيوب هذا الأصل و تبتعد عن مميزات غيره، فلغة (زاي) مثلاً:
 - من الأجيال الأقدم و لا تجاري التقنيات الحديثة في البرمجة، مثل البرمجة الكائنية و الوراثة و التنظيم القوي لملفات البرامج بما يحسن من نوعية التكويد و غيرها، و كل ذلك لأنها نُقلت من لغة الPascal القديمة.

- مُغلقة المصدر أي أنها لن تخرج عن دماغ مُطوِّرها الذي لا يجد الوقت لتطويرها بما فيه الكفاية، فيظل بالتالي الهدف الوحيد الذي تقدر على أن تُخصّص لأجله هو التعليم (و هو الهدف الذي صرّح المُطوِّر أنه هو حقاً السبب وراء إنتاج اللغة).
و لكن حتى في هذه الحالة سوف يكون من الخطأ تعويد الأبطال على البرمجة بالأساليب القديمة و بلغة غير الإنكليزية التي سيُضطّرون إلى البرمجة بها بعد ذلك، و سيكون من المُربك حقاً التحول من العربية إلى الإنكليزية بعد التعود على الأولى.

- هذا النوع كذلك لا يُقدّم جديداً إلى الفكر الإسلامي باللغة العربية لأنه بدوره مجرد إعادة إنتاجٍ للقديم بشكلٍ جديدٍ فقط، و أي تغييرٍ جيّدٍ له ليجعله يلحق بركب التقنيات البرمجية الحديثة سيقوم بتغيير شكله تماماً، أي أننا سنقوم بابتكار لغة برمجةٍ جديدةٍ بالكلية.
و إذا ما أصررنا على إيجاد علاقةٍ بين اللغة القديمة (زاي مثلاً) و التطوير الجديد؛ فإن الناتج لن يكون إلا مسخاً لا ينفع في شيءٍ و لا يُحقّق أي هدف، أي أن الأفضل كما قلنا من قبل هو البدء من الصفر على مياهٍ بيضاء.

اللغات المُبتكرة

(1) لغة ضاد :

المُطَوَّر : د.محمد غزالي خياط بجامعة الملك فهد سابقاً و جامعة الملك عبدالعزيز حالياً
سنة البناء: 1984 م

الجهة الداعمة: مدينة الملك عبد العزيز

البلد : السعودية

لغة البرمجة الشبيهة : BASIC, Pascal, C

نظام التشغيل : DOS و يجري تطويرها على Windows

بنية المُعالج : أولاً على جهاز Cromemco و فيما بعد على الـ IBM-PC
مُوا صَفَاتٌ أُخْرَى :

تحتوي هذه اللغة على مُمَيَّزَاتٍ جَيِّدَةٍ فِي تَرَاكِيْبِ الْبَيَانَاتِ.

(2) لغة خبير:

المُطَوَّر : د.مصطفى عارف و د.حسني المحتسب

سنة البناء: بين 1993 و 1997 م

الجهة الداعمة: جامعة الملك فهد للبترول و المعادن

البلد : السعودية

نظام التشغيل : CPM

مُوا صَفَاتٌ أُخْرَى :

تم تطويرها باستخدام لغة C لتمتاز بسهولة نقلها من جهاز لآخر وبسهولة التكلفة.

(3) لغة (ج):

المُطَوَّر : د. محمد عمار السلكا

سنة البناء: 2001 م

نظام التشغيل : WINDOWS:

موقع اللغة : www.jeemlang.com

بريد المبرمج : drsalka@jeemlang.com

مُوا صَفَاتٌ أُخْرَى :

هى لغة برمجة صُمِّمت خصيصاً من أجل التعليم، وهى فى هذا الجانب تُشبه لغة (زاي) تماماً و إن كانت تختلف عنها فى أن (ج) هى إبداعٌ صرف، حقيقى أنه إبداعٌ تنقيحى إلا أنها لا تزال قريبة من قِمة هرم الإبداع العربى فى مجال ابتكار و إنتاج لغات البرمجة العربية، وقد صَمَّم المُطَوِّر برنامجاً لإنتاج و تنقيح تطبيقاتٍ بها هو برنامج (الخوارزمى).
و قد كانت هذه اللغة فى نظرى تصلح نوعاً ما للبدء بها لإنتاج لغات برمجةٍ أخرى أقوى إذا ما استمرت عجلة تطورها باضطراد، و لكن وقفت عدة عواملٍ قويةٍ فى وجه هذا الأمر ساء ذكرها فى عيوب اللغة فيما يلى.

نقد لغة ج:

- 1) من الأجيال الأقدم و لا تجارى التقنيات الحديثة فى البرمجة، مثل البرمجة الكائنية و الوراثة و التنظيم القوى لملفات البرامج بما يُحسِّن من نوعية التكويد و غيرها.
- 2) مُغلقة المصدر أى أنها لن تخرج عن دماغ مُطَوِّرها الذى لا يجد الوقت لتطويرها بما فيه الكفاية، فيظل بالتالى الهدف الوحيد الذى تقدر على أن تُخصِّص لأجله هو التعليم (و هو الهدف الذى صرَّح المُطَوِّر أنه هو حقاً السبب وراء إنتاج اللغة).
و لكن حتى فى هذه الحالة سوف يكون من الخطأ تعويد الأطفال على البرمجة بالأساليب القديمة و بلغةٍ غير الإنكليزية التى سيُضطَّرون إلى البرمجة بها بعد ذلك، و سيكون من المُربك حقاً التحول من العربية إلى الإنكليزية بعد التعود على الأولى.
- 3) ليس لها مُترجمٌ إلى لغة الآلة، بل تعتمد على آلةٍ وهميةٍ لتشغيل برامجها.

4) لغة Arabic Assembly Language:

المُطَوِّر : د. محمد العدوي، إسماعيل كشك

سنة البناء: 1425 هـ

الجهة الداعمة: جامعة حلوان - كلية الهندسة - قسم الالكترونيات و الاتصالات

البلد: مصر

نظام التشغيل: windows

مواصفات أخرى:

مطوروا هذه اللغة يُؤكِّدون أنهم قاموا بتطوير لغة أسمبلي العربية وليس تعريياً للغة الأسمبلي الإنكليزية، و يُؤكِّدون أن أوامر هذه اللغة تُترجم مباشرةً للغة الآلة بدون ترجمةٍ وسيطة.

تعمل هذه اللغة على معالجات إنتل 8086، و تستخدم تنسيق الملفات (.exe) لتوليد الكود الثنائي (Binary Code).



5 لغة سوبرنوفنا **supernova**:

سنة البناء: من يوليو 2009 إلى مارس من العام 2010 م

المُطَوِّر: م. محمود سمير فايد

نظام التشغيل: WINDOWS:

موقع اللغة: <http://supernova.sourceforge.net>

بريد المبرمج: msfclipper@yahoo.com

مواصفات أخرى:

هى لغة من لغات الجيل الخامس، أنتجها المُطَوِّر باستخدام تقنية البرمجة بدون كود (Programming Without Code Technique - PWCT) التى أنتجها هو نفسه أيضاً.



وقد تم إنتاج اللغة فى الفترة من يوليو من العام 2009 إلى مارس من العام 2010 (و أظن أن العمل لا يزال مستمراً حتى الآن لتطويرها و صيانتها).

و تهدف هذه اللغة إلى أن تكون أكثر قرباً من اللغة البشرية الطبيعية، و هى السمة العام فى لغات ذلك الجيل الحديث. كما أنها لا يبدو عليها أنها تقتصر على الأمور التعليمية فقط، بل يبدو من الواضح أنها لغة خُصِّصت لأجل الإنتاج البرمجى مثل اللغات عالية المستوى المُستخدمة حالياً مثل **Java** ، **C++** و غيرهن. و على الرغم من أن هذه اللغة تُعتبر من اللغات التى تم فيها الإعتماد على الفكر المُجَرَّد دون النقل: إلا أننى أجد عندى انتقاداتٍ حادةً للغاية لها، و ذلك كما سيتم إيضاحه فى نقد اللغة بعد قليل.

نقد لغة سوبرنوفنا **supernova**:

● الحشو فيها أكبر من المضمون المهم، لذا فمقروئيتها **readability** أقل بكثيرٍ من أى لغةٍ أخرى تتجنب هذه الجزئية، كما أن:

● الحشو فيها يُقلل من مكتوبيتها **wriTEability** كثيراً جداً، كما أنه لا يُمكن استعمال الوسائل الحديثة في بيئات البرمجة المتكاملة للمُساعدة على التغلب على هذه الصعوبة لعدم وجود كلماتٍ مفاتيحية في اللغة يُمكن للبيئة البرمجية اقتراحها على المبرمج.

و لرؤية الحشو في اللغة يمكننا أن نرى المثال التالي لبرنامج مكتوب بها:

```

I want window and the window title is Assignment.
The window height is 300. and window back color = {30,200,100}.
I want listbox and the listbox name is mylist1.
i want listbox and the listbox name is mylist2.
listbox left is 250.
[v1]. is ( Hello )( World ).
[v2]. = (15)x(3).
[v3]. equal (Country).
[v4]. = (Hello)[v1].
[v5]. = (12) and (7) or (1) not (13).
[v6]. = | (1) [One] (2) [Two] |.
[v7]. = (Hello )[GoodName].
mylist2. listbox selected
listbox must add item "[v1]. is ( Hello )( World )."
listbox must add item "[v2]. = (15)x(3).".
listbox must add item "[v3]. equal (Country).".
listbox must add item "[v4]. = (Hello)[v1].".
listbox must add item "[v5]. = (12) and (7) or (1) not (13).".
listbox must add item "[v6]. = | (1) [One] (2) [Two] |.".
listbox must add item "[v7]. = (Hello )[GoodName].".
mylist1. listbox selected
listbox must add from [v1].
listbox must add from [v2].
listbox must add from [v3].
listbox must add from [v4].
listbox must add from [v5].
listbox must add from [v6].
listbox must add from [v7].
    
```

حيث قد أخطت الكلمات ذات المعنى و الحاجة بمُستطيلٍ أحمرٍ لكي أُفترق بينها و بين الكلمات التي لا حاجة لنا بها و تُشكّل عبئاً في الكتابة و القراءة. و عبؤها في الكتابة واضح، أما عبء القراءة فيظهر حينما يكون على أن أطوّر هذا البرنامج فيما بعد؛ فحينما أحاول أن أستخلص المعلومات و الأرقام الهامة من البرنامج فسأواجه فوضى عارمة و كما مهولاً من الكلمات أتوه و سطها و لا أخرج بما أريد إلا بعناء.

و حتى لا نكون مجحفين فإن لغاتٍ أخرى مشهورة لها نفس المشكلة، مثل الـ **java** التي يشتهر بين نقادها ضرب المثال على الحشو فيها بالبرنامج التالي:

```
public class Hello{
    public static void main(String [] args){
        System.out.println("Hello CCSC");
    }
}
```

و كذلك لغة الـ **C#** التي يُمكن ضرب مثالٍ مُشابهٍ لها:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("hello, world");
        }
    }
}
```

و كل هذا لكتابة جُملةٍ بسيطةٍ على الشاشة. و لكن هذه اللغات لها هدفٌ من وراء هذا الحشو؛ و هو إجبار المبرمج على استخدام نمط البرمجة الكائني **oop** على اعتبار أنه الأُسلم له و الأفضل تقنياً، و رغم اختلافى مع مصممي اللغتين فى تلك النقطة كل الإختلاف إلا أنه لا يمكننى أن أقارن حشواً له هدفٌ بحشوٍ ليس له هدفٌ عملي.

حتى الآن لم يعرف أحدٌ هل ستكون هذه اللغة فى النهاية لغةً وظيفيةً أم لغةً كائنية. فكل الشروحات الإلكترونية التى قرأتها للمُطوّر (ما كان منها بالعربية و ما كان بالإنجليزية) لا تُقدّم أى معلومةٍ عن هذه الجزئية.

و لم يتحدث المطور عن دعم: الوراثة، معالجة الإستثناءات، الواجهات، و غيرهن من الأمور التى هى عصب لغات البرمجة و عمودها الفقرى.

يُرَكِّزُ المُطَوِّرُ و يستنفذ جهده و وقته على أشياءٍ غريبةٍ للغاية فى البناء التَّحْوِي لِلُّغَةِ، مثل:

1- خاصية الكتابة غير المنظمة و الكلمات المُتداخلة التى تجعل نصاً برمجيّاً كالتالى:

```
i
want window
and
the
window title
e is i want window.
```

و كذا النص البرمجي التالي:

IwantwindowandtheWindowtitleisHello World.

(و هي نفس الأكواد التي ضرب بها المطور المثل على هذه القاعدة) مسموحاً بها في اللغة !

مع أنه من المعروف أن هذا يُعدُّ عيباً واضحاً في اللغة إذ أنها هكذا تُساعد المُبرمج على كتابة أكوادٍ غير مقروءة بالمرّة !،

و إنني لأتساءل عن نوع المبرمج الذي سيستفيد من هذه الخاصية؟! و فيم سيستفيد بها!؟

بل إن هذه الخاصية لا تكفي بخطر إنتاج كودٍ خرافيٍّ غير مقروءٍ فقط، بل تؤثر على أداء مُترجم اللغة بشكلٍ مُخيف، حيث يقول المطور نفسه:

اذن كيف يفرق مترجم السوبرنوكا بين الكلمات حتى يصنفها ؟

هذا تحدى كبير اتداء تصميم السوبرنوكا حيث لا يكون هناك نهاية للكلمات وانما تتوقع اللغة مع قراءة كل حرف جديد ان هذا الحرف هو نهاية الكلمة تم تبدا بالبحث عن هذه الكلمة في النطاقات المسموح بها تبعاً لنطاق اخر كلمة تم قراءتها فاذا وجدت الكلمة فانها تعتبر ان هذا الحرف هو نهاية الكلمة اما ان لم تجدها فانها تواصل القراءة باستقبال حرف جديد وتعيد البحث مرة اخرى وهكذا حتى تصل الي الكلمة او الي نهاية البرنامج بدون جدوى من البحث وبالتالي نحصل على رسالة خطأ من المترجم.

و عملية البحث المستمرة في قاموس اللغة عند قراءة كل حرف تؤدي الي مايكاد ان يكون انهيار في اداء المترجم والحصول على بطء شديد في عملية الترجمة ولكن ماينقذ الموقف هو ان الكلمات المتوقعة تكون في نطاقات محددة مما يقلل فضاء عملية البحث وبالتالي زيادة الاداء تسببا عما كان سيكون عليه الحال اذا تم البحث في فضاء اكبر.

فإذا كان الأمر هيناً في البرامج الصغيرة ذات الأكواد التي لا تزيد عن 1000 سطر فإن الأمر يصير كارثياً في حالة البرامج الكبيرة، و يصير طامةً كبرى لا حل لها في حالة إنتاج حزمة برمجياتٍ مترابطةٍ معا يتم بناءها كل مرة في نفس الوقت.

2- إعطاء المُبرمج القدرة على تسمية المتغير باسمٍ طويلٍ للغاية من الممكن أن يصل إلى عدة أسطر !،

بل و يُمكن له أن يحتوي علي مسافاتٍ أيضاً !

و هو الأمر الذى لا أدرى كيف سمح به المُطَوِّر ما دام يعلم أنه من الخطأ تقنياً السماح بهذا حيث يقول فى كتابه:

والجدير بالذكر ان الحرية متوفرة فى تحديد اسماء المتغيرات ويمكن ان يشتمل اسم المتغير على مسافات او رموز ويمكن ان يكون الاسم عربى او انجليزى بل يمكن ان يكون اسم المتغير طويل جدا ويمكن ان يكون اكثر من سطر وهى حرية قد تصل الى درجة كبيرة للغاية بمعنى انها حرية زائدة عن الحاجة حيث من غير العملى ان يكون اسم المتغير طويل جدا واكثر من سطر ولكنها امكانية متوفرة ومدعومة من قبل مترجم السوبرنوا.

فالمعلوم أن واحداً من أهم الأهداف التى يضعها مُصمِّموا لغات البرمجة أثناء تصميم لغة برمجة جديدة هو الأمن، و لا يُقصد به الحماية من الإختراقات و البرامج الضارة بل يُقصد به إجبار المبرمج على استخدام طرق برمجةٍ صحيحةٍ تحميه من الوقوع فى الأخطاء البرمجية قدر الإمكان.

و لكن الأمر هنا يبدو معكوساً و كأن المطور كان يريد للغة "ميزة" جديدةً هى عدم الأمان على أساس أنه "على المبرمج أن يقوم بما على المبرمج أن يقوم به".

3- الدمج بين اللغتين العربية و الإنكليزية فى النص البرمجى؛ فما دامت سوبرنوا لغة برمجةٍ عربيةٍ تهدف إلى خدمة العرب و اللغة العربية فى مجال البرمجة فما دخل اللغة الإنكليزية فى هذا الأمر؟!.

و ما زاد الطين بلة أن المطور يقول فى كتابه:

السوبرنوا لغة برمجة عربية الهوية من ناحية البناء اما من ناحية الاستخدام فهى تحقق المعادلة الصعبة حيث تتيح ان تكون الشفرة المصدرية باللغة العربية او باللغة الانجليزية او خليط من اللغة العربية والانجليزية مما يعنى الحفاظ على الهوية العربية وفى نفس الوقت التواصل على النطاق العالمى من خلال اللغة الانجليزية

و إنى لأتساءل عن كيفية المُحافظة على الهوية العربية إذا كان الكود مكتوباً باللغة الإنكليزية؟، و أتساءل عن كيفية الحفاظ على التواصل العالمى من خلال اللغة الإنكليزية إذا كان الكود مكتوباً باللغة العربية؟،

و أتساءل عن كيفية فهم غير العرب للبرنامج المُختلط بين اللغتين الإنكليزية و العربية؟.

لذا فإننى أقول أنها ميزةٌ وهميةٌ الغرض الوحيد من وجودها هو إعطاء اللغة أرضيةً أكبر بين المستخدمين ممن يُبرمجون بالعربية مع الحفاظ على إمكانية جذب المبرمجين بالإنجليزية، و يحق لى هاهنا أن أتنبه إلى أن برامج الأمثلة التى يُرفقها المُطَوِّر فى كتابه (كتوضيح لما يشرحه

من قواعد) هى فى المقام الأول إنجليزية اللغة، و البرامج العربية تكون تالية للبرامج الإنجليزية و كأنها هناك فقط لذر الغبار فى العيون !.

و حتى القواعد اللغوية فأنها فى الأغلب تُذكر باللغة الإنجليزية ثم يُذكر لها مثالاً إنجليزي ثم يُذكر المثال العربى، أى أن القاعدة اللغوية يُذكر نصها بالإنجليزية فقط كثيراً فإذا كنت ممن يريدون استخدام النص العربى فى البرمجة بلغة البرمجة "العربية" سورنونا فىجب عليك أن تستخلص القاعدة من المثال العربى بنفسك، لأنه "يجب على المبرمج أن يقوم بما على المبرمج أن يقوم به".

3- المكتبة الخاصة باللغة ليست إلا واجهةً لمكتبة الـ xharbourminiGUI و ليست مكتبةً حقيقيةً مبنيةً باللغة نفسها كما يجدر باللغات القوية أن تكون (بل كما يجدر باللغات العربية الصرفة أن تكون).

حيث أن جانباً قوياً من حوافز إنتاج لغة برمجة عربية ينتفى عند القيام بأمر كهذا، و هو تعلم كيفية إنشاء المكتبات من الصفر، و بالتالى تفوتنا الفرصة العملية المتاحة لفهم واحدة من أهم أركان النهضة البرمجية لنا.

بل و هناك جانبٌ آخر: أن اللغة البرمجية العربية لن تكون فى هذه الحالة أكثر من واجهةً للمكتبة و إمكانياتها، حقيقى أنها ستكون واجهةً متقدمةً للغاية عن لغات الترجمة التى أوضحنها من قبل، إلا أنها ستكون لغة واجهةً فى آخر الأمر و كل ما تفعله يعود فى أغلبه الساحة فى النهاية إلى إمكانيات المكتبة الحقيقية و إبداع مطوريها و ليس لقوة لغة سورنونا و مطوريها.

● بل إن الأدهى و الأمر أن مطور اللغة لم يهتم بتطوير مكتبة قوية لها حتى الآن، و اكتفى ببضعة أوامر بسيطة أخذها من مكتبات DLL خارجية، و هى كما يقول و صفا لغته:

- تشتمل على تعليمات لمعالجة السلاسل الحرفية بمرونة.
- تشتمل على تعليمات للتعامل مع الوقت و التاريخ.
- تتيح التعامل مع الملفات بانثائها و قرانها و كتابتها و تتيح تشغيل البرامج الخارجية.
- تتيح استدعاء دوال فى مكتبات ربط ديناميكية DLL.
- تشتمل على مكتبات جاهزة للاستخدام مثل مكتبة تشغيل الصوت و مكتبة تسجيل الصوت و مكتبة تشغيل الفيديو.

و أهمل أدواتٍ كان يجب توفيرها فى الإصدارات الأولى من اللغة مثل:

- 1- هياكل البيانات الاساسية ك: المكدس stack، الطابور queue، القائمة list، القائمة المرتبة sorted list. و غيرهن.

2- إجراءات رياضية أساسية مثل: جيب الزاوية \sin ، و جيب تمام الزاوية \cosine ، و ظل الزاوية \tan ، و غيرهن.

و كان الأجدر به التركيز على هذا الأمر بشكل ضخم على الأقل ليتسنى لنا (نحن المرشّحين لتكون مسخدمين للغة) أن نختبرها جيداً، بدلاً من التركيز على الأمور التي اهتم بدعمها في لغته و أنكرناها عليه في سابق كلامنا.

7- يضع المطور خاصية "عدم الحاجة إلى تعريف المتغيرات" على أساس أنها ميزة، فيقول واصفاً اللغة:

- تتعامل مع المتغيرات بدون تعريف مسبق وتسد لها قيم ابتدائية بشكل تلقائي حسب سياق الاستخدام.
- لاتجلك تهتم بنوع المتغير (حرفي/رقمي... إلخ) وتختار المناسب حسب سياق استخدام المتغير.

و هي في نظري عيبٌ كبيرٌ من عيوب اللغة عندما تجبر المبرمج على جعل كل المتغيرات على هذا النمط؛ إذ أنها تجعل للغة خواصاً ضارةً هي:

- عدم القدرة على مُساعدة المبرمج على عدم الوقوع في أخطاء استخدام المتغيرات المختلفة، كأن يستخدم متغيراً يحتوي على قيمة نصية على أساس احتوائه على قيمة رقمية؛ فلأن اللغة لا تُقيد المتغير بنوع مُعيّن من الأنواع فهذا الخطأ و مثيلاته لن يتم اكتشافها إلا في زمن التنفيذ، و هو ما يُضيف إلينا أعباءً فوق الأعباء في مرحلة الإختبار لاكتشاف مثل هذا النوع من الأخطاء، على الرغم من أن المحاولات الجادة لتطوير عملية اكتشاف الأخطاء تُحاول جعل هذا في مرحلة التصحيح قدر الإمكان. و هذا الأمر يُعدُّ الكسر الثاني لقاعدة الأمن في تصميم لغات البرمجة.

- استهلاك قدر أكبر من الذاكرة و وقت المُعالجة من ذلك الذي تأخذه اللغات التي تُفرّق بين أنواع المتغيرات، و هذا أمرٌ مُتوقّع ما دام المبرمج ألقى جزءاً من مسؤوليته على عاتق اللغة لكي تقوم هي به نيابة عنه.

و قد يكون هذا السلوك معقولاً في برمجة مواقع الشبكة نظراً لأن تحميل الموقع يأخذ وقتاً أطول من تحميل البرنامج العادي الموجود على القرص الصلب الخاص بالجهاز، و بالتالي لا تُشكّل الزيادة التي تحدثنا عنها فارقاً نظراً لتفاهتها مقارنة بعبء التحميل من الشبكة، و لكنه ليس معقولاً أبداً حينما نتحدث عن برمجة أنظمة حساسة للوقت مثل أنظمة التشغيل و المترجمات و المُفسّرات و غيرهن من الانظمة

المشابهة (و هي الأنظمة الأولى التي يجب أن نهتم بها لإنشاء قاعدة برمجية قوية تُؤسِّس للنهضة القادمة).

8- ثالثة الأثافي⁵ (كما تقول العرب) أنه يُمكن لأكثر من كائنٍ واحدٍ أن يحمل نفس الإسم الذي يُستخدم في الكود البرمجي !!!
و هو الأمر المُتواترُ في لغات البرمجة رفضه و عدمُ عقلانيته؛ إذ لو كان لأكثر من كائنٍ نفس المُسمَّى ثم أردنا أن نُجرى عملية ما على أحدها فكيف سيستئى لنا أن نقوم بالترقية بينهما ما دام الشئ المُخصَّص للترقية بينها (و هو الاسم) فقد هذه الخاصية؟!

العجيب أن المطور يقول في كتابه:

هل من المنطق ان تحمل اكثر من نافذة نفس الاسم؟

اعلم ان هذه القاعدة قد يتعجب لها الكثير من المبرمجين لانه في معظم لغات البرمجة الاخرى هذا شيء غير منطقي وغير مسموح به ويسبب خطأ لكن هنا في السوبرنوتا مع البرمجة بوصف الخيال الامر يختلف لان الخيال يسمح لنا بذلك وانظر الى النتيجة بنفسك لقد تخيلنا اداء التطبيق ووصفنا هذا الاداء بمرونة واصبح الخيال واقع ولا داعي ان يكون الخيال مقيدا بالاساليب البرمجية المتعارف عليها في لغات لاتقوم على وصف الخيال فالاسلوب مختلف وبالتالي القواعد مختلفة نسبيا.

و كل هذا لأن "الخيال" يسمح بذلك !
و لو كان على القواعد البرمجية أن تتماشى مع "الخيال" حتى و لو كان خطأ لصارت الدنيا خراباً؛ إذ لا بد من قواعد صارمة تمنع من الوقوع في الخطأ و ليس مجازاة الخيال علي علاته.

9- ليس للغة حتى الآن خطة عمل يُمكن الركون إليها ممن ينوون التحول إلى البرمجة باستخدامها، فالجدول الزمني يُعطى وضوحاً للرؤية و مصداقية كبيرة عند التعامل مع مثل هذه الأمور، كما أنه يوضِّح التقدم في تطوير اللغة و هل يسير على الخطى المرسومة له أم لا مما يُساعد على تعديل المسار إذا لزم الأمر.

أما بدون جدول زمني و خطة عمل واضحة فإن الأمر يصير بالنسبة للمُطورين المُشاركين تسلياً لا أكثر، و بالنسبة للمُرشحين ليكونوا مُستخدمين للغة البرمجة فإنه يصير حب معرفة لا أكثر، و تتحول

5 و ثالثة الأثافي: القِطْعَةُ مِنَ الْجَبَلِ يُجْعَلُ إِلَى جَنْبِهَا اثْنَانِ، فَتَكُونُ الْقِطْعَةُ مُتَّصِلَةً بِالْجَبَلِ، وَ ذَلِكَ إِذَا لَمْ يَجْلُوا ثَالِثَةُ الْأَثَافِي. وَ بِهِ فُسِّرَ قَوْلُهُمْ فِي الْمَثَلِ: رَمَاهُ اللَّهُ بِثَالِثَةِ الْأَثَافِي: أَي بِالْجَبَلِ: أَي بِدَاهِيَةِ مِثْلِ الْجَبَلِ. قَالَ ثَعْلَبٌ، (تاج العروس من جواهر القاموس للزبيدي).

العملية كلها إلى عبثٍ لا جدوى منه حتى وإن كان المُشاركون يمتلكون حماساً كبيراً في بداية الأمر؛ فالحماس كما جرّبتنا جميعنا لا يكفي أبداً ولا بد من عقل الأمور بعقلٍ قويٍّ مُطمئنٍ.

10- اللغة من لغات الجيل الخامس، أى أنها من اللغات التى تهتم بأن يكون الكود مكتوباً بأسلوبٍ و تركيبٍ أقرب ما يكون إلى اللغة البشرية العادية.

و هذا النوع من لغات البرمجة لا يُمكن الإعتماد عليه إلا فى مجالاتٍ مُعيّنة مُخصّصٍ لها، مثل تطوير أساليب الذكاء الاصطناعى و محاولة الرقى بالبرمجيات التى تعمل على إدارة و تسيير البشر الآليين لكى يُمكنها أن تستوعب التعابير و الأوامر البشرية التى تُوجّه لها باللغة المعتادة حتى يُمكن ضم هذه الآلات إلى الخدمة مع البشر بسهولةٍ و يُسرٍ دون الحاجة إلى إيجاد لغاتٍ ذات قوانينٍ تختلف عن اللغة العادية للتحكم بها (مثل لغات البرمجة الحالية).

و لأننا (نحن المسلمين عامةً و العرب خاصةً) لم نصل إلى هذه الدرجة من الرقى بعد؛ فإن ما نحتاجه الآن هو اللغات التى يُمكننا بها أن ننافس على المستوى العالمى فى مجال إنتاج البرمجيات، و يُمكنها بالتالى إنتاج تطبيقاتٍ تتراوح بين المستوى الأدنى (أى البرمجة على مستوى الخانة **bitwise programming**) و المستوى الأعلى (أى البرمجة على مستوى عالٍ من التجريد و الوراثة و التنظيم الهيكلى للمكتبات)، و كذلك تُبنى بها اللغات المنطقية الراقية التكوين فيما بعد، و هو الأمر الذى لا يتحقق لا فى لغات الأجيال الأولى و الثانية و لا فى لغات الأجيال الرابعة و لا الخامسة و إنما تنفرد به لغات الجيل الثالث.

و الغريب أن المُطوّر يقول عن هذا الأمر:

السوبرتوفا اضافة الى العلم و بداية من حيث انتهى الآخرون و ليست اعادة اختراع للعبة فهى لغة تم بنائها باستخدام اساليب الذكاء الاصطناعي لتكون احدي لغات البرمجة من الجيل الخامس فهى تصنف ضمن فرع معالجة اللغات الطبيعية و تعد اضافة لموسسة الى هذا الفرع من العلم فهى لغة قريبة جدا الى لغة الانسان بتسبة كبيرة تفوق لغات البرمجة الأخرى المتعارف عليها قبل بناء السوبرتوفا.

أى أنه لم يُحاول فقط جعلها من لغات الجيل الخامس، بل حاول جعلها تتفوق فى مستوى القُرب من لغة البشر الطبيعية، فإذا ما سلّمنا للمُطوّر بهذا فإن الإعتراض يظل موجوداً بأن مثل هذه اللغة خطوةٌ متعجّلةٌ على طريق النهضة البرمجية و أن وقتها لم يحن بعد.

أو أنه كان من الخطأ وضعها في مجالٍ ليس مجالها، و كان يجب الإعلان عنها كلغةٍ مُخصّصةٍ غير عامة الأغراض حتى يتم تجنب معظم الاعتراضات التى أورتها هنا.

11- على الرغم من كون بساطة القواعد كان هدفاً أساسياً في تصميم اللغة فإن هناك قواعداً في السوبرنوبا تسير على العكس من ذلك، حيث أنها تكون أصعب بكثيرٍ من نظيراتها في اللغات الأخرى مثل قول المؤلف عن التعبيرات:

- يتم كتابة التعبيرات كالتالي:-
 - يتم وضع أسماء المتغيرات بين أقواس مربعة
 - مثلاً [variable name] أو [اسم المتغير]
 - يتم وضع القيم بين أقواس
 - مثلاً (Value) أو (قيمة)
 - يمكن كتابة القيم بشكل آخر للسماح بكتابة الأقواس
 - | Value | أو | قيمة |
 - يتم وضع المعاملات Operators المختلفة بين المتغيرات والقيم

و مثال واقعي على القواعد السابقة الجزء التالي من برنامج كتبه المطور:

```
[x]. = (1). and do while [x] <= (7).
    listbox must add from [x].
    Do if [x] == (5).
        [t]. = .
        [y]. = (1).
        Do while [y] <= (5).
            [t]. = [t] [y].
            listbox must add from [t].
            [y]. = [y] + (1).
        End while
    End if
    [x]. = [x] + (1).
End while
```

حيث نرى أنه قد تعسّف منتهى التعسّف حينما أرغم المُبرمج المُستخدم للسوبرنوبا على إحاطة كل متغير بقوسين [] و أجبره على إحاطة كل قيمة بقوسين () في حين أن لغة مثل الـ C++ (التي أكرهها لتعقيدها المقيت) تبدو متسامحةً إلي أقصى الحدود حينما تُقارَن بالسوبرنوبا في هذا الشأن؛ فلن يحتاج المُبرمج إلى إحاطة كل رمز من رموزه (التي ربما تبلغ في المعادلات الرياضية التي تحل برمجياً حدها الأقصى) بالأقواس و لا أن يُحيط القيم بالأقواس و لا أن يُراعى الفرق بين أقواس المتغيرات و أقواس القيم.

و يزداد الأمر تعقيداً حينما نأتي إلى التعامل مع المصفوفات حيث نرى ما يشبه الكود التالي:

```
[x]. = (1). and Do while [x] <= (10).
    listbox must add from [myarray]\[x].
    [x]. = [x] + (1).
End while
```

فالمفترض في الكود أن المطور يقوم بأخذ كل العناصر التي في المصفوفة المُسمَّاة (myarray) و يقوم بوضعها في صندوق قائمة (list box) يُسمَّى (listbox). ولكن الالفت للنظر هو كيفية استخدامه لعناصر المصفوفة؛ حيث:

- كتب اسم المصفوفة مُحاطاً بقوسين مربعين []
 - ثم تلاهما بشرطة مائلة خلفية \
 - ليكتب بعدها رقم العنصر المراد استخدامه (بالطبع بين قوسين) مربعين لأن الرقم هنا مُخزَّن في مُتغير، و لو كان قيمة مباشرة لتحتَّم عليه أن يستخدم القوسين ()
- !!!

و لو قارنَّا هذا بالـ C++ لوجدنا أنها ستكون الأكثر يُسراً مرةً ثانية؛ فكل ما هنالك أنك ستكتب اسم المصفوفة (بدون أى أقواس) ثم تكتب رقم العنصر بين قوسين [] بدون شرطة مائلة خلفية \، و القوسين الوحيدين الذين ستضطرُّ إلي كتابتهما هنا مُوحِّدا الشكل و لن يُشكِّل فارقاً أن تضع بينهما رقماً مباشراً أو أن تضع مُتغيراً رقمياً.

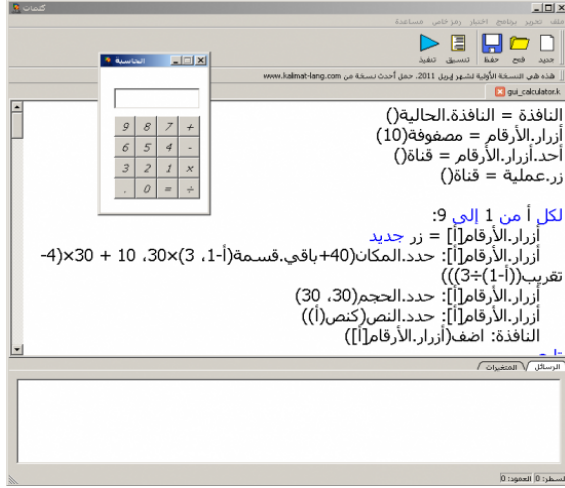
و الكودين التاليين يُلخِّصان لنا خيبة الأمل الكاملة في سهولة قواعد سورنوبا التي وُعدنا بها: حيث في سورنوبا:

```
[س] = (1). و نفذ طالما [س] >= (10).
[مصفوفة1][س] = (1).[س].
[مصفوفة1][س] = (2).[س] x [س].
[مصفوفة1][س] = (3).[س] x [س] x [س].
[س] = [س] + (1).
نهاية طالما
```

يُساوى في C++:

```
x=1;
while (x<=10)
{
    myarray[x][1] = x ;
    myarray[x][2] = x * x ;
    myarray[x][3] = x * x * x ;
    x++ ;
}
```

فما بالكم إذا ضربتُ المثل بالـ visual basic أو بالـ python؟! الأمر هنا أننا نرى أن الهدف الأصل الذي قال المطور أن لغته تهدف إلى الوصول إليه قد تبخر، و صار بالضبط كسرابٍ بَقِيعةٍ نحسبه نحن المخدوعون ماءً حتى إذا أتينا لم نجد شيئاً. بل سنجد التعتُّ الذي لا نُطيقه.



6 لغة كلمات:

المطوّر: م. محمد سامي

متأثرة بـ: python و lisp

نظام التشغيل: GNU/Linux و WINDOWS

موقع اللغة:

<http://www.kalimat-lang.com>

بريد المبرمج:

samy2004 @ gmail.com

مدونة المبرمج:

[/http://iamsamy.blogspot.com](http://iamsamy.blogspot.com)

مواصفات أخرى:

يقول م. محمد سامي عن اللغة:

كلمات لغة برمجة عربية لتعليم الأطفال البرمجة وقد راعينا فيها سهولة التعلم وقابلية البرامج للقراءة.

اللغة تشمل إمكانات جاهزة للرسم وتحريك الصور على الشاشة، وبها أيضا إمكانية التعامل مع لوحة المفاتيح والماوس. كما تأتي في بيئة تطوير متكاملة IDE ليسهل بها كتابة وتنفيذ البرمجيات.

ربما تكون كلمات لغة للأطفال لكنها ليست لعبة. هذه لغة حقيقية بأوصاف مألوفة للمبرمجين المحترفين:

1. Object-oriented programming

2. Dynamic typing

3. Garbage collection

4. عمل ملفات تنفيذية exe (انظر قائمة برنامج- < عمل ملف تنفيذي)

ولعلها تكون بإذن الله جزءاً من نهضة علمية في أمتنا.

مميزات لغة كلمات:

بدايةً: تُعد لغة كلمات أرقى لغات البرمجة العربية في فترة ما قبل ظهور إبداع علي الإطلاق، فلها الميزات التالية:

- **كاملة الهيكل**، يعني أنها لا ينقصها مكوّن من المكوّنات الأساسية التي يجب توفرها في أي لغة برمجة تُستخدم عملياً، مثل:
 - التعامل مع المتغيرات.
 - مُعاملات حسابية و منطقية **logic and arithmetic operators**
 - الجُمَل الشرطية.
 - الجُمَل التكرارية.
 - الدوال **functions**.
 - أوامر إدخال و إخراج للتعامل مع سطر الأوامر **console**.
 - أوامر للتعامل مع الملفات.
 - المكوّنات المُركّبة:
 - المصفوفات **arrays**
 - القواميس **dicts**
- **تحتوي علي مكتبة تجريبية بسيطة**، و لها الإمكانيات التالية:
 - التعامل مع الملفات.
 - الإدخال و الإخراج من/علي الشاشة.
 - أوامر رسومية ارسـم الخطوط و المستطيلات و الدوائر و النقاط.
 - أوامر التعامل مع الصور (الأطيف حسب تعبير اللغة) كالتحميل و الرسم و الإخفاء و الإظهار.
 - أوامر التعامل مع الأحداث **events** المدعومة في اللغة، مثل حوادث الماوس و حوادث لوحة المفاتيح و حوادث الأطيف.
 - إمكانيات برمجة الواجهات الرسومية.
- **لها مميزات متقدمة للغاية عن لغات البرمجة البسيطة**، مثل:
 - دعم البرمجة الكائنية **oop** بشكل جيد، ففيها:
 - تعريف الفصائل (تكافئ **classes**).
 - الوراثة **inheritance**
 - إمكانية تجميع الأكواد المرغوب في إعادة استعمالها في وحدات.

test.k
 باستخدام "welcome.k"
 رحب ("سمعان")

welcome.k
 وحدة الترحيب
 إجراء رحب (شخص):
 اطبع "مرحباً يا "، شخص
 نهاية

- قوية في جزئية الأحداث و معالجتها events and events handling
- تدعم تنفيذ أكثر من عملية في نفس الوقت (شئ يشبه في تأثيره تعدد خيوط التنفيذ (multithreading).

و للمزيد من التفصيل و الأمثلة يمكنكم زيارة موقع اللغة.

عيوب لغة كلمات:

و لكن علي الرغم من كل تلك الميزات التي لها، إلا أن كلمات لها عيوب تجعلها لا تُرضيني علي الإطلاق، و منها:

- هي من اللغات ذات التنوع المتغير **dynamically typed**. و هذا النوع لا يصلح لكتابة البرامج التي تحتاج إلي سرعات فائقة في التنفيذ، و كذلك يُساعد في الوقوع في أخطاء التكويد المنطقية لأن اللغة لا تُعيد المتغير بنوع بيانات مُعيّن (يُرْجى مُراجعة هذه النقطة في نقد لغة السوبرنوبا).
- اللغة تحتوي علي مُكوّنين في غاية التشابه هما:
 - الإجراء.
 - الدالة.
 و كان يجب ألا تُضم اللغة مُكوّنين مُتشابهين كهذين علي الإطلاق؛ حيث أن هذا لا داعي له منطقياً، بل و يُسبب تشابهما الشديد إلي البلبلة عند المبتدئين و سنري مثالا لهذا عند التحدث عن العيب رقم 5. و كان من الأفضل ضم هذين المُكوّنين في مُكوّن واحدٍ أشمل و أقوي من كليهما (و كان يُمكن للكلمات أن تحتوي علي الإجراءات فقط، و لكن مع الاهتمام بأن تأخذ الإجراءات كل جوانب القوة التي تُغني عن كل مثيلاتها مثلما فعلت مع **إبداع**).

- اللغة تحتوي علي مُكوّنِي:
 - المصفوفات arrays

○ القواميس dicts

و كان بالإمكان ضم هذين المكونين في مُكوِّنٍ واحد، وهذا سيُسَهِّلُ علي المبتدئين كثيراً لأنه يضم المكونات المتشابهة التكوين و الوظيفة في مُكوِّنٍ واحدٍ أشمل (هذه نقطة نقدٍ فرعيةٍ لكنها موجودةٌ بالفعل).

- الدوال و الإجراءات في كلمات ضعيفة البنية؛ فهي:
 - تُعيدُ خرجاً واحداً فقط (في حالة الدالة، أما الإجراءات فلا يفترض أن تُعيد مخرجاتٍ من الأصل).
 - لا يُمكن إعطاء قيم ابتدائيةٍ لمُدخَلات الدالة أو الإجراءات، و هو الأمر الذي كان بإمكانه تيسير العمل بها بشدةٍ و خاصةً في المشاريع الكبيرة.
- الفصائل في كلمات ثرثرة؛ فلكي تُعرِّف محتويات فصيلةٍ ما فيجب عليك كتابة كلامٍ كثير، و هو علي الرغم من كونه أقل مما هو عليه في حالة لغاتٍ مثل الـ Eiffel إلا أنه أكبر من المعتاد في اللغات الأخرى (مثل لغة الـ python، و هي اللغة التي تأثرتُ بها كلمات بشدة).

• مثال 1:

فصيلة شخص:

له اسم ، سن

يستجيب ل : اعرض ()

نهاية

استجابة شخص ع ل : اعرض () :

اطبع "الاسم هو " ، اسم ع ، "والسن

هو " ، سن ع

نهاية

• مثال 2:

فصيلة شخص:

له اسم ، سن

يرد على اكبر .من (أ)

نهاية

رد شخص ش على اكبر .من (الآخر):

ارجع ب: سن ش < سن الآخر

نهاية

- تحتوي كلمات علي أمري "علامة" و "أذهب إلى"، و اللذين يُكافئان "Goto" في اللغات الإنكليزية، و أنا أرفض هذين الأمرين لأنه من الممكن الاستغناء عنهما و استخدام مكوّناتٍ أخرى لتؤدي عملهما (الحلقات التكرارية و الجمل الشرطية). و كذلك فهما يُسببان الإرتباك في الكود عند استعمالهما بغزارة (يُرَجَى مُراجعة لخطاب Goto statement considered harmful للبروفيسور ديجكسترا، و الذي أرسله للنشر فى مجلة Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148).

علامة أ

اطبع "عاوز المصروف يا بابي"

أذهب إلى أ

- تعبير الحلقات التكرارية ضعيفٌ للغاية، حيث لا يُتيح تغيير مقدار الخطوة في العملية إلا بقيمة زيادةٍ مقدارها 1 (و قد نَبّه م. محمد سامي إلي أنه ينوي تدارك هذه الجزئية في الإصدارات القادمة بإذن الله عز و جل).

لكل أ من ١ إلى ١٠ :

اطبع أ

تابع

- المكتبة حتي الآن مبنيةٌ داخل المُفسر نفسه، و ليست قائمةً بذاتها و ليست مكتوبةً بلغة كلمات نفسها بل بلغة الـ C++ مع مكتبة الـ QT.

- لا يُوجد مُترجمٌ قويٌّ للغة حتي الآن، و كل ما يُوجد هو مُفسرٌ يُتيح لنا استخدام بعض مكوّنات مكتبة الـ QT، و هذا أمرٌ مرفوضٌ لأننا نحتاج إلي كتابة المكتبة الخاصة باللغة كاملةً بها نفسها ليتسني لنا نقل هذا الجزء من العلم البرمجي إلي العربية. و لكن المشكلة أنه لا يُمكن كتابة مكتبة كلمات بها نفسها كما سيلي في العيب العاشر.

- لا يُمكن كتابة أوامرٍ بلغة التجميع assembly في برنامج كلمات، و هذا يجعلها غير قادرة علي:

○ إنتاج مكتبتها باستخدامها هي نفسها.

○ إنتاج برامج تعمل علي المتحكمات الميكروية microcontrollers.

○ كتابة البرمجيات الأخرى التي نحتاج فيها إلي التعامل مع الذاكرة مباشرةً، مثل:

- أنظمة التشغيل.
 - المُترجمات compilers.
 - المُفسِّرات interpreters.
- الوحدات في كلمات تتيح لنا إنشاء إجراءات و دوال حرة (أي لا يلزم أن تكون مكتوبةً في فصيلةٍ ما)، و هو ما يجعلنا نعاني من مساوئ نموذج البرمجة الإجرائية في المشاريع الكبيرة، و هذه الصفة تناقض مبدأ الأمن الذي نريده في لغة البرمجة القوية.
 - اللغة لم تستقر في تصميمها بعد، و لم تُوضَع لها خطةٌ تُوضِّح حجمها النهائي المُتوقَّع حتي الآن، ونحن نري بالفعل أن اللغة يزيد حجمها يوماً بعد يوم.
 - ليس فيها تعبيرٌ لمُعَالِجَةِ الأَعْلَاط exception handling.
 - الوصول إلي مُكوِّنات الفصيلة غير مُوحَّد الشكل، فنحن نصل إلي المتغير الداخلي للكائن عن طريق كتابة اسمه ثم كتابة اسم الكائن بعده، مثال:

اسم م = "تامر"

سن م = ١٢

عنوان م = "شارع الورد"

حيث م هو كائن object فيه ثلاث متغيرات هي: اسم، سن، عنوان.

أما الإجراءات و الدوال فيجب أن يُكتَب اسم الكائن أولاً، ثم نتبعه بعلامة : ثم اسم الإجراء أو الدالة. مثال:

ق : اصف (١٢)

حيث ق هو الكائن، و اصف هو الإجراء أو الدالة.
 - لا تحتوي علي مُكوِّن هام للغاية هو ال enumerations.
 - لا تحتوي علي فكرة ال delegating (الموجود في لغات مثل ال C#).

الحاجة إلى لغة برمجة عربية جديدة

من خلال كل ما سبق ذكره من انتقاداتٍ للغات البرمجة العربية الموجودة حالياً: تبين لي أنه لا يمكنني الإعتماد علي أي واحدةٍ منهن في المشروع الجديد، و ذلك كما قلتُ سابقاً نتيجةً لضعف تلك اللغات أو لإغلاق المُطَوِّر لمصدرها أو لغيرهما من الأسباب الأخرى.

و بصراحةٍ فقد وافق هذا هويي في نفسي؛ لأنه يعني أنني سوف أقوم بتصميم و بناء لغة البرمجة التي أحتاجها من الصفر، و هذا معناه أن الهدف الشخصي في المشروع سوف يحصل علي تقدم هائلٍ و الحمد لله عز و جل.

كما أن هذا سيربحني بما لا يُقاس فيما بعد؛ فصحيحٌ أنه يعني أنني سأبذل جهداً أكبر بكثيرٍ جداً من الذي كنتُ سأبذله لو كنتُ قد طُوِّرتُ لغة قديمة؛ لكن في النهاية سوف يصير عندي لغة برمجة جديدة صُنعت بيدي و علي عيني من البداية حتي النهاية، و بالتالي فستحمل وجهة نظري في مجال تصميم لغات البرمجة بالكامل بدون أي نقص أو زيادة. و هذا من أكبر المكاسب علي المدى البعيد.

مِنْهَاجِ الْعَمَلِ

وَضَعْتُ الخطةَ الزمنيةَ للمشروعِ الطَّمُوحِ منذَ الأيامِ الأولى له، و قد كنتُ كلَ فترةٍ من الزمنِ أقومُ بالتعديلِ فيها بما يَتَنَسَّبُ معَ ما تعلمتُهُ من جديدٍ و ما أصبحتُ أراه هو الأهمُّ أو الأقلُّ أهميةً، فأضفتُ إليها أشياءً و أزلتُ منها أشياءً أخرى حتى استقرتُ على الشكلِ الحالي لها. و على الرغمِ من اقتناعي الشديدِ بالجدولِ الحالي للأعمالِ إلا أنني أدركُ و أعلمُ تمامَ العلمِ بأن هذا الجدولِ قد لا يكونُ نهائيًا، بل هو قابلٌ للتعديلِ فيه بالإضافةِ أو النقصانِ.

و الجديرُ بالذكرِ أن الجدولَ كان له شكلٌ مُخْتَلِفٌ تمامًا حينما بدأتُ العملَ علي المشروعِ فعليًا، و لكن الأمورَ كلها اختلفتُ بعدما تأهلتُ في مُسَابَقَةِ برنامجِ (نجوم العلوم) في موسمه الرابع؛ حيث اضطررتُ ساعتها أن أخالفَ ما كنتُ قد اقتنعتُ به بعد فترةٍ من الدراسة و التفكيرِ، و غيَّرتُ أسلوبَ العملِ بشكلٍ جذري ليتناسبَ مع ظروفِ مشاركتي في المُسَابَقَةِ !

و علي الرغمِ من أنني لم أكملِ حتي المراحلِ الغاية في التقدمِ في المُسَابَقَةِ إلا أنه بحمدِ **الله** تعالي كان في ذلك التغييرِ الخيرِ كل الخيرِ. و لو كنتُ أعلمُ أن الأمرَ سيكونُ بهذا الشكلِ لكنتُ سلكتُ تلكَ الطريقَ منذَ البداية، و لكن لا يعلمُ الغيبَ إلا **الله** عز و جل، فله الحمد في الأولي و الآخرة.

و قد أدرجتُ الخطتين في هذه الرسالة: خطة ما قبل نجوم العلوم، و خطة ما بعد نجوم العلوم؛ حتي يتسني لأهل الإختصاص (و كذا كل من يرغب) الإضطلاع عليهما لملاحظة الفارق و استخلاص الفوائد.

و الخطة الأخيرة توضح الخطوات الزمنية الكاملة للمشروع منذ بدايته و حتي المراحل المتقدمة منه، و التي يكون بعدها قد وصل إلى مرحلة البلوغ التي نرجوها له، و حينها يكون قادراً بمجتمع الضخم و منتجاته الثرية على طرُق أبواب الجديد في العلوم و المجالات البرمجية مما لم نخطط له من قبل بمُنتهى القوة و الحرية بإذن الله عز و جل.

و بالطبع فإن هناك خطوات قد تم الإنتهاء منها و نُفذت فعلياً على أرض الواقع، و لكن حتي هذه لم أشطبها من قائمة الأعمال؛ لأنني أريد من الجميع أن يعلموا كل الخطوات التي تخص المشروع سواءً أكانت قد نُفذت أم لم تُنفذ بعد حتي يكونوا على بينة من الأمر، و يكون بمقدرتهم تكوين نظرة شاملة عن المشروع و المرحلة التي وصل إليها.

و لكنني وضعت علامات تدل القارئ عن كُون كل مرحلة قد نُفذت أم لا، فالمرحلة التي تم الإنتهاء منها وضعت أمامها علامة (✓)، و التي خلت من وجود العلامة أمامها هي المراحل التي لم تُنفذ بعد و يُنتظر تنفيذها فيما بعد بتوفيق الله تعالى.

خطة (ما قبل نجوم العلوم)

- وضع قواعد لغة إبداع والتصميم المبدئي لها.
- تصميم شامل لأقسام المكتبة القياسية وتحديد جزء المكتبة الخاص بالإصدارات الأولية منها.
- صنع مُترجم كامل لإبداع بلغة الـ java:

 - تصميم وبناء الواجهة الأمامية front end والتي تتكون من الماسح scanner و السّابِر parser.
 - تصميم وبناء المُحقِّق المنطقي semantic analyzer.
 - تصميم وبناء مُنتج التمثيل الوسيط IR.
 - تصميم لغات التجميع العربية للمعالجات التي سيتم دعمها كمرحلة أولى في المُترجم.
 - تصميم وبناء مُولِّدات الأكواد التي ستُحوّل كود الشفرة الوسيطة إلى لغة التجميع الخاصة بالمعالجات المَعْنِيَّة.
 - تصميم وبناء المُجمِّعات assemblers التي ستُحوّل أكواد التجميع إلى أكواد الآلة المعنوية.
 - تصميم وبناء مُحسِّن شجرة السبر parsing tree optimizer.
 - تصميم وبناء مراحل التحسين في الكود الوسيط IR optimizing.

- وإنتاج البرامج سيكون لبيئة القنو/لينوكس gnu/linux فقط في البداية.
- بناء أساسيات المكتبة لمعالجات X86.
- بناء الأصناف الأعلى في المكتبة اعتماداً على المُكوّنات البسيطة الأولى الموجودة بالفعل. و المُكوّنات التي ستُكتَب في البداية هي الخاصة بالإصدارات رقم (1.0) من المكتبة القياسية.
- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام التشغيل windows.
- تصميم آلة إبداع الوهمية التي يُختصر اسمها لـ أبو.
- برمجة بيئة الـ أبو.
- عَرَض الأمر على الآخرين و طلب مساعدتهم.
- دعم المزيد من أروضيات العمل للمُترجم و المكتبة القياسية من المُعالجات الأكثر شهرة و انتشاراً. على سبيل المثال المُعالجات التالية:

• ARM

• AMD

• Sun sparc

• MIPS

- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام التشغيل ماك .mac.
 - تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام التشغيل سُولارِز solaris.
 - التطوير في الشفرة المصدرية لمُترجم إِبْدَاعِ لإنتاج إصداراتٍ جديدةٍ منه باستخدام لغة إِبْدَاعِ، و كلما أنتجنا إصداراً جديدةً تركنا القديمة و استخدمنا الجديدة لإنتاج القادمة، و كذا لإنتاج المُفسِّر و بيئة آبو و هكذا،
- و فائدة هذا:

- أن النظام سيكون مُحتوياً لنفسه؛ فالبيئة البرمجية مكتوبةً بذات اللغة التي خُصِّصت لها و هذا من أكبر الإثباتات على قوة اللغة و احترافيتها.
- سيجعل مُطوِّري إِبْدَاعِ مُستخدمين لها و بالتالي يُعطيهم فرصة المُممارسة العملية لها على أوسع نطاق، و كذا ملاحظة نقاط ضعفها و جوانب النقص فيها و التي تحتاج إلى مُعالجة.
- سيعطي هذا كنزاً ضخماً للمُتعلم الجديد للغة حينما يجشد أمامه شفرةً ضخمةً يستطيع من خلال قراءتها أن يتعلم كيفية البرمجة الإحترافية باستخدام اللغة الجديدة.
- سيحوّل هذا الأمر العلم البرمجي الخاص بإنتاج مُترجمات لغات البرمجة إلى علم يعتمد على اللغة العربية في أدق دقائقه و هي الكود البرمجي نفسه، و بالتالي فإن التعريب حينها سيكون أمراً مفروغاً منه و لم يتبق منه إلا الشروحات فقط.
- سيعطي هذا المُتعلمين الجدد للغة و الراغبين في التعلم العملي على المشاريع الضخمة فرصة المشاركة في تطوير بيئة البرمجة و هكذا نكسب جهود الكثيرين.

- و بالتوازي مع تطوير البيئة البرمجية سنطوِّر في المكتبات الخاصة باللغة أيضاً و لكن على نحوٍ أقل قوة.
- سينقسم فريق العمل إلى قسمين:
 - القسم الأول يُتابع تطوير البيئة البرمجية.
 - القسم الثاني يُتابع التطوير في المكتبات المُختلفة للغة.
 - تصميم لغة قواعد البيانات العربية.
 - إنتاج لغة قواعد البيانات العربية و نظام إدارتها.

خطة (ما بعد نجوم العلوم)

- ✓ - وضع قواعد لغة إبداع والتصميم المبدئي لها.
- ✓ - تصميم تجريبي شامل لأقسام المكتبة القياسية وتحديد جزء المكتبة الخاص بالإصدارات الأولية منها.
- صنع مُفسرٍ كاملٍ لمعظم المواصفات القياسية لإبداع بلغة الـ java (بحيث يتم تصحيح البرامج ثم تنفيذها مباشرة بدون إنتاج ملفات أكواد وسيطة مثل الـ java bytecode) و أسميته أبُدع :obde3

- ✓ • تصميم وبناء الواجهة الأمامية front end و التي تتكون من الفاحص scanner و السابر parser.
- ✓ • تصميم وبناء المُحقِّق المنطقي semantic analyzer.
- ✓ • تصميم وبناء عُقد التنفيذ executing node.

- ✓ - تكملة ونشر مرجع المشروع (رسالة البرمجة بإبداع).
- بناء الإصدار الأول من المكتبة.
- عَرَض الأمر على الآخرين و طلب مساعدتهم.
- تكملة بناء بقية المواصفات القياسية لإبداع، مع:
- التوسع في بناء الأصناف الأعلى في المكتبة اعتماداً على المكونات البسيطة الأولى الموجودة بالفعل.
- تحويل (المُفسر) إلي (مُترجم) بحيث يمكنه إنتاج برامج أصلية native تعمل علي المُعالجات مباشرة، و سيكون إنتاج البرامج في البداية لنظام تشغيل القنُو/لينوكس GNU/linux مع عائلة مُعالجات X86.
- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل الويندوز.
- تطوير المكتبة بحيث تتوافق مع هيكلية المُفسر/المُترجم.
- تصميم اللغة الوسيطة IR لآلة إبداع الوهمية أبو.
- برمجة بيئة الـ أبو.
- إعادة برمجة المُفسر بحيث يحتفظ بخا صية تنفيذ الأكواد انطلاقاً من الملفات النصية مباشرة، بالإضافة لخا صية إنتاج كود البرنامج بلغة آلة إبداع الوهمية أبو في البداية، و من ثم يُمكن تنفيذ هذا الكود بعدها علي الآلة الوهمية علي أي نظام تشغيل مدعوم.
- دعم المزيد من أرضيات العمل للمُترجم و بناء أساسيات المكتبة للمزيد من المُعالجات المشهورة. علي سبيل المثال المُعالجات التالية:

- ARM
- AMD

- Sun parc
- MIPS

- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل الماك.
- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل السُولاريز.
- تطوير المُترجم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل اليونيكس.
- التطوير في الشفرة المصدرية لمُترجم **إبداع** لإنتاج إصداراتٍ جديدةٍ منه باستخدام لغة **إبداع**، و كلما أنتجنا إصداراً جديداً تركنا القديمة و استخدمنا الجديدة لإنتاج القادمة، و كذا لإنتاج المُفسّر و البيئة **آبو**، و قد سبق التّوسّع في شرح هذه الخطوة و فوائدها.

- سينقسم فريق العمل إلى قِسْمَيْنِ:
- القِسْمُ الأول يُتابع تطوير البيئة البرمجية.
- القِسْمُ الثاني يُتابع التطوير في المكتبات المُختلفة للغة.
- تصميم لغة قواعد البيانات العربية.
- إنتاج لغة قواعد البيانات العربية و نظام إدارتها.

الخطوط العريضة لإدارة المشروع

- لم يتم التحدث في النقاط السابقة عن أمورٍ في غاية الأهمية. هن:
 - كيفية الحصول علي التمويل الكافي للإستمرار في العمل،
 - المرحلة التي سيتم فيها جعل المشروع مفتوح المصدر،
 - الهيكل الإداري للمشروع و كيفية التعاون بيني و بين من يرغب في المشاركة فيه،
 - كيف يُمكن لمن يرغب في المساعدة أن يفعل هذا.
- لذا ففي هذا الجزء سيتم شرح هذه المسائل باختصارٍ بإذن الله عز و جل.

أطوار المشروع:

سيكون للمشروع طَوران من أطوار الحياة، الطَور الأول هو: طَور العمل الفردي، و الطَور الثاني هو: طَور العمل المؤسسي الجماعي.

• الطَور الفردي

المرحلة الأولى من حياة المشروع، و فيها ستكون مُنتجاته مُغلقة المصدر و سيكون العمل علي تطويرها فردياً تماماً؛ حيث سأكون المُبرمج الوحيد فيه، و الوحيد الذي يتعامل مع التساؤلات و/أو الإقتراحات و/أو الدعايا و/أو النقاشات المُختلفة لمن يهتمون بالمشروع (سلباً أو إيجاباً). و سيتم تمويل المشروع عن طريق التبرعات⁶ من الأفراد و المؤسسات، و شراء المنشورات الورقية الخاصة بالمشروع (مثل هذا الكتاب).

• الطَور المؤسسي

و فيه سيتم تحويل المشروع من عملٍ فردي إلي مُؤسسة لها كيانٌ خاصٌ بها، و ذلك لرعاية المشاريع مفتوحة المصدر التي ذُكرت من قبل في غايات مشروع البرمجة بإبداع، و سيتم قبول مُشاركات من يرغب في تطوير الاكواد المصدرية لمُنتجات المؤسسة، مع الأخذ في الاعتبار أن أسماء المُنتجات و شعاراتها سيتم اعتبارها علاماتٍ تجارية (تماماً كما يحدث في حالة نواة اللينوكس linux kernel).

أما التمويل فسيكون مُعتمداً علي الطرق التالية:

○ التبرعات

6 لمعرفة كيفية التبرع للمشروع يُمكنكم زيارة الموقع الرسمي له:

http://www.ebda3lang.blogspot.com/p/blog-page_2244.html

- شراء المنشورات الخاصة بالمشروع، مثل كتب تعليم البرمجة بلغة **إبداع**، و كتب تعلم التعامل مع المُفسِّر والمُترجم و بيئة البرمجة المتكاملة و غيرهن من المُنتجات الأخرى.
- بيع خدمات التنزيل **installing** و الصيانة **maintaining** للمُؤسَّسات التي ترغِب في استخدام المُنتجات البرمجية للمشروع. سواءً أكانت تلك المؤسسات تعليمية أم غير تعليمية.
- بيع خدمات التدريب للأفراد، و للمُؤسَّسات التي ترغِب في تدريب مُوظَّفيها علي المُنتجات البرمجية للمشروع، و كذا إجراء امتحاناتٍ تقويميةٍ لإعطاء شهاداتٍ مُعتمَدةٍ من المُؤسَّسة لمن يرغِب في ذلك.

مُوعِدُ الإِنْتِقَالِ مِنَ الطُّورِ الْفَرْدِيِّ إِلَى الطُّورِ الْمُؤَسَّسِيِّ:

هذا الأمر يعتمد بشكل كامل علي ما يتوفر من تمويل، فإذا كان هناك عجزٌ ما دِيَّ شديد: فسأخذ الأمر فترةً غير صغيرةٍ للتطور، أما لو توافرت الموارد المادية فسيكون بالإمكان الإسراع بهذه الخطوة بشكلٍ كبيرٍ يتناسب مع مقدار الدعم الموجود.

كيفية أخذ القرار في المُؤسَّسة فيما بعد:

هذا أمرٌ سابقٌ لأوانه، و لكن ما يُمكن تأكيده أن أخذ القرارات في كل مشروع من المشاريع المُختلفة سيكون بمنهج التشاور الواضح بين أفراد مُجتمَع المُطَوِّرين له، مثلما يحدث في مُجتمَع نواة اللينوكس (بل ربما يعتمد علي ذات الآليات التي يعتمدون عليها في عملهم). و لو حَدَثَ أن انفرد مالك العلامة التجارية للمشروع بقراراتٍ لا تراها الأغلبية: فستظل لهم القدرة علي الانفصال و إنشاء مشروعٍ مُوازٍ مبني علي آخر الإصدارات التي وصل إليها المشروع، و هذا بفضل الرخصة المفتوحة التي ستعتمد عليها المُؤسَّسة في عملها يا ذن الله عز و جل.

هل ستقتصر المُؤسَّسة علي البرمجيات فقط في المستقبل؟

هناك نيةٌ صادقةٌ لديّ لتحويل المُؤسَّسة إلي ما يُشبه شركة **apple** في عالم الحوسبة، بحيث تكون قادرةً علي إنتاج أنظمةٍ متكاملةٍ بكل ما تشمله من برمجياتٍ و عتادٍ صلبٍ **hardware** تعمل عليه تلك البرمجيات.

و لا أقصد بالعتاد الصلب هنا أجهزةَ الحاسوب العادية فقط، بل أعني كذلك كل الأجهزة الرقمية القابلة لإعادة البرمجة، سواءً أكانت حواسيباً أو هواتفاً جَوَّالَةً أو حتي أجهزةَ تحكمٍ رقميٍ قابلةٍ للبرمجة **plc**. أو غيرهن.

علي أن هذا قد يستغرق فترةً طويلةً حتي يصير حقيقةً ملموسة، لما يحتاجه من متطلباتٍ تكاد تجعل الأمر يدخل في عداد المستحيل، لكن علي أية حال: النية موجودةٌ و جاهزةٌ للتحويل إلي واقعٍ متي ما أمكن فعل ذلك.

القاموس

بعض الإصطلاحات العربية المُستخدمة ليست هي الأشهر في ترجمة مُرادفاتها التقنية الإنجليزية. و إنما استخدمتها لأنها أدق و أوضح من حيث المعني، بينما الترجمة المشهورة حَرْفِيَّةٌ أكثر من اللازم.

الإصطلاح العربي	الإصطلاح الإنكليزي المكافئ
برمجة على مستوى الخانة	Bitwise programming level
برمجة كائنية	Object oriented programming
بيئة برمجة متكاملة	Integrated development environment (IDE)
جيب الزاوية (جا)	Sin
جيب تمام الزاوية (جتا)	Cos
خوارزم	algorithm
خيوط التنفيذ	Executing threads
طابور	queue
ظل الزاوية (ظا)	tan
قائمة	list
قائمة مرتبة	Sorted list
كائن = نسخة	object = instance
كود أصلي	native code
كود محكوم	managed code
مُتحكِّمات ميكروئية	Microcontrollers
مُترجم	compiler
مُعالجة الأغلط	exceptions handling
مُفسِّر	interpreter
مَقْرُوءِيَّة	readability
مَكْتُوبِيَّة	writability
مُكَدِّس	stack
نظام تشغيل	Operating system
نواة (في نظام تشغيل)	kernel
واجهة	interface

inheritance	وراثَة
debugger	مُنقِّح
Free software	برمجيات حرة
Programming language theory	نظرية لغات البرمجة
Application programmer interface (API)	واجهة مُبرمج التطبيقات
bug	عِلَّة
unix-like	شبيه اليُونِكس
Assembly language	لغة التجميع
processor	مُعَالِج
assembler	مُجَمِّع
Programming paradigm	منهج برمجي
scanner	ما سح
parser	سَابِر
readability	مقروئية
writability	مكتوبية
Semantic analyzer	مُحَقِّق منطقي
Intermediate representation (IR)	تمثيل وسيط
Object code	الكود الهدف
optimization	تحسين
productive	مُنتِج
List box	صندوق قائمة
Arithmetic operator	مُعَامِل حسابي
Logic operator	مُعَامِل منطقي
function	دالة
console	سطر أوامر
array	مصفوفة
dict	قاموس
event	حَدَث
class	فصيلة = صنف
Events handling	مُعَالَجَة الأحداث

multithreading	تعدد الخيوط
Dynamic typing	تنويع مُتغَيِّر
Front end	واجهة أمامية
computational thinking	التفكير الحوسبي

المصادر

- سُنَن الترمذي رحمة الله عليه.
- Wikipedia the free encyclopedia.
- مقدمة كتاب (العبر، و ديوان المُبتدأ والخبر، في أيام: العرب، و العجم، و البربر، و مَنْ عا صرهم من ذوي السلطان الأكبر) لابن خلدون رحمة الله عليه، و تُعرَف اختصاراً بـ(مقدمة ابن خلدون).
- مقال (مقارنة بين ثلاث لغات برمجية عربية جيم، زاي، لوجو) لكتابه: طه زروقي.
- العرض التقديمي المُسمَّى (تطوير بيئة لغة البرمجة العربية APL) من إعداد: أيمن بن فهد السند و سعد بن عبدالله السلامة.
- (Arabic programming languages) by: sami sarhan.
- العرض التقديمي المُسمَّى (تعريب لغات البرمجة (لغة ج)) من إعداد: فيصل الزامل و عبدالله الغنيم، تحت إشراف الدكتور: عبدالملك السلطان.
- الموقع الرسمي للغة (ج): <http://jeemlang.com>
- العرض التقديمي (لغة البرمجة العربية خبير)، جامعة الملك فهد للبترول و المعادن، كلية هندسة و علوم الحاسب، من إعداد: فهد علي القحطاني.
- كتاب (لغة خوارزمي للحاسب الإلكتروني)، تأليف: عبد الفتاح جمال عبد الحفي، الطبعة الثانية 1406هـ - 1986م، شركة الرائد للحاسبات الإلكترونية.
- الموقع الرسمي للغة (زاي): zegour.uuuq.com
- كُتِب (كتابة مبرمج لغة ض)، تأليف: خليل الأمين عبد الجواد.
- كتاب (Design of Arabic programming language)، تأليف: عبد العظيم أحمد عمّوري، إشراف: لبنى بدري فهد، Philadelphia University Faculty of Engineering Department of Computer Engineering
- كُتِب (CATIB:A Bilingual Authoring Language with Hypertext and Multimedia Capabilities)، تأليف: صبري عبد الله محمود و محمد محمود مندورة، في (مجلة الحاسوب السعودية) المجلد 6 العدد 6
- كتاب (بيئة البرمجة "كاتب" و تطبيقاتها في تطوير نظم تعليم حاسوبية ذكية)، تأليف: صبري عبد الله محمود و محمد محمود مندورة، في (الندوة الثانية لتعريب الحاسوب، جامعة الملك سعود، شَوَّال 1414 هـ).
- كتاب (لغة البرمجة سوبرنوبا) الإلكتروني العربي للمهندس: محمود سمير فايد.

- شرائح العرض الإنكليزية المَعنونة (Supernova_Abstract) و المضمّنة مع ملفات الإصدار رقم 1.2 من لغة البرمجة سوبرنوبا للمهندس: محمود سمير فايد.
- كتاب (Supernova Programming Language) الإلكتروني الإنكليزي للمهندس: محمود سمير فايد.
- العرض التقديمي (كلمات: لغة البرمجة العربية الجميلة)، ل.م. محمد سامي.
- كُتَيْب (تعلم كلمات بالأمثلة) تأليف: م. محمد سامي.
- الموقع الرسمي للغة (كلمات): www.kalimat-lang.com
- كتاب (البرمجة باللغة العربية)، تأليف: أ. عمر مكداشي. شركة منشورات: دار الراتب الجامعية،
- موقع الشبكة: <http://itwadi.com> و المُشارَكَات و المُناقِشات التي دارت عليه حول إنتاج لغة برمجة عربية.
- كتاب (Visual Basic للجميع) الإلكتروني للمؤلف: تركي العسيري.
- عدة مواقع شبكية أخرى للأسف لا أذكرها بالضبط و إن كنت قد استفدت منها علمياً.



القسم الثاني:

عن الموالصفاح القياسية للغة

البرمجة العربية إبداع

الإصدار (1.0) من اللغة

شرح المواصفات القياسية للإبداع

هذا الجزء ليس على هيئة كتب المعايير القياسية المشهورة رغم أنه وُضِعَ للقيام بما يُقَارِبُ هذا الدور، و ليس على شكل كتب الشروح التعليمية للغات البرمجة التي تُدرَّس أو تُباع للمُبرمجين العاديين الراغبين في تعلم اللغة الجديدة رغم أنه أيضاً وُضِعَ للقيام بما يُقَارِبُ هذا الدور؛ فهو يَمْزِجُ بين الشكلين السابقين و يَصْلِحُ للقيام بدوريهما بكل كفاءة.

فمن ناحية المعايير القياسية سوف تكون كل القواعد مشروحةً بِلُغَةٍ واضِحَةٍ كل الوُضُوح لتُحدد ما المسموح به في اللغة و ما غير المسموح به، و ما هي شروط صحة كل عملية (إذا كان لها شروط صحة معينة).

و لكن رغم ذلك فإن أسلوب الشرح غير جافٍ كما هو في كتب المعايير القياسية للغات البرمجة، بل هو سَلِسٌ سهل التناول و مدعومٌ بكم كبيرٍ من الأمثلة التي تُوضِّحُ القواعد عملياً، و بالتالي فهذا الجزء يَصْلِحُ ككتابٍ تعليميٍّ للمُحترفين في لغاتٍ أخرى و الراغبين في تعلُّم لغة **إبداع** كاملةً بسرعةٍ كبيرةٍ و بمنتهى الكفاءة، كذلك،

و أقولُ المُحترفين لأنه يُشترطُ في الراغب في تعلم **إبداع** من خلال هذا الجزء أن يكون على درايةٍ بكل المواضيع التي نتحدث عنها فيه؛ حيث أنه لا يشرح أصل الموضوع بتمعن، بل يتطرق إلى موقف لغة **إبداع** منه فقط و يستفيض في شرح هذا الموقف. و في الفترة القادمة سيكون هناك كتابٌ آخرٌ بإذن الله - تعالي خاصٌ بتعليم البرمجة للمبتدئين من الصفر حتى الاحتراف و فيه سأقوم بشرح كل قواعد **إبداع** باستفاضةٍ و توسع.

و كذا سيكون هناك فيما بعد كُتَيْبٌ منفصلٌ بِإِذْنِ اللَّهِ عَزَّ وَجَلَّ أُقَدِّمُ فِيهِ طَرِيقَةً جَدِيدَةً لَوْ صَفَّ نَحْوُ لُغَاتِ الْبَرْمَجَةِ وَقَوَاعِدَهَا غَيْرَ الطَّرِيقِ الْمَعْتَادَةِ الْمَعْرُوفَةِ حَالِيًا، وَهِيَ فِي نَظَرِي أَفْضَلُ مِنْ تِلْكَ الطَّرِيقِ الْمَعْتَادَةِ بِكَثِيرٍ (عَلَى الْأَقْلِ بِالنِّسْبَةِ لِي)، وَفِي ذَاتِ الْكُتَيْبِ سَأَقُومُ بِشَرْحِ تِلْكَ الطَّرِيقَةِ الْجَدِيدَةِ بِمُنْتَهَى الْإِسْتِفَاضَةِ وَالشَّمُولِيَّةِ لِيَرْجِعَ إِلَيْهِ الْقَارِئُ الْكَرِيمُ عِنْدَ الرَّغْبَةِ فِي فَهْمِ أُسَاسِيَّاتِ الطَّرِيقَةِ الْجَدِيدَةِ لَوْ صَفَّ نَحْوُ لُغَاتِ الْبَرْمَجَةِ.

و هَذَا الْجُزْءُ سَيَتَضَمَّنُ قَوَاعِدَ الْإِصْدَارَةِ الْأُولَى فَقَطْ مِنَ اللَّغَةِ، فَلِأَنَّ اللَّغَةَ لَمْ يَكْتَمَلْ بِنَاوِهَا بَعْدَ فَمَنْ الْمُمْكِنِ أَنْ تَحْدُثَ بَعْضُ التَّغْيِيرَاتِ فِي أَجْزَاءِ التَّصْمِيمِ الْمَوْجُودَةِ حَالِيًا وَالتِّي لَمْ تُنْفَذْ بَعْدَ، وَلِذَا قَمْتُ هُنَا بِعَرَضِ الْمُكُونَاتِ الَّتِي تَمَّ الْإِنْتِهَاءُ مِنْ بِنَائِهَا بِالْفِعْلِ فِي الْإِصْدَارِ الْحَالِيِّ مِنْ أُبْدِعَ.

صفات اللغة:

هى لغة

بسيطة: عدد قواعدها كأقل و أسهل ما يكون، و يكفى كتابٌ متوسط الحجم لتغطية نحو اللغة بشكل كامل،
سهلة و تعليمية: يُمكن استخدامها كلغة تعليمية في مجال البرمجة للأطفال الصغار بسهولة و سرعة. فلقد
 رُوِيَ هذا الجانب بشدة أثناء تصميمها،
هجينة: لها صفاتٌ مُشتركةٌ تمزج بين صفات اللغات الإجرائية و لغات البرمجة الكائنية. و تُحاول
 استخلاص الأفضل من كل نوع من النوعين،
قوية: لأنها تُراعى القوة في كل مناحى تصميمها،
آمنة و نظيفة: تختار للمبرمج أفضل الاختيارات و الأدوات و تُتيحها له ليستخدمها في برامجها،
متعددة الخيوط: أى بإمكانها إنتاج برامج ذات خيوط تنفيذٍ مُختلفة،
تفسيرية و مترجمة: أى يُمكنها أن تُنتج كوداً محكوماً **managed code** و كوداً أصلياً **native code**
 باستخدام نفس النص البرمجي **الإبداعي** و نفس المكتبات،
مفتوحة المصدر: فصناعة مُترجم أو مُفسر لها أمرٌ مُتاحٌ للجميع حسب التوضيحات الواردة في الرخصة
 التى تلى هذه الصفحة، فيُرجى الإلتباه لهذه الوثيقة الفائقة الأهمية،
مستقرة: فلا نية هناك على الإطلاق للإضافة إليها إلا بمعدلاتٍ بسيطةٍ جداً و عند الحاجة الماسة لذلك، و
 على العموم فالتغييرات ستكون قليلةً للغاية إن حدثت.

رخصة مُواصفات لغة البرمجة (إداع) في الإصدار (1.0):

تخضع مُواصفات لغة البرمجة العربية إداع في إصدارتها رقم 1.0 للبنود التالية:

- يُمكن لأى أحد أن يُطالع المواصفات الخاصة باللغة و أن ينشرها دون أى قيودٍ أو عوائق ما دام الأمر خارج نطاق الإستغلال التجاري، و لكن يجب الحصول علي إذن مُصمّم اللغة إذا كان النشر تجارياً.
- الإسم العربي للغة (إداع) و كذلك الإسم الإنكليزي (ebda3) و اسم المُفسّر الخاص بها بالعربية (أُدع) و ترجمته الإنكليزية (obde3) تُعتبر علاماتٍ تجاريةٍ لمُصمّم اللغة و لا يجوز التعدي عليها.
- صناعة مُترجم compiler أو مُفسّر interpreter لها أمرٌ مُتأخّ للجميع بشرط بناء كافة المُواصفات الخاصة بها في هذه الإصدار.

مبادئ التصميم التي تم اعتبارها أثناء تصميم إبداع:

- استخدام الكلمات و الرموز المألوفة، و تقليل الرموز و العلامات التي لا تنتمي إلى الحروف العربية قدر الإمكان لتكون سهلة التعلم و الإستيعاب.
- تحويل النص البرمجي إلى ما يُقارب الوصف العادي باللغة العربية الحديثة، حتى أن أى شخص غير مُتخصِّصٍ يُمكنه فهم الكثير بمُجرد قراءة البرنامج المكتوب **إبداع**. و لكن مع المُحافظة على البُعد عن الحشو الموجود فى اللغة العادية و الذى لا يُناسب صِفَتَي الإنتاجية و المكتوبية الكبيرتين للُّغة.
- مزج التعبيرات و المُكوّنات البرمجية المُتقاربة فى مُكوّنٍ أو تعبيرٍ واحدٍ أشمل و أعم لتكون كذلك سهلة التعلم و الإستيعاب. و لتفادى البلبلة و الخلط الناجمين عن التشابه الصياغى بين تلك المُكوّنات أو التعبيرات.
- الاستغناء عن أى مُكوّنٍ أو تعبيرٍ يزيد من قواعد اللغة بدون تقديم فائدةٍ يُميّز بها و يكون الوصول إليها بالمُكوّنات الأخرى صعباً للغاية.
- الإِستغناء عن أى مُكوّنٍ أو تعبيرٍ يُدخِل المبرمج المُستخدم للُّغة فى تفاصيلٍ فنيةٍ تخص المُترجم أو نظام التشغيل أو العتاد الصلب، أو بطريقة تنفيذ البرنامج بشكلٍ عام. و مثل ذلك الـ Preprocessor الخاص بعائلة الـ C (مثل الـ C و الـ C# و الـ C++) و الذى لم نأخذ منه إلا أقل القليل.
- و الطريقة الوحيدة التى يُمكن للمبرمج بها الوصول إلى مثل تلك التفاصيل هى الأدوات المُتوقّرة فى المكتبة القياسية **إبداع**. و بالطبع فإن هذا يُحافظ على انتهاج المبرمج نهجاً برمجياً أفضل.
- جَمْعُ الأفضل من كل لغة برمجة و عدم الإهتمام بتصنيفٍ مُعيّن للُّغة يحد من صفاتها و ميزاتها، كأن نجعلها لغةً كائنيةً صرفةً أو لغةً وظيفيةً صرفةً فنُحَلِّلها عيوب النوع الذى خُصِّصت له و نُفقدُها ميزات النوع الذى لم تتم له، و ذلك بالطبع عند القدرة على الجَمْع (و هى المُتوافرة فى الغالب) أما عند عدم القدرة على الجمع فيتم اختيار الخاصية الأفضل و الأكثر انسجاماً مع باقى قواعد اللغة.

● عدم الإعتراف بالمبدأ القائل بأن مُصمِّم اللغة ليس عليه أن يبتكر أشياءً جديدةً بل هو مُجرد جامع للصفات الجيدة في لغات البرمجة الموجودة فعلاً في اللغة الجديدة، و بالتالي اعتماد أى صيغة أو فكرة يتم الإقتناع بأحقيتها و جدارتها بالضم للغة الجديدة.

● مُساعدة المُبرمج على إنتاج كودٍ آمنٍ عن طريق:

- 1- مُعالجة الأغلط.
- 2- تعبيرات و مكوّنات قوية تُساعده على التغلب على الأجزاء الصعبة من مهمة التكويد و تُوفّر عليه العناء.
- 3- مبدأ التوارث الذي يُعطى المُبرمج القدرة على الإستفادة من جهد الآخرين و إنتاجهم الذي تم التأكد من أمنه.

قائمة اللغات التي تم الإستفادة منها عند تصميم إبداع:

فى كل مرّة يتم فيها الإعلان عن لغة برمجة جديدة تكون هناك تلك المقولة التى مللناها من كثرة ترادها و هى أن: المُصمّم "إختارَ أحسنَ ما فى اللغات الأخرى و وَضَعَهُ فى اللغة الوليدة"، و فى مُعظم الأحيان حينما ندقّق فى الأمر نرى أنه يُخالف أغلب توقعاتنا، حتى نشك فى أنه لم يكن إلا فقاعة إعلامية فارغة ليس إلا!، و أن الهدف الوحيد من ورائه كان دفع الناس للإلتفاف حول اللغة الجديدة و عقد المُقارنات بينها و بين اللغات القديمة فىكون انتشارها أسهل و أقوى.

لكننى أجد نفسى غير قادرٍ على التهرب من فعل نفس الشيء هنا؛ فأنا بالفعل حاولت قدر الإمكان إعطاء إبداع الميزات التى فى اللغات الأخرى و تلافى عيوب تلك اللغات بحيث تُصبح بالفعل اللغة الجامعة للصفات التى ينبغى أن تكون فى لغات البرمجة عالية المستوي عامّة الأغراض.

و فيما يلى قائمة بلغات البرمجة الموجودة بالفعل و التى استفدتُ منها بشكل واضح أثناء تصميم إبداع: إما معرفة لمُكوّناتٍ مُفيدة تُضم إليها، و إما معرفة بمُكوّناتٍ سيئة يجب رفض ضمها إلى أى لغة برمجة. كما سيتضح من الأجزاء التالية لجزء و صف اللغة فى هذه الرسالة و التى تُناقش القرارات التصميمية المُختلفة التى اتخذتها أثناء التصميم.

فى مكانٍ تالٍ فى هذه الرسالة سوف أتطرق بإذن الله تعالى إلى شرح قناعاتى المُختلفة فى مجال تصميم لغات البرمجة بشكل مُستفيض، و من ثم أتطرق منها إلى ذكر الدوافع المنطقية التى جعلتني أتخذ القرارات التصميمية التى جعلت إبداع على الشكل الذى هي عليه الآن، و فى مكانٍ آخر سوف أذكر المُكوّنات التى تُوجد فى لغات برمجةٍ أخرى و تشتهر بها و رفضتُ ضمّها إلى إبداع و أسباب ذلك الرفض، و ذلك كتوضيح تام لتأثرى بتلك اللغات و إن كان ذلك التأثير قد حدث بصيغةٍ سالبةٍ لا موجبة.

قائمة اللغات:

- C#
- Java
- Matlab
- C++
- Visual basic
- F#
- ج

- Fortran
- Ada
- Pascal
- python
- Eiffel
- C

هَيْكَلُ الْبَرْمَاجِ فِي اللُّغَةِ

سوف أذكرُ في هذا الفصل التكوينَ العامَ للبرماج المكتوب **بإبداع** بكل ما يُمكن أن يحتويه من حيث المُكوّنات التعبيرية أو من حيث الملفات، وقواعد ضم تلك المُكوّنات و الملفات إلى بعضها البعض و استخدامها سوياً، و نظراً لأنه لم يتم بعد شرح المُكوّنات فإنني أتنبه إلى أن هذا الجزء سيتضح معناه و الأجزاء الغامضة فيه أكثر عند المرور على الفصول التي تليه؛ و التي تحوى شروحات مُكوّنات اللغة على نحوٍ تفصيلي. فيرجى التنبه إلى هذا.

تكوين البرنامج من الملفات files و المُجلّدات folders:

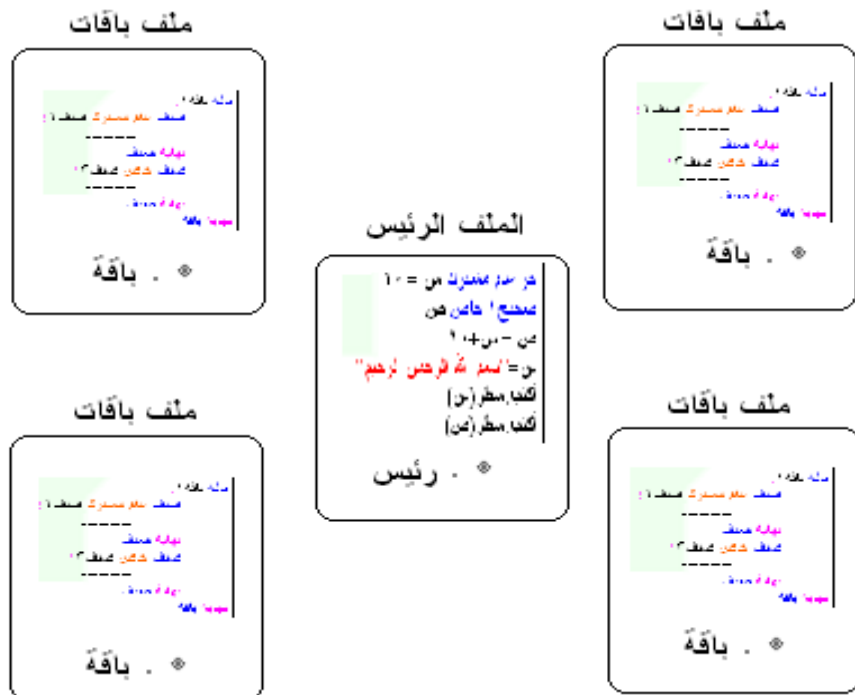
يتكون من ملفات نصية (ملف واحد أو أكثر) تكتب بصيغة اليونيكود (utf-8)، و ملفات البرنامج لها أحد النوعين:

• الملف الرئيس:

و هو الملف الذي يحتوى على المسار الرئيس لسير البرنامج و الذي يبدأ التنفيذ به، و امتداد الملف الرئيس هو (* .رئيس) و لا يُمكن للمشروع الواحد أن يحتوى على أكثر من ملف رئيس واحد.

• ملف باقات:

و هو الملف الذي يحتوى على كود صنف واحد (كما سيلي توضيحه في الجزء الخاص بالأصناف). و يُمكن للبرنامج أن يحتوى على أي عددٍ من ملفات الباقات، و له الامتداد (* . باقة).



و يُمكن لملفات الباقيات أن تُوضَعَ:

- داخل مُجلَّد البرنامج (و هو نفس المُجلَّد الذي يُوجد به الملف الرئيس)، أو:
 - داخل مُجلَّداتٍ أُخري (سواءً أكانت تلك المُجلَّدات الأُخري موجودةً في مُجلَّد البرنامج أم لا)، و تُسمَّى هذه المُجلَّدات الفرعية بالـ(باقيات)، و يُمكن للبرنامج أن يحتوي علي أي عددٍ من الباقيات.
- و إذا كانت هناك باقيةٌ نرغب في استخدامها: فيجب ضمها باستخدام أمر **أضم** الذي سيلي التنويه إليه.

تكوين الملف الرئيس باختصار:

يتكون الملف الرئيس من العناصر التالية:

[أوامر ضم المكتبات (إن وُجِدَت)]
[أوامر البرنامج]
[الإجراءات المنفردة (إن وُجِد أيٌّ منها) و/أو تعريفات الألقاب المنفردة (إن وُجِد أيٌّ منها) و/أو]
[الأصناف (إن وُجِد أيٌّ منها)]

توضيحات:

- الترتيب أمرٌ لازم الإلتباع على النحو سابق التوضيح، فالإجراءات و/أو الأصناف و/أو الألقاب لا يُمكنها أن تسبق أوامر البرنامج، و هكذا فإن أي مخالفةٍ للترتيب سيُنتج عنها خطأً مُناسبٌ من المُترجم.
- كل أمرٍ في **إبداع** ينتهي بانتهاء سطره، و لا يُمكن أن يمتد الأمر الواحد إلى أكثر من سطرٍ واحدٍ إلا إذا كان الأمر نفسه مُركَّباً (مثل جملة **لو** الشرطية مثلاً) أو تم استخدام مازج الأسطر (الذي سيلي ذكره بالتفصيل).
- يتم ضم أي باقيةٍ من باقيات المكتبة باستخدام الصيغة التالية:

أضم الباقية_1_الباقية_2_الباقية_3

حيث تحتوي الباقية (الباقية_1) على الباقية (الباقية_2) و التي تحتوي بدورها على الباقية (الباقية_3). و معني أمر الضم أن مُكوّنات هذه الباقية (من أصنافٍ و ألقاب) يمكن استعمالها بشكلٍ مُباشرٍ و كأنها قد تم تعريفها داخل الملف الرئيس نفسه، بدون الاضطرار إلي كتابة اسم الباقية اولاً ثم يليه اسم المُكوّن المرغوب في استخدامه في كل مرةٍ نرغب في استخدام مُكوّنٍ داخل الباقية.

مثال: بدلا من الجمل الطويلة التالية:

الباقية_1_الباقية_2_الباقية_3 اسم. صنف 1

الباقية_1_الباقية_2_الباقية_3 اسم. صنف 2

الباقية_1_الباقية_2_الباقية_3 اسم. صنف 3

يمكننا كتابة

أضم الباقية_1 الباقية_2 الباقية_3

و بعدها نستخدم الأصناف الداخلية مباشرة كما يلي:

اسم. صنف 1

اسم. صنف 2

اسم. صنف 3

- (أوامر البرنامج) تحتوي على الأوامر التي سيتم تنفيذها في البرنامج مثل استدعاءات الإجراءات **procedures calls** و استنساخات الأصناف **classes instantiatings** و/أو الألقاب **enumerations** و إشهار المتغيرات **variables definitions** و غيرهن من الأوامر، أى أنها هى المسار الرئيس للبرنامج الذي يشبه محتويات الدالة **main** في لغات عائلة الـ **C**. و يجب في كل برنامج وجود أمر واحد علي الأقل في ذلك المسار.
- مُدخلات البرنامج عند تشغيله من سطر الأوامر تُوجد في جدول (سيتم شرح معني الجدول فيما بعد) نوع عنا صره هو **نص**، و هو جدول ذو بُعد واحد يُسمى (**دخول**). و يتم التعامل معه كأبي جدول نصي عادي.
- و يحتوي على المعاملات **arguments** التي تم تمريرها للبرنامج عند استدعائه. بدايةً من العنصر صاحب المُميز **index** ذي القيمة (**1**) إلى آخر عنا صره.
- الإجراءات و/أو الأصناف و/أو الألقاب الموجودات في الملف الرئيس يُمكنها أن تَري بعضها البعض بشكلٍ عادي، و لكن لا يُمكن لأبيها أن يَري المتغيرات و/أو الثوابت و/أو الجداول و/أو كائنات الأصناف و/أو كائنات الألقاب التي تُوجد في مسار البرنامج الرئيس.

تكوين ملف الباقات:

يتكون كل ملف باقةٍ من العنا صر التالية:

[أوامر ضم المكتبات (إن وُجدت)]
[تعريفات الألقاب المنفردة (إن وُجد أي منها)]
[تعريف الصنف]

توضيحات:

- يجب أن يحتوي كل ملفٍ على صنفٍ واحدٍ (و واحدٍ فقط) له نفس اسم الملف، فإن لم يحتو على أى صنفٍ حصل المُبرمج على خطأٍ نحويٍّ من المُترجم.
- الألقاب المنفردة التي يتم تعريفها في ملف الباقية تُعتبر ضمن مكوّنات الباقية ذاتها، بحيث متي ما تم ضم الباقية إلي البرنامج (باستخدام الأمر **أضم**) فإنه يُمكن استخدام تلك الألقاب بشكلٍ مباشرٍ و كأنه قد تم تعريفها داخل البرنامج.

شروط التسمية فى إبداع:

هذه الشروط يجب أن تتوافر فى الأسماء المُعطاة من قِبَل المُستخدم، مثل:

- المُتغيّرات المُختلفة التى يُشهرها،
 - الإجراءات،
 - الألقاب و كائنها،
 - الجداول،
 - أسماء ملفات الباقيات،
 - الأصناف و كائنها،
- و غيرهن من المُكوّنات المُستحدثة التى يكتب هو أسماءها بنفسه.

و الشروط هى:

- يُمكن أن يحتوى الإسم على تجميعية من الحروف و الأرقام و النقطة،
 - عدم البدء بأرقام أو بنقطة،
 - عدم الإحتواء على أي علامةٍ سواءً أكانت من العلامات المحجوزة فى اللغة (و هن المكتوبات فى السطر التالي) أو غيرهن
- # + - _ { } () < > < > | : \ / ، =**
- ألا يكون الإسم من الكلمات المحجوزة **reserved words** مثل إجراء و صنف،
 - أن يكون الإسم مُتصل الحروف و النقاط و الأرقام، أى لا يحتوى على مساحات بيضاء مثل المسافات **spaces** أو الإزاحات **indentations**.
 - ألا يكون مُتكرراً داخل حَيِّز الأسماء **namespace** الخاص به (أى تم استخدامه من قبل مع مُتغيّرٍ أو إجراءٍ أو صِنْفٍ أو خلافه داخل ذلك الحَيِّز).

مازج الأسطر:

فى إبداع ينتهى سطر الأوامر بالنزول إلى سطرٍ جديد (و ليس كما فى عائلة الـ C التى تنتهى الجمل الأمرية فيها بالفاصلة المنقوطة ؛)، و لكن يُمكن باستخدام علامة | جعل المُترجم يَعتبر السطر التالى جزءاً من السطر الحالي حتى يبلغ تلك العلامة، و يُعتبر ما بعدها تعليق.

مثال 1:

حساب الحساب.الأول = حساب(50000
(20

و هذا يُماثل تماماً كتابة:

حساب الحساب. الأول = حساب (20 50000)

مثال 2:

متغير 1 = متغير 2 + إجراء 1 (معامل 1) | هذا تعليق
 | معامل 2
 | معامل 3

و هذا يُماثل كتابة:

متغير 1 = متغير 2 + إجراء 1 (معامل 1 معامل 2 معامل 3)

أنواع البيانات في إبداع:

تنقسم أنواع البيانات في إبداع إلى:

- أنواع بيانات أصلية تُسَمَّى المتغيرات و الثوابت و كائنات الألقاب،
- أنواع بيانات مُركَّبة تتألف من تجميعة من النوع السابق، و هي:
 - الجداول،
 - كائنات الأصناف،

و سيأتي ذكر عنا صر هذين النوعين بالتفصيل فيما يلي.

التعليقات:

الصياغة:

الصياغة	الإستخدام	المُكافئ في عائلة السى
تبدأ ب \ و تنهى ب /	لسطرٍ واحدٍ أو لأ سطرٍ متعددة	/* */

قواعد الاستخدام:

يُمكن استخدام التعليقات في أى مكانٍ داخل الكود.

مثال:

\ هذا مثالٌ لتعليق سطرٍ واحدٍ /

\ هذا مثالٌ

علي

تعليقٍ لعدة أسطرٍ /

الْمُتَغَيِّرَاتُ وَ الثَوَابِتُ

هن ما يُسمي بالـ **variables** و **constants** علي الترتيب في اللغات الإنكليزية، و المتغيرات عبارة عن حاوياتٍ لقيَم (و ليست مؤشراتٍ **pointers** لقيَم)، بمعنى أنه إذا ساوينا متغيراً ما بمتغيرٍ آخر فإن هذا يعني أننا أسندنا قيمة الثاني إلي الأول بدون ربطٍ بينهما زيادةً عن ذلك، و لو تغيرت قيمة الثاني بعد المساواة فإن قيمة الأول لا تتأثر مطلقاً. أما الثوابت فهن متغيراتٌ لا تتغير قيمتها الأولي نهائياً.

الصياغة:

النوع مستوى. الوصول الإستنساخ الثبات المتغير الأول = قيمة المتغير الثاني = قيمة

الأنواع:

النوع	الشرح
رقم	و هو نوعٌ عامٌ يصلح لكل الأنواع الرقمية الموجبة و السالبة، و الصحيحة و الكسرية، و يقوم المُترجم compiler أو المُفسر interpreter الخاصين بإبداع بالتكفل باستنتاج مساحة المُتغير الرقمي و ما يحتاجه من الذاكرة تلقائياً.
نص	يُشير إلى سلسلة نصيةٍ تصلح لاحتواء النصوص ذات الحرف الواحد (التي تُدعي char في لغة java) و النصوص عديدة الأحرف.
منطق	يأخذ أحد القيمتين (صحيح) و (خطأ).
حر	يُشير إلى أي نوع من الأنواع الفاتئة، و كذلك إلى أي كائنٍ مُسْتَنْسَخ من الأصناف أو الألقاب (لم يتم بناؤه في الإصدار 1.0 من المُفسر interpreter الخاص باللغة).

مستويات الوصول:

لو تم تعريف المتغير في صنفٍ يكون لها المعاني التالية:

المستوى	الشرح
عام	يمكن الوصول إليه من أي مكان.
داخلي	يمكن الوصول إليه من داخل صنفه و الأصناف الوارثة له فقط.
خاص (الافتراضي)	يمكن الوصول إليه من داخل صنفه الحاوي له فقط.

الاستنساخ:

الصفة	الشرح

<p>لو تم تعريف المتغير في صنف: فلا يمكن صنع نسخة منه لكل نسخة من صنفه الحاوى له، بل يتم إنشاء نسخة واحدة لكل نسخ الصنف و تُستخدم بإسنادها إلى أي كائن من كائنات صنفها.</p> <p>أما لو تم تعريفه في إجراء: ففي كل مرة يتم فيها استدعاء ذلك الإجراء يكون ذلك المتغير محتفظاً بالقيمة التي تم إسنادها له في آخر استدعاء سابق لذلك الإجراء.</p> <p>و في كلا الحالتين فإن القيمة التي يتم إعطاؤه إياها في نفس سطر تعريفه تُعطي له في أول مرة فقط، بينما في المرات التالية يكون محتفظاً بآخر قيمة أُسندت إليه كما تم شرحه لكلا النوعين.</p>	<p>مشترك</p>
<p>لو كان في صنف: فسوف تُصنع نسخة منه لكل كائن من النوع الحاوى له و تُعتبر كل نسخة متغيراً حقيقياً مستقلاً في الذاكرة، أما لو كان في إجراء فسيتم إعادته إلى القيمة الافتراضية في كل مرة يتم فيها استدعاء ذلك الإجراء.</p>	<p>بدون صفة (الافتراضى)</p>

الثبات:

الشرح	الصفة
<p>قيمه من النوع الثابت أى لا يمكن تغييرها، و هذا النوع يجب إعطاؤه قيمة ما في نفس سطر تعريفه و إلا اعتُبر خطأ نحويًا.</p>	ثابت
<p>قيمه من النوع المتغير أى يمكن تغييرها</p>	بدون صفة (الافتراضى)

مثال 1:

رقم س ص

مثال 2:

رقم ص س = 10
ص = س + 5
أكتب، رقم. سطر(ص)

خرج مثال 2:

15

تحويل قيمة منطقية إلى قيمة نصية:

يُمكن ذلك عن طريق استخدام الإجراء القياسي (منطق.إلى.نص) الذي له الترويسة التالية:

إجراء (نص النص) منطق.إلى.نص (منطق المنطق)

مثال:

أكتب نص. سطر ("القيمة المنطقية هي " + منطق.إلي.نص (صحيح))

الإجراءات القياسية الخاصة بالتعامل مع النصوص:

هناك عددٌ من الإجراءات القياسية في اللغة للتعامل مع النصوص هي كما يلي:

- الإجراءات (حرف.رقم) صاحب الترويسة:

إجراء (نص الحرف) حرف.رقم (رقم الرقم نص النص)

و يُستخدم في الحصول علي حرفٍ معينٍ من داخل النص عن طريق رقم ذلك الحرف داخل النص. مع العلم أن الترقيم يبدأ برقم 1 كما في الجداول التي سيلي ذكرها.

- الإجراءات (طول.النص) صاحب الترويسة:

إجراء (رقم الطول) طول.النص (نص النص)

و يُستخدم للحصول علي طول نصٍ معينٍ (أي عدد الأحرف و المسافات الفارغة فيه).

- الإجراءات (اقتباس) صاحب الترويسة:

إجراء (نص الاقتباس) اقتباس (رقم البداية رقم النهاية نص النص)

و يُستخدم في الحصول علي جزءٍ من النص يبدأ بالحرف صاحب الرقم (البداية) و يتضمنه، و ينتهي بالحرف صاحب الرقم (النهاية) و يتضمنه.

مثالٌ علي استخدام تلك الإجراءات:

أكتب نص.سطر ("طول النص (بسم الله الرحمن الرحيم) هو " + إلي.نص(طول.النص("بسم الله الرحمن الرحيم"))
أكتب نص.سطر ("الحرف رقم 2 من (محمد) هو " + حرف.رقم(2 "محمد")
أكتب نص.سطر ("الكلمة الثانية في جملة (بسم الله الرحمن الرحيم) هي " + اقتباس(5 8 "بسم الله الرحمن الرحيم"))

و خرج هذا البرنامج:

طول النص (بسم الله الرحمن الرحيم) هو 22

الحرف رقم 2 من (محمد) هو ح

الكلمة الثانية في جملة (بسم الله الرحمن الرحيم) هي الله

التحويل من و إلي النوع حر:

النوع حر يماثل النوع var في اللغات الإنكليزية، و يمكن إسناد أي قيمةٍ إليه مهما كان نوعها، لكن لإرجاع تلك القيمة إلي نوعها الأصلي فإن هذا يحتاج إلي قواعدٍ خاصة، تلك القواعد لم يتم الانتهاء من تصميمها في إبداع لأنه لم يتم بناء النوع حر فيها حتي الآن (أي في الإصدار 1.0).

المُعامِلات المنطقية و الحسائية

المُعامِلات المنطقية:

المُعامِل	الشرح
و	تكافئ AND في التعبير الإنكليزي
أو	تكافئ OR في التعبير الإنكليزي
ليس	تكافئ NOT في التعبير الإنكليزي
=	هل يساوى (إذا جاءت في جملة منطقية)، تُكافئ = في لغات عائلة C
#	لا يساوى
<	أكبر من
>	أقل من
=<	أكبر من أو يساوى
=>	أقل من أو يساوى
><	لا يساوي
<>	لا يساوي

المُعامِلات الحسائية:

المُعامِل	الشرح
+	جمع
-	طرح
×	ضرب
÷	قسمة
=	للمساواة
^	أس

الأُسبِقِيَّة في التنفيذ:

- (1) الأقواس،
- (2) الأسس،
- (3) المُعامِلات المنطقية = < = > # > < < > في
- (4) المُعامِل المنطقي ليس،
- (5) المُعامِل المنطقي و،

- (6) المعامل المنطقي أو.
- (7) الضرب و القسمة أيهما أسبق من ناحية اليمين،
- (8) الجمع والطرح أيهما أسبق من ناحية اليمين.

الإجراءات

نبذة بسيطة:

الإجراء هو كتلة من الأوامر تُكْتَب كوحدة واحدة و يُطْلَق عليها اسمٌ مُعَيَّن يتم استدعاؤها به لتُنْفَّذ في كل مرة نود تنفيذها، و يمكننا تمرير مُعَامِلَاتٍ مُعَيَّنة النوع إلى الإجراء لكي يعمل عليها، و يمكننا الحصول على أكثر من مُخْرَجٍ من الإجراء الواحد.
و هو يُمَاتِل الـ **function** في اللغات الأخرى مع بعض الاختلافات.

الصياغة:

إجراء النوع مستوى الوصول الاستنساخ التغطية (المُخْرَجَات) اسم الإجراء (المُعَامِلَات):
| جسم الإجراء /

النوع:

الصفة	الشرح
ثابت (الافتراضي)	هو الإجراء العادي المعروف في باقي لغات البرمجة و الذي يُنْفَّذ الكود المكتوب فيه فقط عند استدعائه بشكل صحيح.
حر	هو الإجراء الذي يقبل وضع استدعاءات لإجراءاتٍ أخرى في آخر كوده في زمن التنفيذ، بحيث تُنْفَّذ بعد الكود الثابت المكتوب قبل زمن التنفيذ. و يقبل كذلك حذف الاستدعاءات التي وُضِعَتْ فيه أثناء زمن التنفيذ. و يتفق مع الإجراء الثابت في كل صفاته بينما يمتلك خصائص أخرى لا تشاركه فيها الإجراءات الثابتة، و سيأتي شرح تلك الصفات فيما بعد. يُشَبِّه هذا النوع الـ delegations في الفكرة الأساسية.
مخصص	هو الإجراء الذي لا يحمل إلا كوداً مكتوباً بلغة تجميع assembly code خاصة بمعالج معين يتم تحديد اسمه بالطريقة التي سيأتي شرحها في بند (الإجراءات المخصصة)، و الإجراء المخصص هو إجراء ثابت و لكنه لا يمكن أن يحتوى على كود مكتوب بإبداع، بل يجب أن يكون كل الكود مكتوب بلغة التجميع (لم يتم بناؤه بعد في الإصدار رقم 1.0 من مُفسِّر اللغة).

مستويات الوصول:

إذا كان الإجراء في صنف فيكون معناها:

المستوى	الشرح
عام (الافتراضي)	يمكن الوصول إلي الإجراء من أى مكان.
داخلي	يمكن الوصول إليه من داخل صنفه و الأصناف الوارثة له فقط.
خاص	يمكن الوصول إليه من داخل صنفه فقط.

الاستنساخ:

إذا كان الإجراء في صنف فيكون معناها:

الصفة	الشرح
مجرد	ليس في الإجراء كود: بل مجرد الترويسة التي فيها و صف للإجراء و خواصه فقط.
مشترك	لا تُصنع منه نسخة لكل كائن مُستنسخ، و لا يجوز داخله استخدام أي مكون من مكونات الصنف الغير مشتركة.
بدون صفة (الافتراضية)	هو الإجراء المُستنسخ العادي.

التغطية:

إذا كان الإجراء في صنف فيكون معناها:

الصفة	الشرح
سقف	لا يمكن تغطيته في الأصناف التي ترث صنفه.
بدون صفة (الافتراضية)	هو الإجراء العادي الذي يقبل التغطية.

مثال 1:

إجراء عام مشترك (رقم س منطوق ص) ماعددالحالات(نص الحرف نص المحتوى):
\ الكود /

مثال 2:

إجراء عام مجرد (رقم أ) أوجدالجذرالتريبعي(رقم س)

و مهما كان عدد مُدخَلات أو مُخرجات الإجراء فيجب في كل الأحوال إحاطتهما بقوسين و إلا حصل المبرمج على خطأ نحوي، أما إذا كان الإجراء لا يُعيد أي مُخرَج أو لا يأخذ أي مُدخَل فلا يمكننا أن نكتب

كلاً من و أياً من القوسين الخاصين بكل حالة فارغين بل يجب ألا نكتبهما على الإطلاق. أي أن الأشكال التالية مسموحٌ بها جميعها في إبداع:

الشكل الأول:

إجراء عام مشترك إجراء ما:
\ الكود /

الشكل الثاني:

إجراء عام مشترك (رقم أ) إجراء ما:
\ الكود /

الشكل الثالث:

إجراء عام مشترك إجراء ما (رقم س):
\ الكود /

بينما الأشكال التالية خاطئة ولا يُسمح بها في إبداع، و تؤدي للحصول على خطأٍ نحويٍّ من المترجم:

الشكل الأول:

إجراء عام مشترك () إجراء ما():
\ الكود /

الشكل الثاني:

إجراء عام مشترك () إجراء ما:
\ الكود /

الشكل الثالث:

إجراء عام مشترك إجراء ما():
\ الكود /

المُدخَلات و المُخرِجات:

يُمكن للإجراء في إبداع أن يُعيد أكثر من قيمة، فإذا ما أخذنا الإجراء (ماعددالحالات) كمثالٍ لنا فسنجد أننا سنستخدم المتغيرات المُشهرّة كمُخرِجاتٍ (في هذه الحالة س، ص) بصورةٍ عاديةٍ في الإجراء، و آخر قيمٍ ستكون مُخزّنةً فيهما ستكون هي ما سيتمّ إعادته عند نهاية تنفيذ استدعاء الإجراء.

و يتم استدعاء الإجراء كما في المثال التالي:

رقم عددالحالات
منطق يمثل.أغلبية
عددالحالات، يمثل.أغلبية = ماعددالحالات("م" "بسم الله الرحمن الرحيم")

حيث نفترض أن الإجراء (ماعددالحالات) هو نفسه في المثال 1، و بالتالى نرى أن شروط استدعاء الإجراء متعدد الخرج في حالة عدم الرغبة في إهمال خرج ما أو أكثر من مُخْرَجَات الإجراء هي:

1. وجود متغيرات لها نفس نوع و ترتيب مُخْرَجَات الإجراء فى الجانب الأيمن من أمر استدعاء الإجراء.

2. الفصل بين كل متغيرين فى الجانب الأيمن من الاستدعاء بفاصلة (،).

و عدم توافر شرط من الشروط السابقة فى استدعاء الإجراء متعدد الخرج يؤدى إلى ظهور الخطأ البرمجي المناسب من المصحح اللغوي.

أما فى حالة الرغبة فى إهمال خرج ما أو أكثر من مُخْرَجَات الإجراء فهو أمر له شروط معينة و سيأتى ذكره بعد قليل فى فصل (إهمال خرج الإجراء عند استدعائه).

تمرير الجداول كمعاملات للإجراءات و مُخْرَجَات منها:

1. الجداول كمعاملات للإجراءات:

إذا ما رغبتنا فى تمرير جدول للإجراء كمعامل له فسنستخدم الصيغة التالية فى معاملات الإجراء:

النوع {عدد الأبعاد} الاسم

مثال:

إجراء (رقم س) ماعددالحالات (نص س نص {1} ص):

\ الأوامر /

2. الجداول كمُخْرَجَاتٍ من الإجراءات:

إذا ما رغبتنا فى تعريف جدول للإجراء كمُخْرَجٍ منه فسنستخدم الصيغة التالية فى مُخْرَجَات الإجراء:

النوع {عدد الأبعاد} الاسم

و هذه هى تقريبا نفس الصياغة التى تُستخدم فى الأصل لتعريف الجداول لكن بدون وجود صفات مُعَيَّنَةٍ هى: مستويات الوصول و الاستنساخ، و التى سيتم شرحها فيما بعد عند التعرض للجداول نفسها.

تحميل الإجراءات **overloading**:

هو إمكانية تعريف أكثر من إجراء فى حيز أسماء واحد (مثلاً: نفس الملف الرئيس أو نفس الصنف الحاوي) لهم نفس الاسم، و لكن يجب أن يكون هناك اختلاف بين الإجراءات المُحْمَلَّة فى عدد و/أو أنواع و/أو ترتيب أنواع قيم المُدْخَلَات.

فإذا لم يكن هناك أى اختلاف بين الإجراءات المُحَمَّلَة يندرج تحت الاختلافات السابقة الذكر فسُيُعد وجود الإجراءات المُحَمَّلَة فى حيز تسمية واحدٍ خطأ من قِبَل مترجم اللغة.

أمثلةً على تحميلاتٍ صحيحةٍ لإجراءات:

إجراء (رقم س منطق ص) إجراء ما (رقم س):
 \ جسم الإجراء /
 إجراء (رقم س منطق ص) إجراء ما (نص س):
 \ جسم الإجراء /

إهمال خرج الإجراء عند استدعائه:

إذا أردنا استدعاء إجراء يُعيد قيمةً أو أكثر ولكن بدون استخدام بعض أو كل تلك القيم (باستقبال قيمتها فى متغيرات) فيمكننا ذلك بالأشكال التالية:

1. الاستغناء عن كل المُخْرَجَات:

وفى هذه الحالة يمكن استدعاء الإجراء بكتابة اسمه مباشرةً ثم تليه قائمة المُعَامِلَات المُرسَلَة إلى الإجراء.

مثال: الإجراء (إجراء ما) له تحميلٌ واحد:

إجراء ما (50) \ تم إهمال جميع مُخْرَجَات الإجراء /
 إجراء ما ("نص")

إجراء (منطق ص) إجراء ما (رقم س):
 \ جسم الإجراء /
 إجراء (رقم ع نطق ص) إجراء ما (نص س):
 \ جسم الإجراء /

2. الاستغناء عن بعض المُخْرَجَات:

يكون ذلك بعدم كتابة أى شئ فى مكان الخرج المرغوب فى إهماله فى جملة استدعاء الإجراء. و يُمكننا تكرار هذا مع أى مُخْرَجٍ نرغب فى إهماله حتى لو كررناها مع جميع مُخْرَجَات الإجراء.

مثال:

ع = إجراء ما (5) \ تم إهمال الخرج الأول فقط للإجراء /
 ع = إجراء ما (50.0) \ تم إهمال جميع مُخْرَجَات الإجراء /

إجراء (منطق ص رقم ك) إجراء. ما (رقم س):

\\ جسم الإجراء /

إجراء (منطق ص رقم ك) إجراء. ما (نص س):

\\ جسم الإجراء /

إسناد قيم ابتدائية لمدخلات و مخرجات الإجراء:

فى حالة ما إذا أردنا أن نعطى المتغيرات التى يُمرّرها المستخدم إلى الإجراء (أى المُدخّلات) أو المتغيرات الناتجة عنه (أى المُخرجات) قيمةً ابتدائيةً معينةً بحيث تُعطى لها فعلياً إذا:

- لم يُمرّر المستخدم للإجراء قيمةً مكافئةً لذلك المُدخل،
- أو لم يتم إعطاء قيمةً للمُخرج فى داخل الإجراء فى زمن التنفيذ.

يمكننا أن نفعل ذلك باتباع الصيغة التالية:

إجراء النوع مستوى. الوصول الإستنساخ التغطية (خرج 1 = قيمة 1....) اسم. الإجراء (معامل 1 = قيمة 1....):

\\ جسم الإجراء /

بالنسبة للمُدخّلات:

إذا مرّر المستخدم للإجراء قيمةً للمُعامل الأول فقط: يتم اعتبار قيمة المُعامل الثانى كما وُضعت فى القيمة الافتراضية، و إذا لم يُمرّر المستخدم أى قيم للمُعاملات: يتم أخذ كل القيم الافتراضية و إسنادها للمُعاملات المقترنة بها، و يصح استخدام خاصية التحميل مع الإجراءات التى لمُعاملاتها قيم افتراضية (و إن كان هذا قد يؤدى إلى ارتباك المبرمج إذا لم يُحسن التعامل معه جيداً).

و لكن يجب مراعاة الأمور التالية:

- أن يكون المُعامل/ات المهمل/ة فى تمرير القيمة فى آخر قائمة المعاملات، فلا يُمكن إهمال تمرير المُعامل الأول فى قائمة المُعاملات و تمرير الثانى لأن هذا سيجعل المترجم يضع القيمة المُمرّرة للمُعامل الثانى فى المُعامل الأول و هو ما سيتسبب فى خطأ نحويّ إذا تضاربت أنواع المعاملات، بالإضافة إلى كونه غلطاً منطقياً فى البرنامج.
- إذا ما أسمىنا توقيع الإجراء بعد استبعاد المُعاملات التى لها قيمٌ ابتدائيةٌ بـ (التوقيع الأدنى): فيجب ألا يتشابه التوقيع الأدنى لكل إجراءٍ مع التوقيع الأدنى لأي إجراءٍ آخر له نفس الاسم. مثال علي التوقيع الأدنى:

الإجراء:

إجراء الإجراء (رقم س نص ص = "بسم الله"):

\\ الكود /

توقيعه الكامل: رقم نص

توقيعه الأدنى: رقم

و بالنسبة للمُخْرَجَاتِ:

فإذا قام المُبرمج بإسناد قيمةٍ للخروج داخل الإجراء و تم تنفيذ هذا الإسناد في زمن التشغيل فلن يتم استخدام القيمة الافتراضية حينها، أما إذا لم يفعل: فسيتم استخدام القيمة الافتراضية. و لا يُشترط هنا أن تكون المخرجات ذات القيم الافتراضية في آخر القائمة، بل يُمكن أن تكون في أي مكان.

مثال: الإجراء:

إجراء (رقم س منطوق ص) ماعددالحالات (رقم الرقم نص الحرف = "ل" نص المحتوى = "الله أكبر"):
\ جسم الإجراء /

يمكن استخدامه كما يلي:

قيم المتغيرين (الحرف) و (المحتوى)	الإستخدام
الحرف = 'ل' المحتوى = "الله أكبر"	ماعددالحالات (1)
الحرف = 'ق' المحتوى = "الله أكبر"	ماعددالحالات (2 "ق")
الحرف = 'ق' المحتوى = "بسم الله الرحمن الرحيم"	ماعددالحالات (3 "ق" "بسم الله الرحمن الرحيم")

بينما يُسبب الاستخدام التالي خطأً نحويًا:

ماعددالحالات ("بسم الله الرحمن الرحيم")

لأنه تم تمرير متغيرٍ من النوع نص فقط إلى الإجراء (ماعددالحالات) الذي أول معاملٍ له من النوع رقم و بالتالي فإن النوعين لا يتفقان.

كسر تنفيذ أوامر الإجراء:

في أحيانٍ معينةٍ قد نرغب في إيقاف تنفيذ إجراءٍ معينٍ عند نقطةٍ محددة، و في هذه الحالة يمكننا استخدام الأمر **أنهي** () أو الأمر **أتخطي** () لأداء هذه المهمة.

مثال:

إجراء تجربة.أنهي :
لو 2 = 2 :
أنهي (تجربة.أنهي)
أكتب.نص. سطر ("هذا أمر لن يتم تنفيذه")

و لأن أنهي () لا فائدة منطقية له إن وُضع في صلب الإجراء مباشرةً: فقد تم منع هذا بحيث لو فعله أحدهم لكان خطأ، و لا يصح وضعه إلا في داخل جملة (لو) أو ما شابهها.

طلب معالجة غلطٍ من داخل الإجراء:

يُمكن في صلب الإجراء إلقاء غلطٍ exception throwing بالطريقة التي سيتم شرحها فيما بعد عند الحديث عن معالجة الأغلط.

مثال:

إجراء تجربة، معالجة، غلط :
أعالج (غلط)

ما ينفرد به الإجراء الحر و لا يُشاركه فيه الإجراء الثابت:

1. إضافة إجراءٍ إلى قائمة استدعاءات الإجراء الحر في زمن التنفيذ:

استخدام الصياغات التالية:

الصيغة 1:

في اسم الإجراء الحر (التوقيع) أضيف (استدعاء الإجراء)

و يجب مراعاة القواعد التالية:

1. بعد اسم الإجراء الحر يمكن إهمال كتابة الأقواس و التوقيع إن لم تكن هناك تحميلات للإجراء الحر، فإن كانت هناك تحميلات و جب كتابة أنواع المتغيرات بالترتيب الصحيح بدون وضع أسماء لها (و هذا هو المقصود بالتوقيع).

مثال:

في نداء الحالات أضيف (ماعدد الحالات ("ق"))

مثال:

في نداء الحالات (رقم نص) أضيف (ماعدد الحالات ("ق"))

2. بعد اسم الإجراء المُضاف يمكن إهمال كتابة القوسين و المُعامِلات إن كان له نفس توقيع

الإجراء الحر، و ساعتها ستكون قيم المُعامِلات المُمرّرة له هي نفسها قيم المُعامِلات المُمرّرة للإجراء الحر الذي تم ضمه إليه.

3. بخصوص القيم التي تُوضع كمُعَامِلَاتٍ في أمر استدعاء الإجراء: سوف يتم حسابها مرةً واحدةً عند تنفيذ أمر الإضافة، و في كل مرةٍ يتم فيها تنفيذ الإجراء الحر الذي تتم الإضافة له سيتم استدعاء الإجراء المُضاف اعتماداً علي قيمة المُعَامِلَاتِ التي تم حسابها عند الإضافة.

الصيغة 2:

في اسم الإجراء الحر 1 (التوقيع) اسم الإجراء الحر 2 (التوقيع) ... أضيف (استدعاء الإجراء)

و النقاط الثلاث معناها أسماء إجراءاتٍ أُخري لو أردنا كتابة المزيد منها، و هنا يجب مراعاة كل شروط الصياغة 1، كما أنه عند إهمال كتابة توقيع الإجراء المُضاف فيجب أن يكون توقيع كل الإجراءات الحرة واحداً و أن يتفق توقيع الإجراء المُضاف مع ذلك التوقيع.

2. حذف إجراءٍ من قائمة استدعاءات الإجراء الحر في زمن التنفيذ:

استخدام الصياغات التالية:

الصيغة 1:

في اسم الإجراء الحر 1 (التوقيع) ... أحذف. آخر. نداء (اسم الإجراء (التوقيع))

حيث في هذه الصياغة سيتم حذف آخر نداءٍ للإجراء الموجود بين قوسي الأمر **أحذف. آخر. نداء** في الإجراءات الحرة المذكورة، و يجب مراعاة نفس القواعد التي نبهنا إليها في صياغات الإضافة. و إذا لم تكن هناك أي نداءاتٍ للإجراء المُراد حذف نداءاته من الإجراءات الحرة: فلن يتسبب ذلك في أي خطأٍ في زمن التنفيذ.

الصيغة 2:

في اسم الإجراء الحر 1 (التوقيع) ... أحذف. كل. نداء (اسم الإجراء (التوقيع))

حيث في هذه الصياغة سيتم حذف كل نداءٍ للإجراء الموجود بين قوسي الأمر **أحذف. كل. نداء** في الإجراءات الحرة المذكورة، و يجب مراعاة نفس القواعد التي نبهنا إليها في صياغات الإضافة. و هنا أيضاً إذا لم تكن هناك أي نداءاتٍ للإجراء المُراد حذف نداءاته من الإجراءات الحرة: فلن يتسبب ذلك في أي خطأٍ في زمن التنفيذ.

4. الجمع بين الإضافة و الحذف:

يمكن الجمع بين العديد من الإضافات لاستدعاءات إجراءاتٍ إلى الإجراء الحر و/أو الحذف منها في تعبير واحد، كما يلي:

في اسم الإجراء الحر 1 (التوقيع) ... :

أضيف (استدعاء الإجراء)
أحذف. آخر. نداء (استدعاء الإجراء)
....

مثال:

في نداء الحالات:
أضيف (ماعدد الحالات ("م"))
أحذف. كل. نداء (ماعدد الحالات (نص))

5. التفريغ الكامل:

بعد تنفيذ الإجراء الحر باستدعائه لن يتم تلقائياً حذف كل الإجراءات التي أُضيفت إليه في زمن التنفيذ بشكل ذاتي، بل إذا أراد المبرمج حدوث هذا فيجب أن يأمر بذلك صراحةً عن طريق استخدام الأمر (أحذف. الكل) كما يلي:

الصياغة 1:

في اسم. الإجراء الحر (التوقيع) ... أحذف. الكل ()

و هنا أيضاً كما في الإضافة و الحذف يمكن إهمال كتابة التوقيع و الأقواس بعد اسم الإجراء الحر إن لم تكن هناك تحميلات له، فإن كانت هناك تحميلات و جب كتابة توقيعه.

مثال 1:

في نداء الحالات () أحذف. الكل ()

و بفرض وجود تحميل للإجراء (نداء الحالات) نكتب:

في نداء الحالات (رقم نص) أحذف. الكل ()

الصياغة 2:

في اسم. الإجراء الحر (التوقيع) ... :
أحذف. الكل ()

ما ينفرد به الإجراء المُخصَّص و لا تشاركه فيه الأنواع الأخرى:

(لم يتم بناء الإجراء المُخصَّص في الإصدار رقم 1.0 من مُفسِّر اللغة).

الإجراءات المُخصَّصة هي إجراءات يرتبط كل واحدٍ منها بمعالجٍ مُعيَّن، بحيث أنه:

- تكون الصياغة:

إجراء مخصص مستوى. الوصول الإستنساخ التغطية (المُخرجات) اسم. الإجراء (المعاملات):
 يخص نوع. المعالج :
 \ جسم الإجراء /

حيث أن كلمة **يخص** هي كلمة مفتاحية يجب كتابتها لتنبه المترجم إلى أن هذا الإجراء مُخصَّصٌ للمُعالِجِ التالي إسمه. أما نوع المُعالِجِ الذي تتم كتابته بعد كلمة **يخص** فهو عنصرٌ من اللقب القياسي (بيئة) الذي يُعبّر كل عنصرٍ من عناصره عن نوعٍ مُعيّنٍ من المُعالِجات، أو عن عائلةٍ من المُعالِجات مُتوافقةً مع بعضها البعض.

مثال تخيلي:

إجراء مخصص إجراء ما:
 يخص بيئة_إنتل 8086 :
 \ جسم الإجراء /

- داخل كل إجراء مُخصَّصٍ يجب كتابة الأوامر بلغة التجميع **assembly language** الخاصة بالمُعالِجِ الذي تم تخصيص ذلك الإجراء له بصورةٍ صرفة، ولا يُسمح بأوامرٍ عالية المُستوي، ما عدا ما سيلي ذكره. وبالطبع فإن لغات التجميع التي سيتم استخدامها ستكون لغات تجميعٍ عربيةٍ بإذن الله عز و جل.

- في داخل الإجراء المُخصَّص لا يُمكن استعمال أكواد بلغة **إبداع** إلا استدعاءات الإجراءات، و التي تُكتَب بنفس الشكل الذي تُكتَب به في برامج **إبداع**، بشرط ألا يكون هناك إسناد قيم عند الاستدعاء:

مثال تخيلي صحيح:

أضع 3، م 2
 إجراء ما ()
 أجمع م 1، م 2

مثال تخيلي به خطأ:

أضع 3، م 2

س = إجراء ما ()
أجمع م 1، م 2

- إذا كانت هناك تحميلات للإجراء المُخصَّص: فعند الإِستدعاء سيتم تنفيذ الإجراء الذي يكون نوع المُعالج الذي خُصَّص له ذلك الإجراء هو نفس نوع المُعالج الذي يعمل (في حالة المُفسِّر) أو سيعمل (في حالة المُترجم) عليه نظامُ التشغيل،

أي بفرض أن لدينا الإجراءات التالية:

إجراء مخصص إجراء.ما:
يخص بيئة إنتل 8086 :
\ جسم الإجراء /

إجراء مخصص إجراء.ما:
يخص بيئة ARM :
\ جسم الإجراء /

فعند الاستدعاء:

إجراء ما ()

و كان المُعالج الذي يعمل عليه هذا البرنامج هو إنتل 8086 فإنه سيتم تنفيذ الإجراء الأول. ويتضح من المثال السابق أنه يُمكن لمجموعةٍ من الإجراءات المُخصَّصة أن تمتلك نفس الخصائص من: الاسم، و عدد و أنواع و ترتيب المُدخلات و المُخرجات، و مُعاملات الوصول، و باقى الخصائص الخاصة بإجراءاتٍ أخرى ثابتة غير مُخصَّصة، أو مُخصَّصة لمُعالجاتٍ أخرى تختلف عن تلك التى خُصَّصت هي لها؛ حيث أن نوع المُعالج يُعتَبَر في حد ذاته مُعامل تفرقة بين الإجراءات المُحمَّلة.

- عند استدعاء الإجراء المُخصَّص، و في حالة لم يكن هناك تحميلٌ خاصٌ بالمُعالج الذي يعمل عليه نظام التشغيل الحالي: فسيُنتج عن هذا غلطٌ زمن تنفيذٍ مُناسب،

- إذا كان هناك إجراءٌ ثابتٌ أو حرٌّ له تحميلٌ واحدٌ أو أكثر من إجراءاتٍ مُخصَّصة: فعند استدعاء ذلك الإجراء فستكون أولوية التنفيذ للإجراءات المُخصَّصة، أما إذا لم يُوجد إجراءٌ مُخصَّصٌ

خاصَّ بالمُعَالِجِ الْحَالِي فَإِنَّهُ يَتِمُّ تَنْفِيذُ التَّحْمِيلِ الثَّابِتِ أَوْ الْحَرِّ.

أَيُّ بَفْرِضِ أَنْ لَدِينَا الْإِجْرَاءَاتِ التَّالِيَةَ:

<p>إِجْرَاءُ إِجْرَاءٍ. مَا:</p> <p>\\ جِسْمُ الْإِجْرَاءِ /</p> <p>إِجْرَاءٌ مُخَصَّصٌ لِإِجْرَاءٍ. مَا:</p> <p>يَخْصُ بِيئَةَ ARM_ :</p> <p>\\ جِسْمُ الْإِجْرَاءِ /</p>
--

فَعِنْدَ الْاسْتِدْعَاءِ:

<p>إِجْرَاءٍ مَا ()</p>

وَكَانَ الْمُعَالِجُ الَّذِي يَعْمَلُ عَلَيْهِ هَذَا الْبَرْنَامِجُ هُوَ ARM: فَإِنَّهُ سَيَتِمُّ تَنْفِيذُ الْإِجْرَاءِ الثَّانِي؛ لِأَنَّهُ مُخَصَّصٌ لِهَذَا النَّوعِ مِنَ الْمُعَالِجَاتِ، مَعَ أَنَّ الْإِجْرَاءَ الْأَوَّلَ يَصْلِحُ بِالطَّبَعِ لِلتَّنْفِيذِ فِي كُلِّ الْأَحْوَالِ.

- فِي دَاخِلِ الْإِجْرَاءِ الْمُخَصَّصِ إِذَا كَانَ هُنَاكَ اسْتِدْعَاءٌ لِإِجْرَاءٍ آخَرَ فَهُنَا حَالَتَانِ:
 - الْإِجْرَاءُ الْمُسْتَدْعَى هُوَ إِجْرَاءٌ مُخَصَّصٌ:

سَيَتِمُّ التَّعَامُلُ مَعَ الْمُخْرَجَاتِ registers كَأَنَّهَا مُتَغَيِّرَاتٌ مُشْتَرَكَةٌ، أَيُّ أَنَّ قِيَمَهَا قَبْلَ الْإِسْتِدْعَاءِ وَعِنْدَ بَدَايَةِ تَنْفِيذِ الْإِجْرَاءِ الْمُسْتَدْعَى سَتَكُونُ وَاحِدَةً، وَقِيَمَهَا فِي نَهَايَةِ تَنْفِيذِ الْإِجْرَاءِ الْمُسْتَدْعَى وَبَعْدَ انْتِهَاءِ تَنْفِيذِ الْإِسْتِدْعَاءِ وَالْعَوْدَةِ إِلَى الْإِجْرَاءِ الْمُسْتَدْعَى سَتَكُونُ وَاحِدَةً.

- الْإِجْرَاءُ الْمُسْتَدْعَى هُوَ إِجْرَاءٌ مُخَصَّصٌ:

بَعْدَ انْتِهَاءِ تَنْفِيذِ الْإِجْرَاءِ الْمُسْتَدْعَى سَيَتِمُّ إِعَادَةُ قِيَمِ الْمُخْرَجَاتِ إِلَى مَا كَانَتْ عَلَيْهِ قَبْلَ تَنْفِيذِ أَمْرِ الْإِسْتِدْعَاءِ.

- يُمَكِّنُ لِهَذِهِ الْإِجْرَاءَاتِ أَنْ يَكُونَ لَهَا خَرْجٌ أَوْ دَخْلٌ مِنْ أَيِّ نَوْعٍ مِنَ الْأَنْوَاعِ الْبَسِيطَةِ أَوْ الْمُرَكَّبَةِ (أَيُّ: رَقْمٌ وَ/أَوْ نَصٌّ وَ/أَوْ مَنْطِقٌ وَ/أَوْ الْجَدَاوِلُ وَ/أَوْ كَائِنَاتُ الْأَصْنَافِ وَ/أَوْ الْأَلْقَابِ)، وَ لَكِنِ التَّعَامُلُ دَاخِلَ الْكُودِ لَنْ يَكُونَ مُمَكِّنًا إِلَّا مَعَ النَّوَاعِيْنِ الْبَسِيطِيْنَ التَّالِيَيْنِ: رَقْمٌ وَ نَصٌّ. مَعَ مُرَاعَاةِ قَوَاعِدِ التَّعَامُلِ الَّتِي سَيَرِدُ ذِكْرُهَا فِي بَنْدِ (كَيْفِيَّةُ التَّعَامُلِ مَعَ الْمُدْخَلَاتِ وَالْمُخْرَجَاتِ).

كَيْفِيَّةُ التَّعَامُلِ مَعَ الْمُدْخَلَاتِ وَالْمُخْرَجَاتِ:

نظراً لأن الكود سيكون مكتوباً بلغة التجميع فإن التعامل مع المتغيرات (سواءً أكانت معاملاتٍ مُمرّرةً أم مُخرجاتٍ) سيكون باستخدام عناوينها، و يكون استخلاص عنوان المتغير باستخدام الإجراء القياسي مرجع كما يلي:

مرجع (اسم المتغير)

مثال: الأمر الوهمي التالي:

أحمل [مرجع (س)]، م 1

يُفترض أن معناه تحميل محتويات العنوان الذي يُشير إليه عنوان المتغير س (أى قيمة المتغير س نفسه) إلى المُخزّن المُسمّى م 1.

مثال 2: الأمر الوهمي التالي:

أخزن م 2، [مرجع (ص)]

يضع قيمة المتغير ص فى المكان فى الذاكرة الذى يُشير إليه عنوان المتغير ص، أى فى المتغير ص نفسه.

مثال على إجراءٍ مُخصّصٍ كاملٍ: (كالعادة: أوامر لغة التجميع المكتوبة فى هذه الإجراء أوامرٌ وهميةٌ للتمثيل فقط):

إجراء مخصص (رقم ص) إجراء ما (رقم س):
 يخص بيثة_إنتل 8086 :
 أحمل [مرجع (س)]، م 1
 أضع 3، م 2
 أجمع م 1، م 2
 أخزن م 2، [مرجع (ص)]

الأصناف

نبذة بسيطة:

الصَّنْفُ هو مُكوِّنٌ يُمَثِّلُ تجميعاً من: المتغيرات مختلفة الأنواع و/أو الإجراءات و/أو الأصناف الأخرى. حيث أنه يُشبه حاوية للإجراءات بحيث تعمل على المتغيرات الموجودة في داخله بجانبها، وهو بذلك يُشبه الـ classes في لغات الـ C++ و الـ java و الـ C# في الفكرة الأساسية ويختلف في الصياغات و التفصيل، و سيلي شرح كل ذلك بطريقةٍ شاملة.

الصياغة:

صنف مستوى. الوصول الوراثة الاسم يرث صنف 1 صنف 2 صنف 3....:

\ تعبير (ما عدا) إن أردنا استعماله /

\ تعبير (تلقيب) إن أردنا استعماله /

\ تعريف حاويات الصنف (المتغيرات و/أو الثوابت و/أو الجداول و/أو

كائنات الأصناف و/أو كائنات الألقاب /

\ تعريف الألقاب الداخلية و/أو الإجراءات الداخلية و/أو الأصناف الداخلية /

مستويات الوصول:

إذا كان الصنف موجوداً في صنف آخر أو في ملف باقية فتكون لها المعاني التالية:

المستوى	الشرح
عام (الافتراضي)	إذا كان صنفاً في باقية فيمكن الوصول إليه من أي مكان خارج باقته. أما لو كان صنفاً داخل صنف آخر فمعني كونه عاماً هو أنه يمكن الوصول إليه من داخل صنفه الحاوي له و الأصناف التي ترثه.
داخلي	إذا كان صنفاً في باقية: فيمكن الوصول إليه من داخل باقته فقط. أما لو كان صنفاً داخل صنف آخر: فيمكن الوصول إليه من داخل صنفه الحاوي له و الأصناف الوارثة له فقط، أي أنه في هذه الحالة فإن وصف عام يشبه وصف داخلي تماماً في العمل لأنه لا يمكن الوصول لصنف داخل صنف آخر من أي مكان خارج صنفه الحاوي له.
خاص	إذا كان صنفاً في باقية: فلا يمكن الوصول إليه من خارج باقته، أي أنه في هذه الحالة يتساوي و صفي داخلي و خاص.

أما لو كان صنفاً داخل صنفاً آخر: فيمكن الوصول إليه من داخل صنفه الحاوي له فقط.	
--	--

صفات الوراثة:

الصفة	الشرح
سقف	لا يمكن وراثته ويمكن فقط صنع كائناتٍ منه، وفي هذا النوع من الأصناف لا يمكن تعريف إجراءاتٍ و/أو أصنافٍ داخليةٍ مجردة لأنه لن تكون هناك فرصة لبنائها في صنفٍ وارث.
مجرد	لا يمكن صنع نسخةٍ منه بل يُمكن وراثته فقط، و عندها يجب بناء كل إجراءاته المجردة.
بدون صفة (الإفتراسي)	الصنف العادي الذي يمكن وراثته أو استنساخه.

مثال:

صنف عام مجرد حساب: \ جسم الصنف /

صيغ الاستخدام:

الصيغة 1:

اسم.الصنف مستوى.الوصول الاستنساخ ثبات.المرجع اسم.النسخة 1 = اسم.الصنف(المعاملات إن وُجِدَت) اسم.النسخة 2 = اسم.الصنف(المعاملات إن وُجِدَت)

مستويات الوصول:

هي نفسها الخاصة بمتغيرات الأصناف إذا أشهر الكائن في صنفٍ ما.

الاستنساخ:

هي نفسها الخاصة بمتغيرات الأصناف إذا أشهر الكائن في صنفٍ ما.

ثبات المرجع:

الصفة	الشرح
ثابت	قيمة المؤشر من النوع الثابت وهو مخصصٌ للنسخة التي يُربط بها في

أول مرة (سواء أكان الربط فى جملة الإشهار أم بعدها)، أى لا يمكن تغييرها و جعل المؤشر يشير إلى نسخةٍ أخرى من نفس الصنف. و فعل ذلك يؤدى إلى إعلان المترجم حدوث خطأ نحوى، و لو تم استخدام هذه الكلمة فيجب أن يتم حجز مكان في الذاكرة للكائن في نفس الجملة التعريفية.	
قيمته من النوع المتغير أى يمكن تغييرها بصورةٍ عادية، و حينها يكون بالإمكان أن نحجز له مكان في الذاكرة في أى وقت	بدون صفة (الافتراضي)

مثال 1:

حساب عام الحساب.الأول = حساب()

و هنا نلاحظ أنه لو كتبنا:

حساب عام الحساب.الأول = حساب()
الحساب.الأول = لاشئ

فإننا نكون بذلك قد أنهينا وجود الكائن فى الذاكرة و أصبح المتغير (الحساب.الأول) لا يشير إلى أى كائن.

الصيغة 2:

اسم.الصنف مستوى.الوصول الاستنساخ ثبات.المرجع اسم.النسخة
اسم.النسخة = اسم.الصنف(المعاملات.إن.وجدت)

عدم استخدام = اسم.الصنف(المعاملات.إن.وجدت) يعنى أننا أشهرنا متغيراً من نوع الصنف و لكننا لم نحجز له مكاناً فى الذاكرة، ثم قمنا فى السطر التالى بحجز ذلك المكان له.

مثال:

حساب عام الحساب.الأول
الحساب.الأول = حساب()

تعبير فى:

يُستخدم تعبير (فى) حينما نستخدم متغير/اتٍ و/أو نستدعى إجراء/اتٍ فى كائن ما لتوفير الوقت اللازم لكتابة الاستدعاءات منسوبةً إلى اسم كائن الصنف فى كل مرة، و لإبقاء الكود مختصراً قدر الإمكان مع المحافظة على الوضوح، و صيغته كما يلى:

الصيغة 1:

فى اسم.كائن:

\ إسناد قيم لعنا صر الكائن /

متغير 1 = قيمة

متغير 2 = قيمة

مثال:

في حساب 1:

القيمة = 20000

الريح = 10

القيمة. الكلية = احسب()

الصيغة 2:

في كائن 1 كائن 2:

\ إسناد قيم لعنا صر الكائنات /

متغير 1 = قيمة

متغير 2 = قيمة

مثال:

في حساب 1 حساب 2:

القيمة = 20000

الريح = 10

القيمة. الكلية = احسب()

استخدام مكونات الكائن من صنف:

نستخدم العلامة _ لكي نستخدم أى مُكوِّنٍ من مُكوِّنات الأصناف مثل المتغيرات و الإجراءات.

الصياغة:

الصنف _ اسم. المكون

مثال 1:

الحساب. الأول _ قيمة. الحساب = 50000

الوراثة في الأصناف:

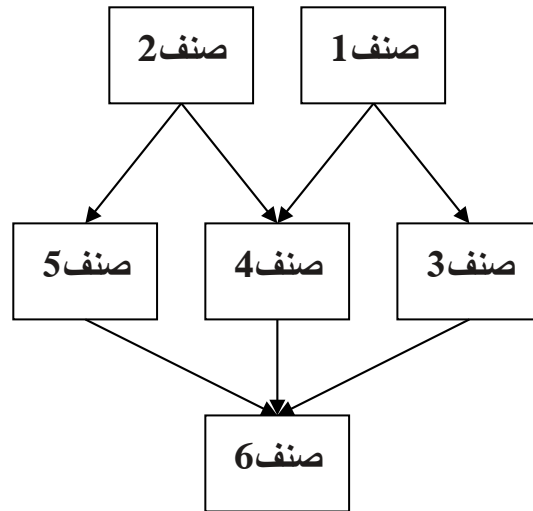
1- تدعم (الوراثة المتعددة multiple inheritance) و تُحل مسألة تداخل أسماء المتغيرات و/أو الكائنات و/أو الثوابت و/أو الإجراءات و/أو الأصناف الداخلية و/أو الألقاب الداخلية (إذا حدث و تشابهت أسماء مكوّنٍ أو أكثر في صنفين موروثين و/أو صنفٍ موروثٍ مع الصنف الوارث) عن طريق:

1- جملة إعادة تسمية للأسماء المتشابهة و هي كما يلي:

صنف صنف.1 يرث صنف 2 صنف 3:
 تلقيب:
 صنف 3_اسم.المتغير.القديم = الاسم.الجديد 1
 صنف 2_اسم.الإجراء.القديم(أنواع.المعاملات.لو.هناك تحميل) =
 الاسم.الجديد 2

لو كان ذلك المكوّن إجراءً فيجب كتابة أنواع المُعاملات لو كان له تحميل. مع العلم بأن الإجراءات الموروثة في كل صنفٍ من الأصناف التي تغير اسم مكوناتٍ فيها سيتم تلقائياً تغيير كودها لتستخدم الأسماء الجديدة بدلاً من الأسماء القديمة.

أما مسألة كون أكثر من صنفٍ موروثٍ لها أصنافٌ مشتركة يرثون منها جميعاً مثل الشكل التالي:



حيث أن الصنف (صنف 6) يرث كلاً من الأصناف (صنف 3) و (صنف 4) و (صنف 5)، لكن المشكلة أن الصنفين (صنف 3) و (صنف 4) يشتركان في الوراثة من (صنف 1) و أن الصنفين (صنف 4) و (صنف 5) يشتركان في الوراثة من (صنف 2).

فلها أكثر من احتمالٍ و أكثر من حل:

1. الاحتمال الأول:

الأصناف الموروثة (صنف 3 و صنف 4 و صنف 5) لم تُعدّل في الأصناف الموروثة المشتركة بينها (صنف 1 و صنف 2). و ليست بها أية إجراءات جديدة زائدة (أضيفت إليها زيادة على الوراثة) متشابهة الأسماء، و هنا لا تكون هناك أى مشكلة لأنه سيتم وراثة المتغيرات و الإجراءات و الأصناف التي تُوجد في الأصناف المشتركة (صنف 1 و صنف 2) كما هي في الصنف الجديد الخاص بنا (صنف 6). و سيتم كذلك وراثة الإجراءات الزائدة في (صنف 3 و صنف 4 و صنف 5) كما هي بدون أى مشاكل لأنها مختلفة الأسماء.

2. الاحتمال الثاني:

هناك تعديل قد تم في بعض أو كل الإجراءات الخاصة بالأصناف المشتركة، عن طريق قيام أحد الأصناف الوارثة لها بتغطية مُكوّن من هذه المُكوّنات أو أكثر (أى التغيير في بنائها كما سيوضح في كلامنا عن الإجراءات)، أو هناك مُكوّنات متشابهة الأنواع و الأسماء في الأصناف الموروثة (صنف 3 و صنف 4 و صنف 5)، و بالتالى تظهر مشكلة أن هناك مُكوّنات تتفق فى الاسم و لكنها تختلف فى البناء و من ثم لا نستطيع الاستغناء عن أحدها بالآخر.

و لهذا الأمر حلّوه فى إيداع و هى كما يلي:

1) يدويا باستخدام تعبير الاستثناء من الوراثة:

حيث يقوم المبرمج باستثناء كل المُكوّنات المتشابهة التى تُسبب المشكلة ما عدا مُكوّن واحدٍ سيقي لتتم وراثته هو فقط فى الصنف الجديد و من ثم تتلاشى المشكلة من الأصل.

و بالنظر إلى المثال السابق يمكننا أن نرى أن الحل سيكون كما يلي:

صنف صنف 6 يرث صنف 3 صنف 4:

ما عدا:

صنف 3_إجراء 1

و لن يكون هناك أى تضاربٍ فى أسماء الإجراءات التى تمت وراثتها، و هكذا نرى أن الإجراءات التى ستُضم نتيجة الوراثة إلى (صنف 6) هى:

إجراء 1) <---- من صنف 4

إجراء 2) <---- من صنف 1

إجراء 3) <---- من صنف 1

و يجب كتابة توقيع الإجراءات إذا ما كان هناك تحميل له.

مثال:

صنف صنف 6 يرث صنف 3 صنف 4:

ما عدا:

صنف 3_إجراء 1(رقم نص)

(2) يدوياً باستخدام التلقيب:

حيث يقوم المبرمج بنفسه بإعطاء كل المكونات المتشابهة أسماءً جديدةً ما عدا واحدٍ فقط، و عندئذٍ سيقوم المترجم تلقائياً بضمه إلى الصنف الجديد لأنه وحيد الاسم و ليس هناك مكوناتٌ أخرى تُشاركه ذلك الاسم بعد عملية التلقيب أى أن المشكلة لن تظل موجودة، و ستقوم المكونات التي لم تُغطى بنداؤه هو فى كودها لأنه الوحيد الذى له الاسم القديم.

مع الأخذ فى الحسبان أنه عند تلقيب جميع المكونات المشتركة و تغيير أسمائها و عدم ترك أى منها باسمها القديم، فإن المكونات الأخرى التي لم تُغطى فى الأصناف الموروثة ستظل فى هذه الحالة أيضاً تنادى المكونات المشتركة بأسمائها القديمة، و من ثم سيُخرج لنا المترجم رسالة خطأ و لن يكمل عمله، و ذلك لأنه لن يكون هناك مكونٌ بنفس الاسم القديم الذى تستخدمه المكونات غير المُغطاة.

و بالتطبيق على مثالنا نرى أن الحل سيكون كما يلى:

صنف 6 يرث صنف 3 صنف 4:

تلقيب:

صنف 3_إجراء 1 = "إجراء 3.1"

فتكون الإجراءات التي تَضم إلى صنف 6 كما يلى:

إجراء 1 () <---- من صنف 4

إجراء 2 () <---- من صنف 1

إجراء 3 () <---- من صنف 1

إجراء 3.1 () <---- من صنف 3

و عندما يقوم الإجراءات ان (إجراء 2 و إجراء 3) باستخدام (إجراء 1) سيجدان أمامهما الإجراء المضموم من (صنف 4) فقط و ليس شيئاً آخر.

ملحظ هام: فى هذه الحالة فإن الإجراءات التي سيتم تلقيبها سوف تستدعى نفسها (لو كانت تستدعي نفسها) بالإسم الجديد.

3- إذا ما تم تعريف إجراء في الصنف الوارث له بنفس اسم إجراء آخر في الصنف الموروث فإن الإجراء الجديد سِيُعْطَى القديم و يُخْفِيهِ عن المستخدم للصنف الوارث، و يظهر له الإجراء الجديد فقط. و لا يُحْتَاج إلى كتابة كلمات خاصة تُعبر عن قابلية الإجراء في الصنف لتغطيته في الأصناف التي ترث صنفه، بل هناك كلمة مفتاحية واحدة لجعل الإجراء غير قابلٍ للتغطية و هي كلمة (سقف) كما سيلي في الجزء الخاص بالإجراءات.

4- لا يُسْمَح بوراثه أصناف لها أسماء متشابهة.

الكلمة المفتاحية (هذا):

هي كلمة مفتاحية تُستخدم للإشارة إلى الصنف الذي تُكْتَب داخله، و نلجأ إليها عندما نريد استخدام مُكوِّن من مُكوِّنات الصنف الحالي و هناك مكوِّن آخر يحمل نفس الاسم في نفس المساحة التكويدية code block، أى أن هناك مشكلة تضارب في الأسماء ستحدث. و مثال على ذلك أن يكون هناك في المُعَامِلَات المُمرَّرة لإجراء متغيرات لها أسماء تُشبه أسماء المتغيرات التي تُوجد في الصنف الحالي، و بالتالي عند استخدام اسم المتغير يتم استخدام المُعَامِل لا متغير الصنف رغم أنه ربما يكون هو المقصود بالاستخدام، و تُحل هذه المشكلة باستخدام الكلمة هذا كما في المثال التالي:

صنف صنف.1:

نص المحتوى

إجراء غير.النص.إلي (نص المحتوى):

هذا_المحتوى = المحتوى

المُشِيدَات:

تُمَاثِل الـ constructors في اللغات الأخرى، و هي إجراءات عادية يتم استدعاؤها عندما يتم تعريف كائن جديد من الصنف المُحتوى على المُشِيد و حجز مكان له في الذاكرة، و في حالة لغة إبداع يتم هذا عندما يُكْتَب:

= اسم.الصنف ()

بعد اسم الكائن في جملة تعريف الكائن الجديد، و قد سبق شرح هذا. و المُشِيدَات في إبداع هي إجراءات عادية تماماً و لكن يُشترط فيها شرط واحد فقط هو أن يكون اسمها هو نفس اسم الصنف الحاوي لها.

و للمُشِيدَات نوعان:

1- الافتراضية: أى الذى لا نمرّر له أى مُعَامِلَات.

مثال:

إجراء عام صنف 1:
هذا_الحساب = 2000

-2

ذو المعاملات:

مثال:

إجراء عام صنف 1 (رقم الحساب):
هذا_الحساب = 2000

- و استدعاء مُشِيدَاتِ الأَصْنَافِ الموروثَةِ عند استدعاء مُشِيدَاتِ الأَصْنَافِ الوارثَةِ يكون طبقاً للقواعد التالية:
- في حالة وجود مُشِيدَاتٍ تُمَاثِلِ مُشِيدِ الصنف الوارث في توقيع المُدخَلَاتِ: يقوم المُترجم تلقائياً باستدعائها في كل مرة يُستدعى فيها مُشِيدِ الصنف الوارث، و تكون قيم المُعامَلاتِ المُمرَّرة لكل مُشِيدٍ منها هي قيم المُعامَلاتِ المُمرَّرة لمُشِيدِ الصنف الوارث.
 - في حالة عدم وجود و لو مُشِيدٍ واحدٍ في صنفٍ من الأَصْنَافِ الموروثَةِ يحمل نفس توقيع المدخلات الذي لمُشِيدِ الصنف الوارث: فإن المُترجم يترك أمر استدعاء مُشِيدَاتِ الأَصْنَافِ الموروثَةِ للمبرمج تماماً لتكون العملية يدويةً بالكامل.

خُيُوطُ التَّنْفِيزِ `executing threads`

لم يتم بناؤها في الإصدار 1.0 من مُفسِّر اللغة).
 في إبداع تُعتبر خيوط التنفيذ المتعددة عبارة عن إجراءات يتم تنفيذ كل منها بالتوازي مع الأخرى، و
 تُعامل كحالة وسيطة بين كائن الصنف و الإجراءات العادية، كما سيوضح من الشرح التالي.

إنشاء الخيط:

أنشيء ("اسم.الخيط.الجديد" اسم.الإجراء (التوقيع إن احتجناه))

مثال:

أنشيء ("الخيط.الأول" إجراء.ما)

الإستخدام:

لبدء عمل الخيط يتم التعامل معه كأنه إجراء عادي تماماً، كما يلي:

اسم.الخيط(المعاملات)

مثال:

أنشيء ("أوجد.الرقم.الأكبر" الرقم.الأكبر)
 رقم أكبر.رقم = أوجد.الرقم.الأكبر(9898 542)

التحكم في عمل الخيط:

يُمكن التحكم في عمل الخيط عن طريق تطبيق إجراءات قياسي عليه (بما يُشبه التعامل مع كائنات الأصناف).
 كما يلي:

إسم.الخيط_الإجراء.القياسي(المعاملات)

مثال:

الخيط.الأول_أوقف(1 2)

الإجراءات القياسية التي يُمكن تطبيقها على الإجراءات الخيط:

ملحظ هام: أسماء الإجراءات القياسية التالية مُجرّد خطوطٍ عريضةٍ لما سيُوجد عند بناء خيوط التنفيذ في
 مُفسِّر اللغة، أي أنها قد تتغير و/أو تزيد و/أو تنقص.

• أوقف()

لإيقاف تنفيذ خيط التنفيذ الحالي نهائياً

• أوقف.الكل()

- لإيقاف تنفيذ كل خيوط التنفيذ الحالية نهائياً
أثبط (فترة زمنية)
- لإيقاف تنفيذ خيط التنفيذ لفترة معينة
أنشط ()
- لإعادة خيط تنفيذ إلى العمل بعد التثبيت
أثبط. الكل (فترة زمنية)
- لإيقاف تنفيذ كل خيوط التنفيذ ما عدا صاحب النداء لفترة معينة
أنشط. الكل ()
- لإعادة كل خيوط التنفيذ المثبطة إلى العمل

الجملة الشرطية (لو)

الصيغة:

لو \ الشرط /:
\ الأوامر /
أما لو \ شرط آخر /:
\ الأوامر /
غيره:
\ الأوامر /

و الأقواس اختيارية

مثال:

لو س = ص :
أكتب. نص. سطر ("س يساوي ص")
أما لو س في من 1 إلي 12 :
أكتب. نص. سطر ("س في المجموعة من 1 إلي 12")
أما لو س في {5 4 3 2 1} :
أكتب. نص. سطر ("س في المجموعة {5 4 3 2 1}")
أما لو س في جدول. الأرقام :
أكتب. نص. سطر ("س في جدول. الأرقام")
أما لو س < 12 و س > 20 :
أكتب. نص. سطر ("س بين الـ 12 و الـ 20")
غيره:
أكتب. نص. سطر ("س خارج الاحتمالات الممنوحة")

حيث نري أن التعبيرات المنطقية المسموح بها كما يلي:

• التعبيرات العادية مثل:

○ س = ص

○ س < 12 و س > 20

- تعبير في: وهو دائماً علي الشكل التالي:

< حاوية > في < تجميع من الحاويات من نفس النوع >

وله الحالات التالية:

- في جدول: أن يكون التجميع عبارة عن إسم جدولٍ أو بُعدٍ في جدول، مثل:

لو س في جدول.الأرقام

- تعبير (من - إلي - بقيمة -): ولا يصح استخدامه إلا مع القيم الرقمية، مثل:

لو س في من 2 إلي 100 بقيمة 2

- قيم مباشرة، مثل:

لو س في { 6 5 4 3 2 1 }

- المزج بين كل ما سبق، مثل:

لو س < 12 و س > 20 أو س في { 5 4 3 2 1 }

الجداول

الصياغة 1:

التعريف الكامل بدون تحديد المحتويات:

النوع {عدد الأبعاد} مستوى-الوصول الإستنساخ ثبات. المرجع الاسم 1 الاسم 2

الصياغة 2:

التعريف مع تحديد المحتويات:

النوع {عدد الأبعاد} مستوى-الوصول الإستنساخ الاسم = |
 اسم. الصف. الأول = {عنا صر الصف الأول}.
 اسم. الصف. الثاني = {عنا صر الصف الثاني}.
 اسم. الصف. الثالث = اسم. جدول. ذي. صف. واحد.
 {

مع ملاحظة أنه في كل الصيغ الفاتحة فإن:

1. أسماء الصفوف يمكن إهمالها مع علامة التساوي التي تليها.

2. لو أعطينا الجدول النوع (حر) فإن كل صفٍ يُمكن أن يكون له نوعٌ خاصٌ به، أما لو كان للجدول نوعٌ آخرٌ غير (حر) فلا يمكن لأي صفٍ أن يحوى غير ذلك النوع، ولا يمكن كتابة نوعٍ أمام اسم الصف.

مستويات الوصول:

إذا استُخدمت مع الجداول التي توجد في الأ صناف يكون لها نفس معاني مستويات و صول المتغيرات التي توجد في الأ صناف.

الاستنساخ:

إذا استُخدمت مع الجداول التي توجد في الأ صناف يكون لها نفس معاني صفات استنساخ المتغيرات التي توجد في الأ صناف.

ثبات المرجع:

تماماً ككائنات الأ صناف.

مثال 1:

رقم {1} عام مشترك الأعداد

مثال 2:

رقم {2} عام الأعداد = {الصف.الأول = {4 6 2} الصف.الثاني = {33 16 10}}

مثال 3:

رقم {2} عام الأعداد
الأعداد = {{33 16 10} {4 6 2}}

الجدول ذات الأبعاد مختلفة الطول:

للمبرمج إمكانية إظهار جداول ذات أبعاد مختلفة الأطوال.

مثال:

رقم {2} عام مشترك الأعداد = {{14 8 16} {5 46 8 4 6 2}}
الأعداد {4 1} = 20
أكتب.رقم. سطر(الأعداد {4 1})

حجز أماكن فارغة لعنا صر الجدول:

إذا أردنا أن نحجز أماكن فارغة لعنا صر الجدول يمكننا أن نفعل ذلك كما يلي:

اسم.الجدول = جديد(عد دعنا صر البعد الأول عد دعنا صر البعد الثاني)

مثال:

رقم {2} الأرقام = جديد(20 10)

و سيتم إعطاء تلك العنا صر قيمها الافتراضية حسب نوعها بمجرد حجز مكان لها.

الإستخدام:

و للوصول إلى عنصر في جدول فهناك طريقتان لفعل ذلك هما:

1- بالمُمَيِّزَات indices:

نكتب الأرقام التي تعبر عن ترتيب العنصر في الجدول بين أقواسٍ من النوع $\{ \}$ ، و لا يُفصل بين رقم إحدائى كل بُعدٍ و الآخر بأي فاصلٍ، كما يلي:

اسم الجدول {إحدائى.البعدالأول إحدائى.البعدالثانى}

مع الأخذ في الحسبان أن العد يبدأ في الجداول في لغة إبداع بالميميز 1.

مثال:

رقم {2} الأعداد = { { 7 8 5 } { 4 6 2 } }
الأعداد { 3 1 } = 15

و هنا قمنا بتغيير قيمة العنصر ذى الصف الأول و العمود الثالث و الذى كان يحمل القيمة (4) إلى القيمة (15).

و للمبرمج إمكانية تحديد جزءٍ معينٍ للعمل عليه من الجدول باستخدام تعبير (من - إلي -)، كالتالى:

مثال:

الأعداد {من 1 إلي 10 1} = الأعداد الأخرى {5 من 10 إلي 20}

و في هذا المثال وضعنا الأعداد الموجودة في الصف الخامس من الجدول المُسمَّى (الأعداد الأخرى) بدايةً من العنصر رقم 10 إلى العنصر رقم 20 في العمود الأول من الجدول المُسمَّى (الأعداد) بدايةً من الصف رقم 1 إلى الصف رقم 10.

2- طريقةً أخرى للاستخدام:

يمكننا التعامل مع الصف كأنه جدولٌ كاملٌ (ذو بُعدٍ واحدٍ). كما يلي:

اسم الجدول_ اسم الصف { }

مع ملاحظة أنه لديه ميزةً أخرى، هي أنه لو أردنا الحصول على قيمةٍ في الصف المسمى (ب) و التي تُناظر القيمة (ق) في الصف (أ) فإنه يمكننا القيام بالآتي:

الجدول_ أ_ب (ق)

و هو ما يعني (قيمة الصف ب المُوازية للعنصر ذى القيمة ق في البعد أ).

معرفة طول الجدول أو بُعد من أبعاده:

يُمكن ذلك عن طريق استدعاء الإجراء طول () كما يلي:

اسم الجدول_طول ()

معرفة عدد الأبعاد لجدول أو لبُعد من أبعاده:

يُمكن ذلك عن طريق استدعاء الإجراء عدد.الأبعاد () كما يلي:

اسم الجدول_عدد.الأبعاد ()

المُتسلسلات العددية:

التعبير

{ من قيمة 1 إلى قيمة 2 }

يعيد جدولاً أحادي يبدأ بالعنصر (قيمة 1) و ينتهي بالعنصر (قيمة 2) و يزيد كل عنصر عن ما يسبقه بمقدار (1)، و لو أردنا تغيير مقدار الفرق بين كل عنصر و الذي يليه نستخدم الصياغة التي في المثال التالي:

{ من قيمة 1 إلى قيمة 2 بقيمة قيمة 3 }

و هذا التعبير يعيد جدولاً أحادياً يبدأ بالعنصر (قيمة 1) و ينتهي بالقيمة (قيمة 2) و كل قيمة جديدة تزيد عن القيمة القديمة بمقدار (3).

و يمكن استخدام هذا التعبير لملء الجداول كالتالي:

مثال 1:

رقم {2} الأعداد = { من 4 إلى 6 } { { 7 8 5 } }

مثال 2:

رقم {2} الأعداد = { من 4 إلى 8 بقيمة 2 } { { 7 8 5 } }

الجُمَلُ التِّكرارية

جملة بينما الشرطية:

تكرار الاسم:
بينما \ شرط / :
\ الكود /

مثال:

تكرار التكرار. الأول:
بينما س > 15 :
أكتب. نص. سطر(س)
س = س + 1

جملة بينما للتتابعات:

تكرار \ الاسم / :
بينما \ اسم. الحاوية / في \ اسم. التجميعية / :
\ الكود /

و التجميعات هنا هي نفسها التي ذكرناها في شرح جملة لو الشرطية.

مثال 1:

تكرار التكرار. الأول:
بينما س في الأعداد:
أكتب. نص. سطر(س)

مثال 2:

تكرار التكرار. الثاني:
بينما س في {من 10 إلى 20} :
أكتب. نص. سطر(س)

إجراءات قياسية ذات صلة:

1- أتخطي (اسم التكرار):

و تُستَخدم في الحالات التي نريد من المترجم أن يُهمل تنفيذ الأوامر التي تليها في صُلب الجملة التكرارية و يقوم بتنفيذ الدورة التكرارية التالية. و تُكافئ كلمة **continue** في عائلة الـ C.

مثال: للبرنامج التالي:

رقم {1} ص = { 5 4 3 2 1 }

التكرار التكرار. الأول:

بينما س في ص :

لو س = 3 :

أتخطي (التكرار. الأول)

أكتب. نص. سطر(س)

سيكون الخرج كما يلي:

1

2

4

5

و إذا لم نكتب اسم حلقة تكرارية داخل قوسي الأمر **أتخطي** فسيقتض المتبرجم أنه اسم آخر حلقة تكرارية من اللواتي كتب داخلهن.

مثال: للبرنامج التالي:

رقم {1} ص = { 5 4 3 2 1 }

التكرار التكرار. الأول:

بينما س في ص :

لو س = 3 :

أتخطي ()

أكتب. رقم. سطر(س)

سيكون الخرج كما يلي:

1
2
4
5

2- أنهي (اسم التكرار):

و تكافئ كلمة **break** في عائلة ال-C، حيث تُستَخدم لكسر تنفيذ الجملة التكرارية، و تُعامل كأمر أتخطي تماماً.

و مثالٌ على ذلك البرنامج التالي:

رقم {1} ص = { 5 3 0 4 2 1 }
التكرار التكرار.الأول:
بينما س في ص :
لو س = 0 :
أنهي (التكرار.الأول)
أكتب، نص. سطر (1 ÷ س)

سيكون الخرج كما يلي:

1
0.5
0.25

و هناك أمورٌ هامةٌ بخصوص استعمال هذين الإجرائين:

- لا يمكن استعمالهما في المسار العام للبرنامج و لافي داخل الإجراءات مباشرةً، بل يمكن استعمالهما فقط داخل تعبيرَي **لو** و **بينما**.
 - عند كتابة أي من هذين الإجرائين في كتلة أوامرٍ معينة فإنه لا يمكن ان يأتي بعدهما أي أمرٍ آخر في ذات كتلة الأوامر.
- مثال:

بينما س في ص :
لو س = 0 :
أنهي (التكرار.الأول)
س = 5 هذا خطأ

الألقاب

نبذة بسيطة:

هي مكوّناتٌ تمكّننا من ضم مجموعةٍ من الكلمات على أساس أنها تمثّل قيماً ضمن نوعٍ مُعيّنٍ من أنواع البيانات هو اللقب، فكما نقول أن النوع رقم يضم القيم

5 4 3 2 1

و يُمكننا إشهار متغيرٍ من النوع رقم له القيمة 3 فإنه يُمكننا باستخدام الألقاب القول بأن النوع (يوم) يضم القيم (سبت أحد الإثنين) حيث يمكن إشهار متغيرٍ من النوع (يوم) له القيمة (سبت) على سبيل المثال، واستخدام ذلك في الكود. و بالتالي هي تماثل ال-Enumeration في عائلة ال-C و التي يُترجمها كل من قرأت لهم بالكلمة (تعدادات) و التي لم أستسغها و فضّلتُ عليها كلمة الألقاب الواضحة المعنى و الأصغر حجماً.

الصياغة:

يمكن إشهار لقبٍ جديدٍ باستخدام الصيغ التالية:

الصيغة 1:

لقب مستوى.الوصول الاسم :
العنصرالأول العنصررقم.

مستويات الوصول:

حينما تُستخدم مع الألقاب التي في الباقيات أو في الأصناف يكون لها المعاني التالية:

المستوى	الشرح
عام (الافتراضي)	يمكن الوصول إليه من أى مكان
داخلي	إن كان في صنفٍ فيمكن الوصول إليه من داخل صنفه و الأصناف الوارثة له فقط، أما إن كان في ملفٍ باقيةٍ فيمكن الوصول له من داخل باقته فقط.
خاص	إن كان داخل صنفٍ فيمكن الوصول إليه من داخل صنفه فقط، أما إن كان داخل باقيةٍ فيمكن الوصول له من باقته فقط.

مثال:

لقب خاص يوم:

السبت الأحد الإثنين الثلاثاء الأربعاء الخميس الجمعة

صِيغُ الاستخدام:

صياغة:

اسم. اللقب مستوى. الوصول الإستنساخ الثبات اسم. الكائن = اسم. اللقب_ اسم. العنصر

مستويات الوصول:

إذا استُخدمت مع كائنات الألقاب التي تعرف في الأصناف: فكائن اللقب له نفس مستويات وصول متغيرات الأصناف و نفس معانيها.

الاستنساخ:

إذا استُخدمت مع كائنات الألقاب التي تعرف في الأصناف: فكائن اللقب له نفس صفات متغيرات الأصناف و نفس معانيها.

الثبات:

تماماً كموا صفات المتغيرات.

مثال:

يوم عام مشترك أحد الأيام = يوم_الخميس يوم. الإجازة = يوم_الجمعة

تحويل قيمة كائن اللقب إلي نص:

قد نحتاج لهذا لو أردنا أن نكتب قيمة كائن اللقب علي الشاشة مثلاً، و يُمكننا هذا عن طريق كتابة:

اسم. اللقب_إلي. نص (القيمة)

أمثلة:

يوم_إلي. نص (أحد الأيام)

يوم_إلي. نص (يوم_الجمعة)

مُعَالَجَةُ الْأَغْلَاطِ

نُبذةٌ بسيطةٌ:

في إبداع يُطلق الاسم **غلط** على ما يُسميه الآخرون (استثناء) و كلاهما ترجمةٌ لكلمة **exception**، و قد استُخدمت كلمة **غلط** لأن الاستخدام الغالب على هذا النوع من الأدوات يكون لمعالجة ما يعتبره المبرمج خطأً منطقيًا أثناء زمن التشغيل و هي الأمور التي يَصِحُّ لغويًا إطلاق و صف الأغلط عليها؛ لأن لفظة **غلط** تُطلق على الشيء الذي يُوَضَع في غير محله، و بذلك تُعبِّر عن المعنى جيدًا و في نفس الوقت هي أخف و أصغر من لفظة استثناء التي تزيد حروفها عن حروف **غلط** و تتناثر أكثر على لوحة المفاتيح.

تعريف غلطٍ جديد:

حينما نريد أن نعرِّف أمرًا ما على أساس أنه غلطةٌ منطقيةٌ، يمكننا أن نفعل ذلك بالخطوات التالية:

1) تعريف صنفٍ جديدٍ يرث من الصنف (**غلط**).

مثال:

صنف غلط 1 يرث غلط:

نص السبب

إجراء إلى نص:

أكتب نص.. سطر ("**هذا غلط من النوع غلط 1**")

إجراء تنبيه:

أكتب نص.. سطر ("**هذا تنبيه على وقوع الغلط في البرنامج بسبب أن "** +

السبب)

2) وضع جملة إعلان وجود غلطٍ في الأوامر المُراد معالجة الغلط فيها، و نستخدم الكلمة

المفتاحية (**أعالج**) يليها اسم نسخةٍ من الغلط المراد معالجته.

و يمكننا ضبط متغيرات الكائن قبل أمر المعالجة بحيث يستفيد منها المُعالج عند القيام

بالمُعالجة كما في المثال التالي.

مثال:

لو س < ص :

غلط 1 غ = غلط 1()

غ_السبب = "المتغير س < المتغير ص"
أعالج(غ)

3) نُحِيطُ جُمْلَةَ الْإِعْلَانِ عَنِ الْغَلْطِ بِتَعْبِيرٍ مَعَالِجَةَ الْأَغْلَاطِ، وَهُوَ تَعْبِيرٌ (أَحَاوَلُ) التَّالِي
شرحہ.

مثال:

أحاول:
لو س < ص :
غلط 1 الغلط = غلط 1 ()
الغلط_السبب = "المتغير س < المتغير ص"
أعالج(الغلط)
ع = ص - س
معالجة غلط 1 غ :
غ_تنبيه ()

صياغة معالجة الأغلاط:

أحاول :
\ الأوامر المراد تنفيذها عادة /
معالجة نوع.الغلط الاسم:
\ الأوامر التي ستنفذ في حالة حدوث الغلط المكتوب في الأعلى /
معالجة نوع.الغلط الاسم:
\ الأوامر التي ستنفذ في حالة حدوث الغلط المكتوب في الأعلى /
دوماً:
\ الأوامر التي ستنفذ لو حدث غلط أم لم يحدث /

مثال:

أحاول :
أكتب.نص. سطر ("تجربة")

معالجة غلط. دخل. خرج س :
س_إلى.نص ()

في حالة رفع غلطٍ ما ووجود أكثر من فرع من أفرع (معالجة) فسيتم اختيار الفرع الأكثر تناسباً مع نوع الصنف الخاص بالغلط تلقائياً، تماماً كما يحدث عند نداء إجراء مُحتمَل.

مثال: البرنامج التالي:

أحاول:

تجربة.رد.الفعل ()

معالجة غلط غ :

أكتب.نص. سطر ("معالجة غلط غ")

معالجة غلط 1 غ :

أكتب.نص. سطر ("معالجة غلط 1 غ")

دوما:

أكتب.نص. سطر ("تنفيذ دوما")

إجراء تجربة.رد.الفعل:

غلط 1 غلطة = غلط 1 ()

غلطة_الرسالة = "الغلط دا حصل بمزاجي"

أعالج (غلطة صحيح)

صنف غلط 1 يرث غلط :

إجراء غلط 1 :

لا شيء

له الخرج التالي:

معالجة غلط 1 غ

تنفيذ دوما

الأغلاط واجبة المعالجة:

في بعض الأحيان قد نحتاج إلي جعل معالجة الأغلاط التي قد تحدث عند تنفيذ إجراءٍ ما أمراً لازماً للمبرمج الذي سوف يستدعي ذلك الإجراء، و لذلك يمكننا تمرير القيمة المنطقية **صحيح** لأمر المعالجة **أعالج** حتي تؤدي هذا الدور.

و مثالٌ علي ذلك البرنامج السابق الذي كُتِب فيه:

أعالج(غلطة **صحيح**)

لكي يفرض علي من يستدعي الإجراء (تجربة.رد.الفعل) أن يحيط أمر الاستدعاء بتعبير **أحاول** فإن لم يفعل فإنه يحصل علي خطأٍ نحويٍ من المترجم.

الأغلاط القياسية في إبداع:

في كل لغة برمجة تدعم معالجة الأغلاط يكون فيها مجموعةً من الاغلاط القياسية التي من الممكن للمبرمج الإعتماد عليها في أكوادها، مثل (غلط القسمة علي الصفر) أو (غلط محاولة الوصول إلي عنصرٍ في المصفوفة باستخدام مميز خارج نطاق حدودها)، أما في الإصدار الأولي من **إبداع** فلم يتم بعد تصميم تلك الأغلاط القياسية أو بناؤها. علي أنه من المُخَطَّط له أن يتم تصميمها الواحدة تلو الأخرى و بناؤها بإذن **الله** تعالي في الإصدارات القادمة.

إِستخدام الأكواد الخارجية في إبداع

الأكواد الخارجية في إبداع تنقسم إلى الفئات التالية:

- ملفات الباقات العادية التي تحتوي على أكواد أصناف:

ويمكن استخدامها كما قلنا من قبل عن طريق أمر **أضم** ثم التعامل مع الباقات الداخلية و/أو الأصناف و/أو الألقاب التي بداخلها بصورة عادية جداً.
- ملفات مكتبات الربط الديناميكي (مثل DLL في الويندوز و الـSO في القنو/لينوكس) و التي هي عبارة عن أكواد مكتوبة **بإبداع** ثم تم ترجمتها لكي تكون مكتبة متقلة قابلة للاستخدام في زمن التشغيل:

و هذه تُعامل كما تُعامل ملفات الباقات العادية تماماً، أي نستخدم معها الأمر **أضم**، ثم تُستخدَم مكوناتها الداخلية كالعادة.

(لم يتم بناؤها في الإصدار 1.0 من مُفسّر اللغة).
- واجهات أنظمة التشغيل المُختلفة API التي ربما يود المُبرمج الوصول إليها في برامجه: و هذه لم يتم الإستقرار علي الموقف منها في **إبداع** (في هذه الإصدار من المُفسّر علي الأقل).

رُوحُ الْإِبْدَاعِ

فى هذا الجزء سوف أناقش أهم القضايا البرمجية التى ركزتُ عليها بشدة أثناء تصميم **إبداع**، و ربما يلمس القارئ بنفسه هذه القضايا و يستنتجها جميعها لوضوح الإهتمام بها فى الشروحات السابقة للغة، و سر تركيزى على هذه القضايا أكثر من غيرها يعكس اقتناعى الشديد بأنها ذات أهمية قُصوى فى العمل البرمجى عامةً و فى تصميم لغات البرمجة خاصة، و ذلك حيث أن لغة البرمجة هى الأداة التى بها يتم بناء باقى مُخرجات العمل البرمجى فبالتالى هى المكون الأكثر عكسا للقضايا الهامة برمجيا.

و فى هذه القضايا قد أخالف الكثيرين من أبرز أهل الإختصاص فى آرائهم و قناعاتهم، و ليس الأمر أنى من مُحببى التفرد و الغرابة اللذين يفعلون ما يفعلون من أجل لفت الأنظار فقط، بل إننى فعلتُ ما فعلتُ عن قناعةٍ حقيقيةٍ و لدها فى تأملٍ كبيرٍ فى العلم البرمجى و ما أحب له أن يصير إليه، و كذا ما آخذهُ على الواقع الحالى من ما أخذٍ من الأفضل أن تُزال، و ما أراه فيه من حسناتٍ حَبِذا لو تُقَوَّى و تُزاد.

و مرَدُّ الأمر فى النهاية إلى القناعات الذهنية الذاتية، و ليس هذا المجال من المجالات التى يُمكننا فيها القول بالخطأ المَحْض و الصواب المَحْض، بل الأمر هنا هو الإجتهد البشرى الذى يُصيب صاحبه و يُخطئ، و ليس لأحدٍ تحجيم آراء أحدٍ أو منعها أو تقييد الحرية فى التَّقْوُّه بها و نشرها ما دام الأمر أولاً و آخرًا جهدا بشريا خالصا قابلا للنقد و الدحض.

و قد كان من المُمكِن أن أُفرد لهذا الجزء ورقةً علميةً خاصةً به أتوسع فيها في تبيان وجهات نظري في مسائل تتعلق بتصميم لغات البرمجة، إلا أنني رأيت أنه لا بد من إدراجه هنا بعد ذكر مكوّنات لغة إبداع و قبل ذكر ما ليس من مكوّناتها؛ حتى يُبيّن و يُوضّح الأساس الذي بُنيت عليه كل ما سبق و كل ما سيأتي. ففي هذا الجزء سيتضح السبب وراء إدراج كل مكوّن من مكوّنات اللغة، و سيتضح كذلك القنّاعة التي دفعتني إلى رفض ضم أي مكوّن رَفُضتُ ضَمّه إلى اللغة، و بالتأكيد سيتضح للقارئ ما غَمُضَ عليه في شرح أسباب رفض المكوّنات في الجزء التالي بسرعةٍ كبيرةٍ لفهمه لنظرتي للغات البرمجة جيداً.

و القضايا التي سأركّز عليها هنا هي على الترتيب:

- (1) التخصُّص،
- (2) الوراثة،
- (3) التهجين،
- (4) الشمول،
- (5) البساطة و الاستقرار،
- (6) الأمن،
- (7) الألفة.

التخصُّص:

قضية التخصُّص أحصرها هنا في المسألة التي تُناقش ما يجب على المُبرمج البانى للمُترجم القيام به و ما يجب أن يُترك للمُبرمج مُستخدم اللغة لكي يفعله، و يُمكننى أن أُعبّر عن رأى فى هذه القضية بجُملة بسيطة و هى " دع الإدارة للمُترجم و حلّ المشاكل الأساسية للمُستخدم ". و حينما أقول الإدارة فإننى أقصد مُجمل الأنشطة التى تختص بإدارة الذاكرة، و التى أُفصلها فى البندين التاليين:

- تحديد المكان الذى ستُوضَع به البيانات فى الذاكرة، أى عملية حجز الأماكن لها،
- تحديد الوقت المُناسب الذى سيتم فيه إرجاع الأماكن التى حُجزت لمُتغيّرات فى الذاكرة (ثم لم تُعد هناك حاجة لها) إلى حوزة نظام التشغيل مرةً أخرى لإعادة استغلالها.

فهاتان النقطتان تُمثّلان التعامل الكامل مع الذاكرة من وجهة النظر البرمجية للمُستخدم لنظام التشغيل، أما الإدارة التى يقوم بها نظام التشغيل فهى عملية أكثر تعقيداً و تشعباً من هذه بكثير، و إن كانت تتركز عليها بشكلٍ أساسى.

و نظرتى إلى هاتين النقطتين على أنهما من نصيب المُبرمج البانى للمُترجم أو المُفسّر يُوضّح نظرتى إلى بنية إدارة الذاكرة و توزيع المسؤوليات و فقها، حيث أُقسّمها إلى ثلاثة أقسام هى:

(1) القسم الأدنى:

و يختص به نظام التشغيل، و هى الأمور التى تتعلق بكيفية تنظيم العمليات التى يجرى تنفيذها على الحاسوب فى الذاكرة، و كيفية إحلال البرامج الجديدة التى يرغب المُستخدم للحاسوب فى العمل عليها محل تلك التى قيد التنفيذ و لكنها مُتوقّفة عن العمل وقتياً.

و كذا كيفية التنسيق بين التصور للذاكرة على أنها منطقة مُتصلة من الأماكن التخزينية مصفوفة فوق بعضها البعض، و بين التصور الواقعى لها على أساس البناء العتادى لها (أى طريقة بنائها كما صمّمتها شركة التصنيع لرقاقة الذاكرة).

(2) القسم الأوسط:

و يختص به المُترجم و المُفسّر، و يختص بحجز مكانٍ فى الذاكرة للمُتغيّرات و مُكوّنات البرنامج المُختلفة، و فى النهاية إرجاع أى مساحة فى الذاكرة لم يعد البرنامج فى حاجة إليها إلى نظام التشغيل مرةً أخرى؛ لتقليص المفقود من إمكانيات جهاز الحاسب إلى أدنى المراتب.

(3) القسم الأعلى:

و هو مُجَرَّد الإِسْتِخْدَامِ بِدُونِ التَّقْيِيدِ بِكَيْفِيَّةٍ مُعَيَّنَةٍ، وَ يَخْتَصُّ بِهِ بِطَبِيعَةِ الْحَالِ الْمُبْرَمَجِ الْعَادِي الَّذِي يَسْتُخْدِمُ لُغَةَ الْبَرْمَجَةِ.

وَأَنَا أَعْتَبِرُ أَنَّ أَى مَسَاسٍ بِهَذَا التَّقْسِيمِ سَيُحَوِّلُ الْعَمَلِيَّةَ الْبَرْمَجِيَّةَ إِلَى حَلْبَةِ "جودو"، حَيْثُ تَتَدَاخَلُ أَقْدَامُ وَ أذْرَعُ الْمَتَصَارِعِينَ بِشَكْلِ لَا يَدْرِي مَعَهُ أَحَدٌ لِمَنْ هَذَا الذَّرَاعُ وَ لِمَنْ هَذِهِ الْقَدَمُ! . وَ بِالطَّبَعِ فَإِنَّ هَذِهِ الْحَالِ لَا تُرْضِي مَنْ يُدَافِعُ عَنِ الْحَقِّ الطَّبَعِيِّ لِلْمُبْرَمَجِ الْعَادِي فِي الْحَصُولِ عَلَى الْحَرِيَّةِ الْقُصْوَى وَ الرَّاحَةِ الْكَامِلَةِ فِي التَّرْكِيزِ عَلَى حَلِّ مُشْكَلَتِهِ الَّتِي يَرِغِبُ فِي حَلِّهَا، بَدَلًا مِنْ تَفْرِيعِ جِزْءٍ مِنْ وَقْتِهِ وَ جَهْدِهِ لِعَمَلِيَّةٍ مِنَ الْوَاضِحِ أَنَّهُ لَا نَاقَةَ لَهُ فِيهَا وَ لَا جَمَلَ.

ثُمَّ إِنِّي أَتَسَاءَلُ عَنِ الْوِظِيْفَةِ الَّتِي نَدَّخَرُهَا لِلْمُتَرْجِمِ إِنْ كَانَتْ إِدَارَةُ الْذَاكِرَةِ مِنْ نَصِيبِ الْمُبْرَمَجِ الْعَادِي، فَهَلْ سَتَكُونُ مَهْمَةً الْمُتَرْجِمَاتِ الرَّئِيسَةِ الَّتِي تُرَكِّزُ عَلَيْهَا أَغْلَبَ التَّرْكِيزِ هِيَ عَمَلِيَّةُ التَّحْوِيلِ مِنَ النَّصِّ الْبَرْمَجِيِّ عَالِي الْمَسْتَوَى إِلَى الْأَصْفَارِ وَ الْآحَادِ فَقَطْ وَ نَتْرِكُ الْمَهْمَاتِ الْوَعْرَةَ الْآخِرَى لِلْمُبْرَمَجِ لِتُتَوَلَّأَهَا بِنَفْسِهِ؟

إِنِّي أَظُنُّ أَنَّهُ مِنَ الْإِلْزَامِ حَقًّا أَنْ يُرَكِّزَ مُطَوِّرُوا اللُّغَاتِ الْبَرْمَجِيَّةِ عَلَى التَّمُودِجِ النَّظِيفِ مِنْهَا، وَ أَنْ يَضَعُوا نَصْبَ أَعْيُنِهِمْ تَطْوِيرَ مُتَرْجِمَاتٍ كَفُوَّةٍ تُعَيِّنُ الْمُبْرَمَجَ الْمُسْتُخْدِمَ عَلَى التَّرْكِيزِ عَلَى عَمَلِهِ الْأَسَاسِيِّ، وَ فِي نَفْسِ الْوَقْتِ تُطَبِّقُ أَفْضَلَ الْخَوَارِزِمَاتِ الَّتِي تُعْطِي الْبَرْنَامَجَ كِفَاءَةَ الْإِسْتِغْلَالِ الْأَفْضَلِ لِمَوَارِدِ الْجِهَازِ.

إِنْ جَعَلَ إِدَارَةُ الْذَاكِرَةِ مِنْ نَصِيبِ الْمُبْرَمَجِ النَّهَائِيِّ لَا يَكْتَفِي فَقَطْ بِبَلْبَلَةِ فِكْرِ الْمُبْرَمَجِ وَ شَغْلِهِ عَنِ الْأَمْرِ الْأَسَاسِيِّ لَهُ (وَ هُوَ بَرْمَجَةُ التَّطْبِيقِ الْمَرْغُوبِ فِيهِ)، بَلْ يَفْرِضُ عَلَيْهِ أَنْ يَعْلَمَ أَشْيَاءًا لَا تَهْمُهُ فِي قَلِيلٍ أَوْ كَثِيرٍ فِي الْأَدْوَاتِ الْبَرْمَجِيَّةِ الَّتِي يَسْتُخْدِمُهَا.

وَ سَأَضْرِبُ لَكُمْ مَثَلًا شَخْصِيًّا عَلَيَّ هَذَا الْأَمْرَ:

فَأَنَا عَنِ نَفْسِي كُنْتُ أَسْتُخْدِمُ مَكْتَبَةَ الـ **OpenCV** الْحَرَّةَ الْمَصْدَرِ لِأَنِّي كُنْتُ أَعْمَلُ فِي مَرْكَزِ أبحاثٍ فِي مَشْرُوعٍ يَتَعَلَّقُ فِي شِقِّهِ الْأَوَّلِ بِمُعَالَجَةِ الصُّورِ الرَّقْمِيَّةِ. وَ كُنْتُ أَسْتُخْدِمُ لُغَةَ الـ **C++** الَّتِي تَجْعَلُنِي أَعَانِي الْأَمْرَيْنِ؛ فَالْبَرْنَامَجِ الَّذِي يَحْتَاجُ مِنِّي إِلَى يَوْمٍ لِكِتَابَةِ خَوَارِزِمِهِ وَ ضَبْطِهِ مَنْطِقِيًّا يَحْتَاجُ عَلَى الْأَقْلِ لِيَوْمٍ آخَرَ لِكِتَابَةِ الْكُودِ نَفْسَهُ!، مَعَ أَنَّهُ مِنَ الْمُفْتَرَضِ أَنَّ مَرْحَلَةَ التَّكْوِيدِ هِيَ أَبْسَطُ الْمَرَاكِلِ فِي الْبَرْمَجَةِ، بَلْ وَ يَحْتَاجُ إِلَى أَكْثَرِ مِنْ يَوْمٍ آخَرَ لِلتَّنْقِيحِ وَ التَّخْلُصِ مِنَ الْأَغْلَاطِ الَّتِي تَظْهَرُ نَتِيجَةً لِلْقِيُودِ الَّتِي تَضَعُهَا اللُّغَةُ الْمُسْتُخْدَمَةُ عَلَى كَاهِلِي (أَنَا الْبَائِسُ الْمَسْكِينُ).

وَ مِثَالٌ لِدَلِكِ: الْبَرْنَامَجِ التَّالِي الَّذِي فِيهِ غَلْطَةٌ مَنْطِقِيَّةٌ تَظْهَرُ قَبْلَ انْتِهَاءِ عَمَلِ الْبَرْنَامَجِ:

```

// step1: Load codebook binary file
Network net("D:\\codebook archive\\gray_codebook\\gray_codebook.bin", false);
// getting input from camera
CvCapture * camera= cvCreateCameraCapture(0);
// taking the current shot of the camera in an IplImage structure to work on
IplImage * colored_frame = cvQueryFrame(camera);
// making a new window to show the image in it
cvNamedWindow("camera view");
// keeping working on the frames taken from the camera untill the user wants some thing
while(true)
{
    // get the new frame
    colored_frame = cvQueryFrame(camera);
    |
    |
    |
    |
    |
    |

cvDestroyWindow("camera view");
cvReleaseCapture(&camera);
cvReleaseImage(&colored_frame);

```

ففى هذا البرنامج أقوم بالقراءة من الكاميرا الرقمية المُوَصَّلة بالحاسوب و أُسجِّل ما أقرأ فى كائن يُمَثِّل لقطة كاميرا، بعد ذلك أقوم بأخذ اللقطة الجديدة من ذلك الملف إلي كائن يُمَثِّل صورة، و من ثم أقوم بعرض هذه الصورة فى نافذة على الشاشة، و بعد أن أنهى العمل عليها أقوم بإغلاق النافذة و تحرير الذاكرة المُخَصَّصة لملف اللقطة و الصورة.

فما هى تلك الغلطة المنطقية التى تظهر؟!!

اتضح لى فى النهاية أننى كتبتُ الأمر الخاص بتحرير جزء الذاكرة المحجوز للصورة الجديدة من الكاميرا بعد أن حررتُ جزء الذاكرة الخاص بملف لقطة الكاميرا،

فماذا فى هذا؟!!

كانت المُشكلة أن الـ **OpenCV** حينما عرّفتُ صورةً جديدةً علي أنها مأخوذة من ملف اللقطة (المقروء من الكاميرا) لم تنسخ الصورة من ملف اللقطة إلى مكانٍ جديدٍ فى الذاكرة لتعمل عليها، بل جعلت الكائن من النوع (صورة) يُشير إلى الصورة و هى فى مكانها الأول فى الذاكرة الخاصة بملف لقطة الكاميرا.

إلى هنا و الأمور جيدة؛ فهذا يُفيد فى الحصول على أداءٍ فائقٍ عند التنفيذ، لكن المُشكلة أن اللغة تطلبُ منى تحرير الذاكرة التى أستخدمها بنفسى، و حينما فعلتُ لم أنتبه إلى كيفية تعامل المكتبة مع الذاكرة: فحررتُ اللقطة ثم عدتُ و حررتُ الصورة التى تم تحريرها أصلاً من ضمن لقطة الكاميرا، أى أننى بهذا أكون قد حررتُ جزءاً من الذاكرة أصبح غير محجوز أصلاً!!

و السبب هو التواطؤ غير المقبول بين المكتبة و اللغة، و لو أن اللغة كانت مُتعاونةً لكانت هذه ميزةً فى مكتبة الـ **OpenCV** و ما كنتُ لأشغل بالى بكيفية التحسين الذى وضعه مُصنِّموها و مُبرمجوها ليجعلوها

تُعالج الصور بشكل أسرع، و كنتُ سأستفيد منه بدون أن أشغل ذهني بما فعلوا بتاتا، و لكن استخدام الـ C++ جعل الأمر كابوساً بالفعل و فرَضَ عَلَيَّ أن أعلم كيف يُديرون الذاكرة حتى لا أقع في أخطاءٍ منطقيةٍ نتيجةً لمضادة الكود الذي أكتبه للكود الذي يكتبونه هم !. أى أن مُبرمج الـ C++ مُعرَّضٌ في أي لحظة لأن تُصبح خاصية التجريد abstraction ليس لها أي معنى معه !

فإذا كان الأمر هنا هيناً و يُمكن التعايش معه فإنه سيُصبح مأساةً حقيقيةً إذا ما نظرنا إلى المشاريع البرمجية الضخمة التي تُبرمج باستخدام الـ C++. مثل أسطح المكتب الرسومية و أنوية أنظمة التشغيل و غيرها من المشاريع العملاقة، و التي يتم فيها بطبيعة الحال التعامل مع الذاكرة دوماً و في كل منْحَى من المناحي بشكل في غاية التعقيد بسبب استخدام لغة الـ C أو الـ C++ في التكويد. و سيكون عَلَيَّ أن أعاني الأمرين لو أردتُ المشاركة في مشروع كهذا لمجرد فهم كيفية سير الأمور؛ حتى لا أسير في إتجاه مُعاكس فيحدث ما لا تُحمد عُقباه !

أى أنني بدلاً من أفضى الوقت في التفكير في كيفية تطوير المشروع و تحسينه و إضافة الأفضل إليه: سأفضى أغلب الوقت أتعلم أشياءً هي من صميم مكوّناتٍ أخرى من المشروع، و لكنها تتعلق بالذاكرة التي عَلَيَّ أن أديرها حينما أتعامل مع ذلك المُكوّن أو غيره. و لو أنني أهملتُ هذه المسألة فسأدفع الثمن غالباً بالحصول على ذاكرةٍ كبيرةٍ مُهدّرة، و هو الأمر الذي سيؤدّي إلى جعل فريق العمل يركل مؤخرتي و يطردني بلا رجعةٍ منه؛ نظراً لأن الأنظمة التي يبنوها لابد من أن تكون في غاية الكفاءة في استخدام موارد النظام و لا يُسمح فيها بإهدار أي جزءٍ من الموارد على الإطلاق.

و من الغرب أن الكثيرين من المُبرمجين مُقتنعون اقتناعاً تاماً بأن وُضِعَ إدارة الذاكرة بين يدي المُبرمج النهائي يُعطى البرامج المُنتجة قوةً و كفاءةً عاليتين، و أصبح هذا القول بدوره من التابوهات المُحرّمة في عالم البرمجة، و أصبح الإقتراب منه يُعرّض المُشكّك للسخرية اللاذعة على أساس أنه لا يعلم أبسط القواعد في عالم البرمجة الاحترافية. رغم أن دراسة (The Measured Cost of Conservative Garbage Collection by B. Zorn) التي أُجريت للمقارنة بين كفاءة البرامج التي تُسلّم إدارة الذاكرة إلى المُبرمج النهائي في لغة الـ C و البرامج التي أُسلّم مُبرمجوها أمر إدارة الذاكرة و جمع المُهمَل منها إلى مُجمّع نفاياتٍ جيدٍ: أثبتت أن البرامج الثانية كانت أفضل من ناحية الأداء، على العكس تماماً من القتاعات التي رَسَخها كثيرٌ من المُبرمجين القدامى في أذهان الكل !

و دعونا نفترض أن لكلا النوعين من البرامج نفس الأداء و مُعدّل استغلال الذاكرة. بل فلنفترض أن إدارة الذاكرة اليدوية تُعطي نتائجاً أفضل: فماذا عن الأغلاط التي تملأ البرنامج و تُحيل حياة المُبرمجين إلي جحيم لا يُطاق، و تحوّل البرنامج إلي أرضٍ مَلأى بالألغام المُعرّضة للانفجار في أي لحظة !؟
ماذا عن الوقت الضائع في التنقيح و البحث عن مواضع الأغلاط و كيفية حلها !؟
ماذا عن تكلفة كل هذا !؟

ما ذا عن الكود المُتَضَخِّم بلا معني، بينما أكواد البرامج التي تستعمل مُجَمَّع نفاياتٍ أصغر و أوضح؟!!

ما يُمكنني أن أقوله بكل ثقة أن قناعة تَفُوق الإدارة اليدوية للذاكرة علي الإدارة الآلية ليست إلا تجلياً إضافياً للإنعزال المعنوي عن التِقْنِيَّات الحديثة في عالم البرمجة لدى أجيالٍ من المُبرمجين القُدَامَى و المُحدَثين، تماماً كرفض استعمال الواجِهات الرسومية و بيئات التطوير المُتكامِلة و لغات البرمجة الحديثة، و الإصرار على إستخدام سطر الأوامر في كل الأمور.

و لو أننا التمسنا العُذر لِقُدَامَى المُبرمجين في ذلك (على أساس أنهم نشأوا على سطر الأوامر و تَعَوَّدوا عليه، و كذلك أصبحوا خُبراء لا يُشَق لهم غبارٌ في البرمجة باستخدام اللغات التي استخدموها منذ صِغَرهم، و تكيّفوا مع عيوبها قبل مُميزاتها و أصبح في غاية الصعوبة بالنسبة لهم البدء من الصفر مع لغةٍ جديدة)، أقول: لو التمسنا لهم العذر بسبب ما فات ذِكره لما استطعنا التماس العُذر للأجيال الجديدة من المُبرمجين الذين نشأوا و تربوا على التِقْنِيَّات الأكثر حَدَاثةً من تلك التي اعتاد عليها قُدَامَى المُبرمجين (بئس بالضبُط) و لكنهم استخدموا تلك الأقدم على الرغم من ذلك؟!!

فالسبب الوحيد الذي أراه دَفَعهم إلى هذا هو: الجانب النفسي الذي جعلهم يتصورون أنه مادام القُدَامَى مُصْرُفِين على استخدام التِقْنِيَّات و الأدوات القديمة: فهي الأفضل بلا جدال، و استخدامها دليلٌ على الإحتراف و التمكن، أما التِقْنِيَّات الجديدة فلا يستخدمها إلا المُرقَّهون من المُبرمجين الصغار الذين لم يَصِلوا إلى مرتبة الإحتراف بعد و مازال الطريق أمامهم طويلاً!!

و أنا بطبعي أكره التعصب لتقنيةٍ على حساب تقنيةٍ أخرى لمُجرَد الألفة مع التقنية الأولى و عدم اعتياد الثانية، أو لأن استخدام الأولى يُوحى بالخبرة العميقة في البرمجة و القدرة على أداء الأعمال الغاية في الصعوبة ككبار المُبرمجين مِمَّن يَسْتخدمون التقنية الأولى دوماً، و يُعجبني دائماً المُبرمج الذي يتعامل مع التِقْنِيَّات المُختلفة على أساس الإستفادة القُصوى من كل واحدةٍ منها و الإبتعاد قَدْر الإمكان عن مساوئها و عيوبها. و هذا هو قول العقل السليم الذي يبحث عن الأفضل في كل شئ.

فنجِد المُبرمج العاقل يستخدم سطر الأوامر في أمورٍ مُعَيَّنَةٍ و يستخدم الواجهة الرسومية في أمورٍ أخرى، و يُبرمج بنفسه البرامج التي يحتاجها في أمرٍ ثالث، و يستخدم برامج الآخرين في أمورٍ أخرى. و في كل مجالٍ من المجالات يحاول اختيار اللغة البرمجية الأفضل (إن استطاع تعديد اللغات التي يستخدمها) بدون تعصبٍ لتلك اللغة على حساب تلك.

الوراثة:

حينما يتحدث أي شخص عن مفهوم الوراثة فإنه يقصد نقطة: صُنِعَ صِنْفٌ مِنَ الْأَصْنَافِ، ثم وراثته مُحتوياته في صِنْفٍ آخَرَ و التغيير فيها بالإضافة و/أو التعديل. أما مفهوم الوراثة عندي فهو أمرٌ يتجاوز ذلك إلى معنىٍ أوسع و أشمل؛ فيحتوي على أي طريقةٍ لإعادة استخدام الأكواد في البرامج بأي شكلٍ من الأشكال بشرط أن يضمن تأثير الكود الوارث بالتغييرات التي تطرأ بعد الوراثة على الكود الموروث، و بالطبع فإن هذا يستثنى طريقة النسخ و اللصق التقليدية؛ لأنه فيها تنقطع العلاقة بين الوارث و الموروث تماماً بمجرد انتهاء العملية.

لذلك فمفهوم الوراثة عندي يتضمن الأمور التالية:

- **الوراثة العادية** بين صِنْفٍ مِنَ الْأَصْنَافِ و صِنْفٍ آخَرَ، وهو التعريف النمطي الذي يُعطيه الكل للوراثة في البرمجة الكائنية المُنْحَى OOP.

- **تعريف كائني أو نسخة من صنف**؛ فلو فكرنا فيها جيداً لوجدنا أن الكائنات التي نُعَلِن عنها ما هي إلا إعادة استخدام لكود الصنف بدون الحاجة إلى كتابته مرةً أخرى لكل كائني كما يحدث في منهجيتي البرمجة الوظيفية **functional programming** و الإجرائية **procedural programming**، و كذلك فإن أي تغيير يطرأ على كود الصنف سيكون مطبقاً على كافة النسخ المنتجة منه. و هذا يتحقق شرطاً الوراثة عندي على هذه العملية.

- **استخدام المكتبات البرمجية ذاته**. و هذا الأمر قد يدفع إلى الظن بأن الوراثة عندي ما هي إلا مُسَمَّى آخر لعملية إعادة الاستخدام، و لكنه ظنٌ خاطئٌ تماماً؛ ففي عملية إعادة الاستخدام يكون الشرط الرئيس لكي يُمكنك أن تقول أن ما حدث هو إعادة استخدام هو أن يقوم المُبرمج باستخدام كودٍ برمجي مرةً أخرى فقط، أي أنه لا يُشترط لكي تكون العملية ضمن نطاق إعادة الاستخدام أن تكون التغييرات الجديدة في الأصل (و التي تمت بعد تنفيذ عملية إعادة الاستخدام) ذات أثر على الفروع المُستخدمة للأصل البرمجي، في حين أن هذا شرطٌ أساسيٌّ عندي لكي أُطلق على العملية لقب الوراثة.

و لو نظرنا إلى عملية النسخ و اللصق التقليدية لوجدنا أنها خير مثالٍ يُوَضِّح الفارق عندي بين الوراثة و إعادة الاستخدام؛ لأنها يَصْدُق عليها و صف إعادة الاستخدام بكل قوة، في حين أنه لا ينطبق عليها مفهوم الوراثة عندي و لا يُمكنني أن أعدها لوناً من ألوان التوراث لعدم تطبيق التغييرات المُحدثة إلا على الأصل المنسوخ فقط.

و قد أفضتُ في شرح الفارق بين نظرتي للوراثة و بين النظرة الحالية لها لكي أستطيع بعد ذلك أن أتحدث بحرية عن مدى أهمية الوراثة عندي. و ذلك بدون خوف الإلتباس على القارئ لهذه الأسطر. و الحق أن للوراثة عندي أهمية في الحدود القصوى لها؛ فأنا أعتقد اعتقاداً جازماً أن أي كود مهندس جيداً قد يكون قابلاً لإعادة الاستخدام التوارثي بطريقة تمكّنا من حل المشاكل الجديدة بطرقٍ أسرع و أفضل بكثير.

و جديرٌ بالذكر أن الوراثة تدخل في أغلب المواضيع الهامة التالية إن لم يكن فيها كلها، و ذلك لأن لها من الفوائد الكثير، و يُمكنني أن أعد هنا طرفاً من هذه الفوائد فيما يلي:

• سرعة الإنتاج:

و القول المنتشر الذي يصف هذه النقطة جيداً هو "عدم إعادة اختراع العجلة" (علي الرغم من إساءة تفسير البعض لهذا المبدأ)، حيث لن يكون على المبرمج كتابة الآلاف من الأسطر في كل مرة يحتاجها في عمله؛ بل يكتبها مرةً واحدةً فقط ثم يقوم باستدعائها كل مرةً بمنتهى السرعة. مما يعنى التركيز على المُشكلة الحقيقية و عدم إهدار الجهد في أمرٍ فعلٍ من قبل و هو ما يؤدي إلى الإنتاجية الكبيرة في النهاية.

• الأمان:

الأكواد التي ستستخدم في الوراثة ستكون قد اختبرت جيداً قبل وضعها في الخدمة التوارثية. مما يعنى أن المبرمج المُستخدم لهذه الأكواد الموروثة يستطيع الإطمئنان إلى كون الكود الموروث ذو أمانٍ عالٍ، فلا يحتاج إلى إعادة اختباره مرةً أخرى، و ذلك على العكس من الأكواد الجديدة التي يكتبها إن لم يستخدم الوراثة؛ حيث سيكون عليه التأكد من أمن تلك الأكواد في كل مرة.

• توفير التكلفة:

عند إنتاج البرمجيات بصورة تجارية يكون من أهم الأمور: إنتاج التطبيق المُراد بأقل التكاليف المُمكنة، و قد قام علم هندسة البرمجيات من الأصل ليصل إلى هذا الهدف من ضمن الأهداف الرئيسة التي قام لأجلها.

• توارث الكفاءة:

حينما يتم اكتشاف خوارزم جديد أكثر كفاءةً للقيام بالعملية التي كان يُستخدم فيها خوارزمٌ أقل كفاءةً؛ يكون إحلال الجديد محل القديم مكسباً كبيراً؛ ليس فقط للتطبيقات التي ستستخدم الخوارزم بعد الإحلال، بل حتى للتطبيقات التي تستخدم الخوارزم من أيامه الأولى

قبل التجديد، أى أن التغييرات الجيدة لن تجد طريقها فقط للمستخدمين الجدد بل سَتَم توارثها هى الأخرى (بالطبع قد يحتاج هذا إلى إعادة ترجمة `recompile` الأكواد التي تَستخدِم الخوارزم مَرَّةً أُخرى).

• التَّجَرُّدُ الْأَقْصَى وَبَسَاطَةُ حَلِّ الْمَشَاكِلِ:

حينما تتوافر تحت يد المُبرمج أدواتٌ كثيرةٌ جاهزةٌ للاستخدام: سيكون الأمر الوحيد الذى سيركِّز عليه جهده و تفكيره هو حل المشكلة التى يُريد استخدام البرمجة لحلها، وهو الأمر الذى سيجعل البرمجة أكثر سهولةً بكثير (بل و أكثر إمتاعاً بكثير).

التَّهْجِينُ:

يَتَفَاخِرُ كَثِيرٌ مِنْ مُصَمِّمِي لُغَاتِ الْبَرْمَجَةِ الْمُخْتَلِفَةِ وَالْمُسْتَعْدِمِينَ الْمُتَعَصِّبِينَ لَهَا الْيَوْمَ بِأَنَّهَا لُغَاتٌ كَاتِنِيَّةٌ أَوْ وِظَائِنِيَّةٌ أَوْ إِجْرَائِيَّةٌ صَرَفَةً، وَ ذَلِكَ مِنْ مُنْطَلَقِ أَنَّ الْخَيْرَ كُلَّ الْخَيْرِ فِي ذَلِكَ النَّمَطِ الْبَرْمَجِي الَّذِي تَخَصَّصَتْ فِيهِ تِلْكَ اللُّغَاتُ وَ لَا خَيْرَ فِيهَا عَادَاهُ، وَ هُوَ الْقَوْلُ الَّذِي أُوْدُ أَنْ أُنْوَهُ هُنَا إِلَيَّ خَطُّهُ الْفَادِحُ الْوَاضِحُ.

نَرَى مِنْ يَتَعَصَّبُ لِلـ **java** يَقُولُ أَنَّ الْبَرْمَجَةَ الْكَاتِنِيَّةَ هِيَ الْأَفْضَلُ، وَ هِيَ النَّوْعُ الْأَقْوَى مِنَ الْأَنْمَاطِ الْبَرْمَجِيَّةِ الَّتِي وَصَلَتْ إِلَيْهَا الْفِكْرُ الْبَرْمَجِي الْبَشْرِي، وَ بِالتَّالِي كَانَ مِنَ الرَّائِعِ أَنْ تَكُونَ الـ **java** تَامَّةً الْكَاتِنِيَّةَ **pure oop**، وَ هُنَاكَ مِنْ يُصِرُّ عَلَى أَنَّ الْبَرْمَجَةَ الْوِظَائِنِيَّةَ أَوْ الْإِجْرَائِيَّةَ هِيَ الْأَفْضَلُ وَ الْأَقْدَرُ عَلَى الْقِيَامِ بِكُلِّ الْمَهَامِ بِسَهُولَةٍ، وَ بِالتَّالِي تَكُونُ الْبَرْمَجَةُ بِلُغَاتٍ مِثْلَ الـ **C** مُنْتَهَى الْمَتَعَةِ بِالنَّسْبَةِ لَهُ.

وَ الْحَقُّ أَنِّي لَا أَرَى الْحَقَّ مَعَ أَيِّ مِنْهُمَا، بَلْ أَظْهَرُ رَأْيَا مَا أَمْتَلَأُ مِنَ الْكُوبِ وَ لَمْ يَنْتَبِهْ إِلَى مَا فَرَّغَ فِيهِ؛ فَالْمُبْرِمَجُ بِاللُّغَةِ الْكَاتِنِيَّةِ الصَّرْفَةِ لَمْ يَنْتَبِهْ إِلَى مَا بَهَا مِنْ عُيُوبٍ كَانَ يُمْكِنُهُ تَفَادِيهَا عِنْدَ الْبَرْمَجَةِ بِالنَّمَطِ الْوِظَائِنِيَّ أَوْ الْإِجْرَائِيَّ، وَ كَذَا فَالْمُبْرِمَجُ الْوِظَائِنِيَّ أَوْ الْإِجْرَائِيَّ (إِنْ صَحَّ لِي أَنْ أَسَمِّيَهُ هَكَذَا) لَمْ يَنْتَبِهْ إِلَى أَنَّهُ كَانَ يُمْكِنُهُ الْإِفَادَةُ مِنْ مِيزَاتِ الْبَرْمَجَةِ الْكَاتِنِيَّةِ بِدَلَالَةٍ مِنَ الْمُعَانَاةِ الَّتِي وَضَعَتْ نَفْسَهُ فِيهَا حِينَمَا اسْتَعْدِمَ نَمَطَ الْبَرْمَجَةِ الْآخَرَ بِمُفْرَدِهِ.

وَ قَوْلِي هُنَا يَسْتَنْدُ إِلَى أَنَّ كُلَّ نَمَطٍ بَرْمَجِيٍّ لَهُ مِيزَاتُهُ وَ عِيُوبُهُ الَّتِي يَعْلَمُهَا كُلُّ مَنْ دَرَسَ عِلْمَ هِنْدَسَةِ الْبَرْمَجِيَّاتِ وَ مَارَسَ الْبَرْمَجَةَ عَمَلِيًّا بِشَكْلِ كَبِيرٍ، وَ مَا دَامَ الْأَمْرُ هَكَذَا فَإِنَّهُ مِنَ الْأَفْضَلِ لَوْ اسْتَطَاعَ الْمُبْرِمَجُ أَنْ يَسْتَعْدِمَ كُلَّ نَمَطٍ مِنْ هَذِهِ الْأَنْمَاطِ فِي أَفْضَلِ التَّوَاخِي الَّتِي يَتَمَيَّزُ فِيهَا ذَلِكَ النَّمَطُ وَ يَكُونُ مِنَ الْأَكْفَأِ اسْتِعْمَالَهُ فِيهِ.

وَ مَا دَامَتْ اللُّغَةُ الْبَرْمَجِيَّةُ هِيَ الْأَدَاةُ الَّتِي يَسْتَعْمَلُهَا الْمُبْرِمَجُ فِي عَمَلِهِ فَإِنَّ هَذِهِ الْأَدَاةَ سَتَكُونُ هِيَ الْفَيْضَلُ فِي تَحْدِيدِ الْإِخْتِيَارَاتِ الَّتِي سَيَقُومُ الْمُبْرِمَجُ بِاخْتِيَارِ نَمَطِ الْبَرْمَجَةِ الْمُنَاسِبِ مِنْ بَيْنِهَا، فَلَوْ كَانَتْ اللُّغَةُ صَرْفَةً النَّمَطِ (سِوَاهُ أَكَانَتْ كَاتِنِيَّةً أَوْ وِظَائِنِيَّةً أَوْ إِجْرَائِيَّةً) فَإِنَّهَا تَقُومُ عِنْدَئِذٍ بِفَرْضِ ذَلِكَ النَّمَطِ عَلَى الْمُبْرِمَجِ وَ تُكَبِّلُهُ بِهِ وَ تُقَيِّدُهُ بِهِ تَمَامًا فَلَا يُفَارِقُهُ، أَيُّ أَنَّهَا تُعْطِيهِ كُلَّ مِيزَاتِ ذَلِكَ النَّمَطِ وَ لَكِنَّا تُحْمِلُهُ كُلَّ عِيُوبِهِ، وَ فِي نَفْسِ الْوَقْتِ تَحْمِيهِ مِنْ عُيُوبِ الْأَنْمَاطِ الْآخَرَى وَ لَكِنَّا تَحْرِمُهُ مِنَ الْمُمَيَّزَاتِ الَّتِي تَتَمَتَّعُ بِهَا وَ لَا تُوجَدُ فِي نَمَطِهَا الْمُخْتَارِ.

وَ الْمُجَادَلَةُ بِأَنَّهُ يُمَكِّنُ بَعْضَ الْإِلْتِفَافِ بِنَاءَ نَمَطٍ بِاسْتِعْمَالِ نَمَطٍ آخَرَ هُوَ نَوْعٌ مِنَ الْإِسْتِجَارَةِ بِالنَّارِ مِنَ الرَّمَضَاءِ؛ فَمَا دَامَ هُنَاكَ الطَّرِيقُ الْأَسْهَلُ وَ هُوَ إِتَاحَةُ الْفُرْصَةِ أَمَامَ الْمُبْرِمَجِ لِيَخْتَارَ النَّمَطَ الَّذِي يَرْتَضِيهِ لِتَطْبِيقِهِ؛ فَلِمَ تَكَلَّفَ الْعَنَاءَ وَ إِثْقَالَ عَاتِقِ الْمُبْرِمَجِ بِمَهْمَةِ الْإِلْتِفَافِ وَ الدُّورَانَ لِيَحْصُلَ عَلَى مَا يُرِيدُ؟! ثَمَّ فِي النِّهَايَةِ نَجِدُ أَنَّ لِهَذَا الْأَمْرَ أَضْرَارَهُ الَّتِي لَا تَخْفَى عَلَى أَحَدٍ مِنْهَا:

- هناك أنماطٌ يكون من الصعب أن تُعلِّمها للمُبرمج المُبتدئ في البداية قبل الأنماط الأخرى، مثل البرمجة الكائنية التي لا يُمكن لأحدٍ أن يفهمها جيداً قبل أن يفهم أولاً البرمجة الإجرائية و الوظائفية، و ما يحدث عندما يكون أمامه لغة برمجة كائنيةٍ صرفةٍ مثل الـ `java` و يبدأ تعليمه بها ببرنامج هو الأبسط في أي لغةٍ مثل:

```
public class Hello{
    public static void main(String [] args){
        System.out.println("Hello CCSC");
    }
}
```

ثم يتساءل ببراءةٍ عن معاني `public class` و `static`: أن مُعلِّمه يبدأ في قول كلامٍ كثيرٍ من عينية (لن نتحدث عن هذا الآن) و (اكتبها كما هي و سأخبرك بالسبب في وقتٍ لاحق) و غيره الكثير من الكلام الذي لا يجد في نفس المُستمع إلا دُور إثارة الحيرة و الإرتباك بشدة.

و يكون بعدها من الجلي أنه لا يُمكننا تعليم البرمجة بلغةٍ مثلها لأننا نحتاج إلى المرور على اللغات الأخرى أولاً قبل المَجئ إليها، و بالتالي يَضِيع الكثير من الوقت على المُبرمج المُبتدئ مُتخبطاً بين اللغات المختلفة لمُجرّد تعلُّم البرمجة بصورةٍ جيدة، و هو الأمر الذي لا أجد له داعياً من أي نوع.

و خلاف ذلك الوضع مع اللغات التي لا تدعم البرمجة الكائنية مثل الـ `C`؛ حيث يكون على الراغب في تعلُّم البرمجة الكائنية الذهاب إلى لغةٍ أخرى، و لعل ما يُخفِّف من عناء ذلك و مشقته في حالة مُبرمجي الـ `C` أنه سيجد لغة الـ `C++` التي يُمكنه أن يُغمض عينيه بعض الشيء عن الفروقات التي بينها و بين الـ `C` ليقول (بضمير غير مُستريح) أنهما ذات اللغة. و أنه لم يُضطر إلى ترك اللغة الأولى لتكملة مشوار التعلُّم البرمجي. و لكن الحقيقة تظل هي الحقيقة من حيث أنه حتى في هذه الحالة فإن الأمر كان مُرهقاً بالنسبة له و لم يكن كما يُحاول إيهامنا نزهةً ريفيةً جميلةً.

و حتى في تلك الحالة يظل القول بأن المُبرمج ما زال تحت سيطرة نوع واحدٍ من الأنماط البرمجية حقيقةً لها مُنتهى القوة و العُنْفوان، و يظل الحصول على ميزات الأنماط البرمجية الأخرى مرهوناً بمدى البراعة و الدهاء و الوقت و الجهد، مما في إمكان المُبرمج إبداءه و بذله للإلتفاف حول التَّمَط الوحيد للغة لبناء التَّمَط المطلوب. و هكذا سنرى إهدار الوقت و الجهد و حيل الحِواة و المشعوذين في أكواد المُبرمجين من تلك النوعيات المسكينة المقهورة على أمرها.

لذلك و بكل بَسَاطَةٍ أَسْتَطِيعُ أَنْ أَقُولَ أَنَّ الْمَبْدَأَ الَّذِي يَجِبُ عَلَى كُلِّ لُغَةٍ بِرْمَجَةٍ أَنْ تَنْتَهِجَهُ عِنْدَ التَّعَامُلِ مَعَ أَنْمَاطِ الْبَرْمَجَةِ الْمُخْتَلِفَةِ هُوَ: التَّهْجِيْنِ، وَ أَقْصَدُ بِالتَّهْجِيْنِ هُنَا مَزْجَ أَقْصَى قَدْرِ مُمَكِّنٍ مِنَ الْأَنْمَاطِ الْبَرْمَجِيَّةِ فِي اللُّغَةِ بِحَيْثُ يَكُونُ الْمُبْرِمَجُ قَادِرًا عَلَى اسْتِخْدَامِ أَيِّهَا حِينَمَا يُرِيدُ ذَلِكَ، مَدْفُوعًا بِالطَّبْعِ إِلَى ذَلِكَ بِقَرَارَاتٍ تَصْمِيْمِيَّةٍ تُخْبِرُهُ أَنَّ ذَلِكَ هُوَ الْقَرَارُ التَّصْمِيْمِيُّ الْأَفْضَلُ فِي هَذِهِ الْحَالَةِ.

و لَيْسَ الْأَمْرُ بِالصَّعُوبَةِ الَّتِي سَيَتَّصِرُهَا الْبَعْضُ؛ وَ الْحَقُّ أَنَّ إِلقاءَ نَظْرَةٍ وَاحِدَةٍ عَلَى لُغَةِ **إِبْدَاعِ** وَ قَوَاعِدِ تَصْمِيْمِهَا يَكْفِي لشرحِ الْأَمْرِ بِكُلِّ وُضُوحٍ بَدُونَ الْحَاجَةِ إِلَى مَزِيدِ قَوْلٍ؛ فَفِي **إِبْدَاعِ** يُمَكِّنُ لِلْمُبْرِمَجِ أَنْ يُبْرِمَجَ بِالتَّمَطِّ الْبَرْمَجِي الَّذِي يُرِيدُهُ سِوَاءُ أَكَانَ نَمَطًا إِجْرَائِيًّا (أَوْ الْبَرْمَجَةَ بِالْأَعْرَاضِ الْجَانِبِيَّةِ **side effects** كَمَا يُسَمُّونَهَا أَحْيَانًا) أَمْ كَانَ نَمَطًا وَظَائِفِيًّا (وَ هُوَ يُضَادُّ الْأَعْرَاضَ الْجَانِبِيَّةَ تَمَامًا) أَوْ حَتَّى التَّمَطِّ الْكَائِنِي الْوَاضِحِ الْقُوَّةِ، وَ كُلُّ هَذَا لَمْ يُعَقِّدْ تَصْمِيْمِ اللُّغَةِ كَمَا قَدْ يَتَّصِرُ الْبَعْضُ، وَ لَكِنَّهُ عَلَى الْعَكْسِ كَانَ لَهُ سِمَةٌ التَّسْهِيلِ وَ التَّخْفِيفِ الشَّدِيدِينَ (عَلَى الْأَقْلِ بِالنِّسْبَةِ لِي).

الشُّمُولُ:

حينما تُريدُ برمجة تطبيقٍ نظاميٍّ يَحْتَاجُ إلى الأداءِ الأفضل: فعليك بالبرمجة بلغة الـ C، و حينما تُريدُ برنامجاً يتعامل مع النوافذ و الواجهات المرئية أو قواعد البيانات فعليك بالـ java أو الـ visual basic أو الـ C#.

و ... و ...

ما فات هو كلامٌ أثقُ أن الكثيرين (إن لم تكن الأغلبية) قد سمعوه مراراً و تكراراً في المُنتديات المُختلفة و الكتب المُتنوعة، التي تناقش موضوع الترجيح و المُفاضلة بين لغات البرمجة المُختلفة، و لا تكون الفائدة التي يخرج بها القارئ بعد ذلك إلا الإحساس أن كل تلك اللغات لا يُغني أحدها عن الآخر، و لا يُمَثِّل واحداً منها دور الخادم الجيد في كل مجال، و هو الإحساس الذي لا يُسعد أحداً بحالٍ من الأحوال.

و أنا أجد أنه من حقى كمبرمجٍ بائس أن أفق لأتساءل عن السبب في هذا، فهل السبب أن كل مجالٍ من مجالات الحاسوب البرمجية يختلف عن المجالات الأخرى بدرجة تجعل اللغة التي تناسبه برمجياً لا تُناسب الأخرى بنفس القدر؟

أم أن السبب أن كل لغةٍ كان لها مُجتمعها التقني الذي ركّز على جانبٍ من الجوانب البرمجية، و تراخى عن تغطية المناطق الأخرى بذات القوة، و من ثم اشتهرت اللغة بمجال التركيز الأكبر و أهملت في الجوانب الأخرى؟

أم قد يكون السبب أن كل مجالٍ من المجالات كان له أهله ممن قام على أكتافهم مُعظم (أو كل) بنائه، و كانوا قدراً يستخدمون لغةً موحدةً أو لغاتٍ قليلة العدد في ذلك البناء، و بالتالي كانت تلك اللغة (أو تلك اللغات) هي الوحيدة أو الأكثر ظهوراً عند أي حديثٍ برمجيٍّ عن ذلك الجانب، و رُويداً رُويداً أَصْبَحَتْ كأنها اللغة (أو اللغات) المُمَثِّلة له فعلياً؟

أم قد يكون السبب الرئيس هو كل تلك الأسباب مُجمعة؟

الحق الذي أراه أن كل تلك الأسباب لها واقعٌ يُحقِّقها و تنطبق عليه في تاريخ لغات البرمجة بدرجات قوةٍ تختلف من سببٍ لآخر؛ فالسبب الأول الذي يقول أن هناك مجالاتٍ برمجيةٍ تختلف كثيراً عن باقي المجالات (و بالتالي فإن لغات البرمجة ذات النوعيات التي تُناسب تلك المجالات لا تُناسب المجالات البرمجية الأخرى) يتحقق في أحوالٍ عديدةٍ منها ما يلي:

● أنظمة الذكاء الاصطناعي:

التي تحتاج إلى استخدام القواعد المنطقية في عملها، فمثل هذه الأنظمة لها مُستوى عالٍ للغاية من التجريد و البعد عن الآلات و طرق عملها، و بالتالي فإن لغة البرمجة التي تُستخدم فيها لا بد أن

تكون بعيدة كل البعد عن التعقيد، بل يجب أن تكون سهلةً مُتَدَقِّقَةً غزيرة الإنتاجية و تستخدم المنطق، مثل لغات الـ **prolog** و الـ **logo** و غيرهن من نوعية تلك اللغات المنطقية العالية للغاية في المستوى.

و لا يمكن مع مجالات كهذه أن نستخدم لغات مُنخِضَةِ المُستوى مثل الـ **C**؛ فكل ما سنجنيه حينها هو الإحباط و اليأس في اليقظة، و الكوابيس و بلل الفراش في المنام.

● أنظمة قواعد البيانات و التعامل معها:

حيث نجد أنه يتم استخدام نوع خاص من اللغات البرمجية (مهما اختلف شكله فهو متشابه) و يُسمى "لغات الإستعلام المُهيكل **structured query languages**" للتعامل معها، مثل الـ **SQL** القياسي و الـ **oracle SQL** و الـ **MS SQL** و غيرهن، و التقنية الجديدة التي ابتكرها في **microsoft** و هي تقنية الـ **LINQ** التي يُستهدف أن تُستخدم للتعامل مع كل أنواع مصادر البيانات و ليس مجرد قواعد البيانات فقط،

أى أنه لم يتم استخدام لغات البرمجة العامة العادية مثل الـ **java** و الـ **C#** على الرغم من أنه يمكننا التحايل لذلك و صنع مكتبات لتلك اللغات تقوم بعمل الصياغات المُختلفة للغات الإستعلام (و إن كان ذلك صعباً جداً)، لكن الأمر ليس في إمكانية فعل ذلك من عدمه إنما يكمن سر الموضوع في طبيعة التعامل مع قواعد البيانات عامة.

و لذلك نجد أنهم لم يقوموا بدمج إمكانيات لغات الاستعلام في اللغات البرمجية العامة مباشرة، بل قاموا بإعطائها القدرة على استخدام أوامر لغات الاستعلام الصريحة، و إن كان هناك بعض التغليف اللازم لقبول تلك العبارات في بيئة لغة البرمجة العادية، و المثال التالي بلغة الـ **C#** يُرينا مثل ذلك التغليف عملياً:

```
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText =
    "SELECT CustomerID AS MyCustomer, CompanyName, Address " +
    "FROM Customers AS Cust " +
    "WHERE CustomerID = 'ALFKI'";
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
```

على الرغم من أنه كان من الممكن الاستغناء عن كل ذلك بصنع إجراء يأخذ كمعاملات:

- أسماء الحقول التي سيتم اختيارها للتعامل معها،
- اسم الجدول الذي يحويها،
- الشرط الذي يجب أن ينطبق على القيم المُختارة.

و هذا كما قلنا صعبٌ للغاية، و لكن الأمر ليس فى الصعوبة فقط إنما فى الإختلاف الشديد عن البرمجة العادية و هو الأمر الذى تُعزى إليه هذه الصعوبة نفسها و تُعد جانباً من جوانب تجلياته.

● تطبيقات الواجهات المرئية:

التي هى من أسهل ما يُمكن عند التعامل معها باستخدام لغات البرمجة الكائنية، و أضعب ما يُمكن لو تم التعامل معها بلغات البرمجة الإجرائية أو الوظائفية؛ حيث فى اللغات الكائنية يُمكننى ببساطة أن أقول (بمُفردات اللغة البرمجية) أننى أريد كائناً من نوع النافذة، و هذا الكائن يَحوى زراً له القيمة الفلانية، و صندوق نص له القيمة العلانية و...و... إلى آخره، أى أن الأمر يَسيرُ جداً و منطقيٌ للغاية.

فماذا عن اللغات الوظائفية أو الإجرائية و تعاملها (بطريقةٍ وظيفيةٍ أو إجرائيةٍ صرفة) مع هذا الأمر ؟

أنا شخصياً قد تعاملتُ مع مكتبة الـ **openCV** من خلال عملى فى مجال مُعالجة الصور الرقمية **digital image processing**. و بالطبع كنتُ أتعامل معها باستخدام لغة الـ **C++** للحصول على أفضل أداءٍ ممكنٍ (و لأسبابٍ أخرى ليس هذا محلها). و يُسعدنى أن أقول أن الأمر كان مأساةً بالنسبة لى.

فمثلاً كان عَلَيَّ أن أستدعى الدالة (**cvNamedWindow**) التي لها الترويسة **header** التالية:

```
int cvNamedWindow(
    const char* name,
    int flags = CV_WINDOW_AUTOSIZE
);
```

لكى تقوم بإنشاء نافذةٍ مرئية، ثم كان عَلَيَّ أن أستدعى دالةً أخرى لى أقوم مثلاً بضم عمود تعقب **track bar** إلى تلك النافذة التي صنعناها، و هى دالة (**cvCreateTrackbar**) التي لها الترويسة التالية:

```
int cvCreateTrackbar(
    const char* trackbar_name,
    const char* window_name,
    int* value,
    int count,
    CvTrackbarCallback on_change
);
```


و لكي أقوم بتكرار الأمر (أى إنشاء أكثر من نافذة و ضم عمود تعقب لكل نافذة) يكون على أن أقوم بتكرار الأوامر ذاتها مرارا؛ لأن خاصية كتابة صنف يفعل ذلك ثم الإشتقاق منه فى كل مرة غير موجودة (بالطبع) فى التّمْطِينِ الوِظَائِفِي و الإِجْرَائِي، بل هى (و ياللعجب) الفكرة الرئيسة فى البرمجة الكائنية.

● **بناء أنظمة البنية التحتية البرمجية للحاسوب** مثل أنظمة التشغيل و أدوات البرمجة المختلفة من مترجمات و مُفسِّرات و مُجمِّعات و غيرهن؛ ففى هذا المجال نجد أن لغة الـ C هى اللغة الأشهر و الأكثر تغلغلا و لا يُنَافِسُها إلا نسختيها الكائنتين الـ ++C و الـ objective C (و ربما قليل جداً من اللغات الأخرى)؛ بسبب أنها حينما صُنعت منذ البداية كانت قد صُمِّمت لتكون لغة بناء للبنى التحتية فى الأصل، وبذلك وضعوا فيها كل المواصفات اللازمة لهذا فأصبحت انعكاساً قويا للغاية للفكر البرمجى فى تلك الناحية (و الإنعكاس هنا كان بالطبع حسب طريقة تفكير مُصمِّمِها و ليس هو بالإنعكاس الوحيد المُمكن).

و قد استسلم أغلب المُبرمجين لتلك الحقيقة و لم يُحاولوا إلا القليلون منهم بحالٍ من الأحوال تغييرها بالإستغناء عن لغة الـ C التى كرهها الكثيرون. و بالتالى كان من الطبع أن اللغات الجديدة التى يتم تصميمها سوف يتم إهمال تحقيق المواصفات اللازمة للتعامل مع البنى التحتية للحاسب فيها؛ لأنه لا حاجة لهذا مع الرضا باحتلال الـ C لعرش ذلك المجال، و الأكثر طبيعية هو الاهتمام بإعطاء اللغة الجديدة الميزات التى تؤهلها للتخصص فى مجالٍ آخر يُمكنها أن تُنافس فيه بقوة.

الأمثلة السابقة كلها كانت لتوضيح أن هناك بالفعل مجالات برمجية تختلف عن المجالات الأخرى، بصورة تجعل اللغة البرمجية التى تُناسب التعامل معها بقوة لا تُناسب العمل فى المجالات الأخرى، إلا بتعسفٍ لا يُؤدى إلا إلى بقاء العمل و عدم كفاءته (هذا إن نجح من الأصل).

لكن هناك مجالات أُظن أنها تختلف عن المجالات الأخرى فى طريقة التعامل معها برمجيا، و بالتالى صارت لها لغاتها البرمجية الخاصة، بل صار للغاتها نفسها سمّا خاصا لا تُخرج عنه، و منها مجال برمجة مواقع الشبكة الذى يكاد يكون من المعلوم من العقل بالضرورة عند كل المُبرمجين أنه مجال له صفته الخاصة التى لا تُشبهه صفات المجالات الأخرى، و مع الزمن و توطُّن هذه الفكرة فى عقول المُبرمجين منذ تعلمهم البرمجة أصبح الأمر بدوره تابوها من التابوهات المُحرّمة فى المُجتمَع البرمجى و لم يتعرض له أحدٌ بالتغيير الجرى إلا مؤخراً.

و أنا أريد فى هذه النقطة أن أؤكد أن التفرقة بين برمجة البرامج العادية و برمجة مواقع الشبكة، و القول بأن للغات التى تُؤدى كل عمل من الأعمال أشكال تصميم مختلفة؛ أمرٌ خاطئ، و التفرقة هنا هى تفرقة زورٍ لا تجوز فى شرع العقل؛ فالإختلاف الوحيد فى برمجة البرامج العادية و لُسنَمَها (برامج سطح المكتب) و برمجة مواقع الشبكة يقع فى الصيغة النهائية التى سيخرج بها البرنامج،

من حيث كونها صيغةً تستهدف المُعالِجَ مباشرةً أم تستهدف مُفسِّراً؛ فمواقع الشبكة تعتمد على برامج المُتصفِّحات **browsers** في عملها مثل **Firefox** و **internet explorer** وغيرهما من المُتصفِّحات دائماً، ولا تعمل بأي حالٍ من الأحوال بصورةٍ مُنفردةٍ مُستقلة، في حين أن برامج سطح المكتب لها الخيار في أن تكون مُعتمدةً على مُفسِّرٍ مُعيَّنٍ مثل برمجيات الجافا التي تعمل على منصة الـ **JVM** و برامج الـ **C#** و الـ **visual basic.NET** اللواتي يعملن على منصة الـ **.NET**، و لها كذلك الخيار في أن تكون مُستقلةً تعمل مباشرةً على المُعالِجِ و تتحاور مع نظام التشغيل لتطلب منه ما تُريد من مهام بنفسها لا من خلال وسيط.

و بالتالي نري أن التفرقة بالفعل غريبةً للغاية في تصميم لغات البرمجة؛ و قد قلتُ من قبل أنني لا أجد أي فارقٍ حقيقي من وجهة نظر المُبرمجِ عالي المستوى، و دليلٌ على ذلك لغاتُ الـ **java script** التي لا تُشبه لغة الـ **HTML** من قريبٍ أو بعيد. و تُستخدَم في برمجة مواقع الشبكة ضمن أكواد الـ **HTML**، و لماذا أضرب المثل بالـ **java script** في حين أن الحبيبة الغالية الـ **java** موجودةٌ في مجال تصميم مواقع الشبكة بكل عنفوانها و هي اللغة التي تُستخدَم في الأصل لبرمجة برامج سطح المكتب، و أنا هنا آخذ الـ **java** كدليل أقوى و أكثر حسماً في هذه النقطة لأنها تقول كلامي عن الافتعال في التفرقة بين نوعي لغات البرمجة (لسطح المكتب و للشبكة) بصورةٍ عمليةٍ لا جدال فيها.

بل إن هناك من حطّم هذا الفارق بشكلٍ واسع، حيث قاموا في شركة قوقل بجعل من المُمكن تحويل برامج سطح المكتب ذات الأكواد الأصلية **native** إلي برمجياتٍ تعمل علي الشبكة من خلال **Google Native Client**. أي أنه يكفيك كتابة برنامجك بأي لغة برمجةٍ عادية، ثم ترجمته إلي لغة الآلة الخاصة بالمُعالِجين **Intel x86** و **ARM**؛ و سيكون بإمكانك تشغيله بصورةٍ عاديةٍ ليعمل علي المُتصفِّحات إذا وافق بعض الشروط الأساسية.

و يُمكنكم مشاهدة مُحاضرة (beyond javascript: programming the web) (with native client) علي قناة (google IO) و التي يتحدّث فيها (david springer) بالتفصيل عن الموضوع و يُقدِّم مثلاً عملياً عليه.

هذا عن السبب الأول، أما عن السبب الثاني و هو (أن كل مجالٍ من المجالات كان له أهله ممن قام على أكتافهم معظّم بنائه و كانوا قدراً يُستخدمون لغةً موحَّدةً أو لغاتٍ قليلة العدد في ذلك البناء، و بالتالي كانت تلك اللغة أو تلك اللغات هي الوحيدة أو الأكثر ظهوراً عند أي حديثٍ برمجيٍّ عن ذلك الجانب، و رُوِّدًا رُوِّدًا أَصْبَحَتْ كأنها اللغة أو اللغات المُمثِّلة له فعلياً) فهو بدوِّره له من الحقيقة نصيبه، و الواقع يدلُّ على ذلك.

حيث لو أخذنا على سبيل المثال بناء بيئات سطح المكتب في أنظمة التشغيل المُختلفة مثل **KDE** و **GNOME** و **LXDE** في الأنظمة شبيهة اليُونِكس **unix-like**، و غيرهن من البيئات المشهورة الأخرى: لو جَدْنَا أنها يغلب عليها أنها قد بُنِيَتْ باستخدام لغة الـ **C++** التي يتحقق فيها شرط اللغات التي

تصلح لهذا المجال، و لكن على الرغم من ذلك فقد كان من المُمكِن أن يتم بناؤها بلغاتٍ أخرى غير الـ C++ مثل الـ java أو الـ object pascal، مثلاً، إلا أن ذلك لم ينتشر لأن المُبرمجين الذين قاموا بذلك العمل كانوا مَمَّنَّ يَسْتخدِمون لغة الـ C++ بِحُكْمِ كَوْنِهِم من المُهْتَمين بتطوير و بناء النُظْمِ الأساسية أو البنية التحتية في عالم الحاسوب، و بالتالي أصبح استخدام الـ C++ في هذا المجال أمراً طبعياً، بل و أصبح مع الزمن من المُقدِّسات المُحرَّمِ لمسها في عالم البرمجة.

و على الرغم من كَوْنِ استخدام لغاتٍ أخرى في هذه المجالات قد يكون أكثر نفعاً من حيث: السهولة في البناء، و الكِبَرِ في الإنتاجية، و البساطة في المُتَابَعَةِ و التطوير (و هي الأمور التي تسعى خلفها كل دراسات هندسة البرمجيات)، إلا أن الأمر قد صار مُقدَّساً بالفعل عند المُبرمجين القُدَامى و المُحدَثين، و صار المساس به و لو من بعيدٍ مُثيراً لعوا صفٍ من النقد الهَدَامِ لصاحب التعليق و لأفكاره، و أظن أنه لو كان مُجمَعُ المُبرمجين النظاميين و ثنياً: لعبدوا آلهةً يُخصِّصونه للـ C و الـ C++ و لقدّموا لها قرابيناً بشريةً ممن يكفرون بهن.

و هذه هي خطورة كَوْنِ المُهْتَمين بمجالٍ مُعيَّن من المجالات الحاسوبية من مُستخدِمى لغة برمجةٍ مُفْرَدَةٍ؛ فَرُوَيْدًا رُوَيْدًا سوف تُسيطر تلك اللغة على ذلك المجال حتى و إن لم تكن خصائصها الذاتية مُناسبة له تماماً، و سيكون أمراً منطقياً أن تصير تلك اللغة البرمجية هي المُمَثِّلُ الرسمي الوحيد لذلك المجال، و أن تصير باقى اللغات كالأيتام على موائد اللثام عند وُلوج مُطوِّريها إلى عالمه، فلا كرامة لها على الرغم من كَوْنِ بعضها أنسب للمجال من تلك اللغة التي تُقرَن به !

و السبب الأخير و هو أن (كل لغةٍ كان لها مُجمَعُها التقنى الذى رَكَّز على جانبٍ من الجوانب البرمجية، و تراخى عن الإهتمام بالمناطق الأخرى فلم يُغَطِّها بذات القوة، و من ثم اشتهرت اللغة بمجال التركيز الأكبر و أهملت في الجوانب الأخرى) هو بدوره له نصيبه من الصحة.

و على سبيل المثال لا الحصر يُمكننا أن نرى أن حال لغة الـ visual basic في مجال التكامل مع نظام الويندوز windows و جِزْمَةِ برامج مكتب ميكرو سوفت Microsoft office جعلها هي اللغة الأشهر في هذا، و لأن microsoft رَكَّزت على هذا الموضوع بقوة فقد كان من المُعتَاد أن نسمع في فترة عُنْفوان الـ visual basic عباراتٍ على شاكلة (الإنتاجية العالية = visual basic) و (تطبيقٌ به نوافذٌ مرئيةٌ = تطبيقٌ مكتوبٌ بـ visual basic) على الرغم من أن هناك مجالاتٍ أخرى تصلح لها الـ visual basic برمجياً مثل كل اللغات عالية المستوى العامة الاستخدام إلا أنها اشتهرت بهذا بالذات.

و لعل هناك سببٌ آخرٌ يندرج تحت القول بالتركيز على مجالٍ مُعيَّن هو: الإهمال المُتعمَّد؛ ففى بعض الأحيان يكون هناك إهمالٌ مُتعمَّدٌ من مُنتج اللغة البرمجية لمجالٍ من المجالات لغرضٍ ما (هو فى الغالب

تجارى أو يَصَب في خانة هدف تجارى)، و لغة الـ **visual basic** و لغات الـ **NET**. عموماً دليلٌ على ذلك كل الدلالة؛ فالمُبرمج الذى يَسْتخدِم اللغات السابقة يكاد يكون قد حَكَم على نفسه بالبقاء تحت مظلة أنظمة تشغيل **microsoft** للأبد، لأن المَحْمُولِيَّة **portability** لم تكن أبداً فى أى يومٍ من الأيام هدفاً من أهدافها.

و لعل الشئ الذى يُخَفِّف ذلك على مُبرمجى الـ **NET**. هو وجود المشاريع مفتوحة المصدر التى تُغْنَى عن بديلها المُنتَج من قِبَل **microsoft** و أشهرها (و ربما الوحيد) هو مشروع الـ **mono**، الذى يتكون من بيئة برمجية متكاملة IDE يمكن استعمالهما مع لغتي الـ **C#** و الـ **visual basic**، و بيئة العمل التى تحل محل بيئة الـ **NET**. الخاصة بزمن التشغيل على نظام الـ ويندوز و تعمل على باقة أنظمة التشغيل المشهورة مثل اللينوكس و السولاريز و ماك (و التى تم إهمالها بطبيعة الحال فى بيئة الـ **NET**. الخاصة بـ **microsoft**).

و لكنني أظن أنه حتى فى هذه الحالة فإن هناك عوائقاً ستظهر فى طريق من يرغب فى الاستفادة القصوى من اللغة؛ نظراً لأنه سيضطر إلى الحصول على التطويرات الجديدة فى بيئة العمل و اللغة متأخراً عن المبرمجين الذين يتعاملون مع **microsoft** نفسها، فاعتماد التغييرات الجديدة فى **mono** لن يحدث بين يوم و ليلة، بل سيأخذ وقتاً بعد التصريح عن تلك التغييرات رسمياً (و بالطبع سيكون قد تم اعتمادها فى برمجيات **microsoft** لتحصل على السبق فى التسويق كما هى العادة). اللهم إلا إذا تعاونت مايكروسوفت مع فريق عمل **mono** بقوة بحيث يُسأِر فريق عملها فى الجدول الزمنى.

و قد أدى وجود براءات اختراعاتٍ لمايكروسوفت فى بنية عمل مشروع **mono** إلى تحذير (ريتشارد ماثيو ستولمان) من استخدامه فى عمل البرمجيات مفتوحة المصدر؛ لأن مايكروسوفت قادرةٌ فى أى لحظةٍ بهذا الشكل على هدم الأمر كله أو على الأقل عرقلته بشدة، و أنا عن نفسى أقبل اعتراضه هذا تماماً رغم أننى كنتُ من مُحترفى البرمجة بالـ **C#** إلا أن الحق أحق أن يتبع، و إسناد الظهر إلى **microsoft** سيكون أكثر قرارات عمرى حماقةً إن اتخذته، لذا فأنا أسأل الله تعالى ألا أضطر إليه فى يوم من الأيام.

كل ما فات كان توضيحاً لأسباب عدم شمولية لغات البرمجة، و تخصصها فى مناطقٍ مُعينةٍ من مجالات الإبداع البرمجى، و ضعف أو غياب وجودها فى باقى المجالات، إلا أننى لم أوضِّح حتى الآن موقفى من كل ذلك الذى ذكرناه، و موقفى واضحٌ و بسيطٌ للغاية، و يُمكن تلخيصه بدون أى إخلالٍ فى جُملةٍ واحدةٍ هى (لغة البرمجة أداة المبرمج، و الأداة لا تكون أداةً جيدةً إلا إذا أغنت عن مثيلاتها فى مُعظم العمل إن لم يكن كله، فإن لم تكن كذلك فإنها لا تستحق الإستعمال و ينبغى طَرْحها جانباً).

أى أن اللغات التى تَشترط على المُبرمج التضحية بكثيرٍ من المجالات ليُمكنه التركيز على مجالٍ مُعينٍ هى أداةٌ سيئةٌ يجب على المُبرمج نَبذها؛ حتى لا تُكبِّله و تُحد من قدراته و أنشطته، أما لغة البرمجة الحقيقية التى تستحق الإهتمام فهى التى تُعْطى المُبرمج القدرة على الإبحار باستخدامها فى أى مجالٍ برمجى يعن له أن يضع قدمه فيه، أو على الأقل مُعظم المجالات البرمجية.

و بالتطبيق على لغات البرمجة الحالية سنرى أن اللغات التي تُحَقِّقُ هذا الشرط قليلةٌ للغاية مُقَارَنَةً بما هو مُسْتخدَمٌ منها بالفعل، و أشهر الأمثلة التي ستُضْرَبُ لهذا النوع من اللغات هي لغات الـ (C#, java, .NET, visual basic, C++) لذا ففى تالِي كلماتي سوف أناقش موقفي من كل لغةٍ مُوضَّحاً صحة هذا الزعم من كذبه.

و أول تلك اللغات هي الـ java و هي مِن أَحَبِّ لغات البرمجة الحالية إلى قلبي؛ لكونها يَتَحَقَّقُ فيها مِن ميزات لغات البرمجة الجيدة عندى الكثير، و على الرغم من هذا فإن عندى انتقاداتٍ حادةٍ للـ java فى بعض المسائل التصميمية (سأَصْرِحُ ببعضها فى أماكنٍ أُخْرَى من هذا الكتاب، و سأفْرِدُ للبعض الآخر أوراقاً علميةً خاصةً به)، لكن الأكثر أهميةً من هذا هو كَوْنُ الـ java قد حَقَّقَتْ ميزة الشمولية هذه إلى حدودٍ كبيرةٍ للغاية، بما يجعلها بحقٍ أشمل اللغات البرمجية التي أعرفها.

فالـ java تُعْطِي المُبرمج القدرة على البرمجة لمجالاتٍ واسعة المدى مثل:

1. برامج سطح المكتب ذات الواجهات الرسومية GUI applications.
2. برامج سطح المكتب النصية console applications.
3. برامج العملاء على الشبكة client applications.
4. برامج الخادومات servers applications.
5. برمجة الأجهزة الذكية مثل الهواتف المحمولة و الثلاجات المُعدَّلة و غيرهن من تلك النوعية من الأجهزة.
6. برمجة الألعاب.
7. التعامل مع قواعد البيانات.
8. و قد استخدمها البعض بالفعل فى بناء أنظمة تشغيلٍ مثل Jx و Jnode و ربما أخريات !

و بالتالى فليس هناك مجالٌ يُمكن أن يُفَكَّرَ فيه المبرمج إلا و جَدَ الـ java واقفةً فى انتظاره تمدُّ إليه يد المساعدة، و أهم من ذلك كون الـ java تعمل على أنظمة التشغيل المختلفة بمنتهى المُساواة و بنفس الكود البرمجي، و كما يُعبَّرُ مُنتَجو الـ java عن هذا المبدأ بقولهم (اكتب مرةً واحدةً و شغِّل فى أى مكان) و يستحق هذا المبدأ كل ما يُبذَل من جهدٍ للوصول إلى تحقيقه.

أما الـ C# فهى لغةٌ قويةٌ بالفعل و أراها أقوى من الـ java لأسبابٍ كثيرة، و لكن و رغم كل هذا فإن الـ C# لا تصمد أمام الـ java إذا ما قارنَّاهما من مُنْطَلَقِ الشمولية؛ و ذلك على الرغم من أنها تصلح لنفس المجالات التي ذكرناها للـ java! و لكن بالطبع مع وَضْعِ جُمْلَةٍ (مع أنظمة تشغيل microsoft فقط) أمام اسم كل مجالٍ مما ذُكِرَ بالقائمة فى الأعلى، و هذا الأمر ليس بالهين أبداً إذا ما علمنا أن نسبة الخوادم servers التي تُستخدَمُ نظام تشغيل القنو/لينوكس تبلغ ما يزيد على الستين

فى المائة من مجموع الخوادم العاملة فعلاً، و أن أشهر المواقع العالمية تستخدمه بدلاً من الـ windows.

و كذا حينما نعلم أن هناك الكثيرين ممن يتحولون سنوياً إلى أنظمة التشغيل الأخرى (و على الأخص القنول/لينوكس) بسبب ضعف الـ windows الأمنى الشديد و تكلفته و تكلفة برامجه المبالغ فيها، بالتالى فإن القول بأن كل البرامج التى سيكتبها المبرمج لن تعمل إلا على أنظمة microsoft فقط هو ضربة فى الصميم لكل ما حازته الـ C# من شمولية، بل و ضربة قاضية تماماً لأن المحمولىة هى واحدة من أهم أركان الشمولية، إن لم تكن بالفعل هى الركن الأهم و الأشد خطراً؛

فشمولية محصورة فى ركن معين من عالم الحوسبة بالمقارنة بالشمولية المنفتحة على العالم الحاسوبى كله تشبه بالضبط موقف الجارية من الزوجة (رغم الغرابة فى التشبيه)، فالزوجة و الجارية متساويتان فى الحق فى المعاملة الحسنة من الزوج أو المالك، و للزوج عليهما حقوق مشتركة كثيرة،

و لكن هل هناك أى مجال للمقارنة بين وضع الزوجة و وضع الجارية؟

أى صاحب عقل سيرفض المقارنة من الأصل، و بالمثل المقارنة بين شمولية الـ java و شمولية الـ C# : فهما إسمياً تكادان تتساويان تماماً من ناحية أنواع التطبيقات و المجالات التى يمكن أن يُستخدما فيها، و لكن ذلك لا يكفى أبداً للمساواة بين الزوجة (الـ java) و الجارية الـ (C#).

و ما قيل عن الـ C# يُقال بالضبط عن الـ visual basic .NET نظراً للتوأمة التى صنعتها Microsoft بين اللغتين حينما ضمتها إلى إطار عمل الـ .NET. و بالتالى هى تأخذ نفس حكم الـ C بلا زيادة أو نقصان.

و نعود إلى اللغة الثالثة الشهيرة فى عالم الشمولية و هى الـ C++، و التى حازت على مكانتها فى عالم الشمولية بلى الذراع (إن كان لى أن أستخدم هذا التعبير)؛ فعلى الرغم من أن الـ C++ لغة تتميز بالشمولية و يمكن للمبرمج بها الإعتماد عليها فى كل المجالات البرمجية التى تخطر على باله (أو لا تخطر) لأى نظام تشغيل أو معالجة تحت الشمس، إلا أن تلك الشمولية فى رأى شمولية خداعة لا يمكن الاطمئنان إليها بحالٍ من الأحوال (على الأقل بالنسبة لى)؛ نظراً لتعقيد اللغة الذى يضيع أى ميزة أخرى تمتلكها و على الأخص فى المجالات التى تتعقد فيها الخوارزميات و الأفكار فلا ينقص الطين بلة تعقيد اللغة البرمجية نفسها.

و لكن لأننا فى هذا المقام نناقش الشمولية فقط فيمكننى أن أقول أن الـ C++ تتربع على عرش الشمولية بجانب الـ java تماماً (و شتآن ما بينهما فى التعقيد).

البساطة و الإستقرار:

من أكثر الأمور التي تُثيرُ غيظي في لغة الـ **C#** أنني بعد أن قضيتُ فترةً لا بأسُ بها أتعلّمها فيها و أتسرّبُ طريقة التفكير التي تتميز بها كسابقتها في العمر الـ **java**: أصبحتُ أحسُّ بأنني لا أعرفُ البرمجة بها على الإطلاق!؛ و يُراوِني الإحساس بهذا كلما رأيتُ مقالاً أو كُتِيباً يتحدث عن الإصدارات الجديدة التي تُصدرها **microsoft** من اللغة كل فترة. فهذه الإصدارات أصبحت رُوَيْدًا رُوَيْدًا تُنفخ في كمية قواعد اللغة حتى انفجرت في وجه المستخدمين الأبرياء (و لى الفخر في أن أكون واحداً منهم).

و بعد أن كنتُ أتفاخر بأنني أفهم الـ **C#** جيداً أصبحتُ أخافُ من التصريح بأنني أستخدمها في البرمجة؛ حتى لا يقوم أحدهم بإحراجي حينما يسألني عن القاعدة الفلائية أو القاعدة العالائية التي ضُمَّت إلى اللغة في الإصدار ذات الرقم الفلاني فيبدو جهلي واضحاً له. و الأمر أصبح مدعاةً للثناء و تقليب الأُكف حيرةً عند التفكير في السبب الذي يقف وراء تلك "التطويرات" و النفخات الجديدة التي تُعطى للغة على فتراتٍ مُنتظمة، حتى أن نحو اللغة الآن لا يكفيه لشرحهُ بمُفرده (شرحاً احترافياً مُختصراً) كتابٌ ضخّم الحجم لا يقوى على إستيعابه المُبرمج العادي إلا في شهورٍ (أو علي الأقل أسابيع) طويلةٍ ينقطع فيها لدراسة اللغة تماماً. و في النهاية لا يكون أمامه إلا تركُ جانبٍ كبيرٍ من اللغة لدراسته في وقتٍ لاحقٍ؛ نظراً لأنه لا يستطيع ترك باقي العلوم البرمجية و التفرغ لدراسة لغة البرمجة كل الوقت (و هي التي لا تمثل إلا جزءاً صغيراً من العمل البرمجي)، و الجدير بالذكر أن الوقت اللاحق الذي يُؤجّل الدارس إليه دراسة ما بقي من اللغة لا يأتي أبداً، و ذلك عن تجربةٍ واقعية.

و هكذا لا يكون أمام المُبرمج إلا خيارٌ من اثنين:
أولهما أن يتفرغ لتعلم اللغة فترةً كافيةً تمكّنه من التمكن الجيد منها، ثم متابعة التطورات و الإضافات الجديدة التي تحدث و تُضاف للغة كل فترةٍ حتى يُصبح مُواكباً لأحدث تقنياتها على الدوام.
و ثانيهما الإكتفاء بمعرفة الجزء الأكبر و الأهم من اللغة و التمكن منه جيداً، ثم الانطلاق إلى العلوم البرمجية الأخرى التي لا بد من التمكن منها بجانب لغة البرمجة العملية.
 أما الخيار الأول فيُناسب من بدأ مشواره في تعلم البرمجة منذ أن كان صغيراً؛ و بالتالي كان أمامه الفترة الكافية للتعلم قبل مواجهة سوق العمل الذي يضغط الأوقات و الأعصاب إلى أقصى الحدود، و لا يُمكن أن يتناسب مع ظروف مثل كاتب هذه الأسطر على سبيل المثال؛ فالعمل يدفع من هم مثله للقراءة و التعلم و الإحتراف في مجالاتٍ أخرى غير لغة البرمجة المعنية، بل وربما يدفعه سوق العمل على إتقان لغةٍ أخرى غير اللغة التي يهتم بها و يجد أنها تُناسب نمط تفكيره.

و أنا مثالٌ جيدٌ على ذلك: فقد كنتُ أستخدمُ في البرمجة لغة الـ **C++** التي أمقتها بعنف، و كنتُ مُطالِباً باحترافها و التمكّن منها حتى أستطيع العمل بها بحرفية في مجال مُعالجة الصور الرقمية، و رغم أنني أحب الـ **java** و الـ **C#** إلا أنني لا أستطيع مُجاراة آخر التحديثات و التغييرات التي تتم لهما؛ لأنني لا أستطيع أن أقضى وقتي كله في تعلم الـ **C++** من ناحية و تعلم الجديد في الـ **C#** من ناحيةٍ أخرى، في حين أن هناك علوماً أخرى يجب أن أُحَصِّلها حتى أكون مُبرمجاً بحق (مثل هندسة البرمجيات، و الخوارزميات، و غيرهن) و التي لم أتعلّمها من قبل لأنني احترفت مجال البرمجة و الحوسبة عموماً منذ فترةٍ لا تزيد على الخمس سنوات.

و هناك من يُناسبه خيار المُتَابَعَة المُستَمِرَة لِلُغَة (غير الذي بدأ في التعلم منذ الصغر) و هو: مَنْ بدأ مع اللغة منذ بداياتها الأولى و أول إصداراتها، و بالتالي كانت لديه الفرصة الكاملة للمُتَابَعَة الدائمة كأفضل ما يكون، و هؤلاء ليسوا بالقليلين بل هم في كل مكانٍ حولنا، و يُمكن أن نراهم كثيراً حينما يأتي الأمر على لغةٍ مثل الـ **C#** التي أصبحت البرمجة بها موضحةً من موضوعات المُبرمجين في السنوات الماضية، و بالتالي انهمك الجميع في دراستها و هي بعد في المهد، و بالمُتَابَعَة تعلموا الجديد أولاً بأول.

أما الخيار الثاني القائل بالتمكّن من الجزء الأكبر و الأهم من اللغة فقط لكي يتبقى الوقت اللازم لتعلم باقى العلوم البرمجية فهو أمرٌ فيه جدالٌ يطول؛ و ذلك لأنه و إن كان مع أصحابه الحق في القول بأهمية تعلم باقى العلوم البرمجية غير لغة البرمجة ذاتها فإنهم لا يتبهنون إلى نقطةٍ أخرى في غاية الأهمية: و هي أنهم بتطبيقهم خيار التجزئى لِلُغَة لن يتمكنوا من فهم الأكواد البرمجية التي يكتبها غيرهم ممن يُبرمجون بنفس اللغة البرمجية و لكنهم يستخدمون مُكوّناتٍ لم يتعلموها هم.

ولن يكون مثل هذا الموقف قليل الحدوث بل سيحدث كثيراً جداً كلما زادت كمية الأكواد التي يضطلع عليها المُبرمجون، و هي كثيرة العدد جداً في هذه الأيام بفضل عالم المصادر المفتوحة و الحاجة المضطّرة للإعتماد عليها حتى لو علي سبيل التعلم.

و هذا يعنى أن أمثال هؤلاء المُبرمجين سيُجبرون إن عاجلاً أو آجلاً على التعرف على مُعظم المُكوّنات التي تركوها خلفهم في اللغة إن لم تكن كلها، و بالتالي يكون من الأفضل لهم أن يفعلوا ذلك من البداية في ذروة تعلم اللغة ما دام الأمر لا مفر منه في النهاية، و من ثم يكون ما نخافه من الإهمال لباقي عنا صر العلم البرمجي أو حتى عدم التنبه الكامل لها حتى الانتهاء من تعلم لغة البرمجة (التي هي الأداة التي يستخدمها المبرمج في عمله ليس إلا).

و هكذا نرى بأم أعيننا الوقت الكبير يضيع في تعلم ما كان يجب أن يكون هو الأمر الأبسط في العملية التعليمية كلها، بينما الأمور الأهم تُترك لما بعد، و بعد كل ذلك تأتي مرحلة التطبيق على المجالات التي سوف تُستخدم فيها كل تلك الخبرات البرمجية مثل برمجة الشبكات أو برمجة تطبيقات معالجة الصور الرقمية. و ليس لنا بعد كل ذلك الوقت المُهدر التعجب من كره الكثيرين من التعمق في

دراسات البرمجيات النظرية مثل هندسة البرمجيات و الخوارزميات و رؤيتهم لها على أنها غير ذات جدوى، بينما ينظرون إلى تعلم لغة البرمجة على أنه هو كل شيء في عالم البرمجة ما دام يأخذ وقتاً كبيراً و يأتي دائماً في المقدمة لكل شيء، و يُرَكِّز عليه الجميع أغلب اهتمامهم.

و هكذا نرى أيضاً كل عام أكواماً من الكتب التعليمية التي لم تعد لها أي فائدةٍ رغم الجهد الذي بذله مؤلفوها فيها؛ لأنها أصبحت تتحدث عن ماضى اللغة لا عن حاضرها، و رغم المال الذي دفعه فيها مُشترؤها الراغبون في تعلم اللغة، و بالتدرّج سيكون على المُتعلِّم التفرقة بين الكتب القديمة و الحديثة بُمُنْتَهَى الحرص حتى لا يقع في معلوماتٍ مغلوطةٍ عن اللغة.

و مثل هذا الموقف نراه في الـ **C#** التي كنا منذ فترةٍ ليست بالبعيدة نقول عنها أنها لغةٌ ذات أنواع ثابتةٍ أو (statically typed) و من ثم فاجأتنا الإصدارات الجديدة أنها صارت أيضاً ذات أنواعٍ متغيرةٍ أو (dynamically typed).

و هذا الحال بالطبع لا يُرضيني ولا أقتنع بجذواه على الإطلاق، بل أقول أن كل هذا عبثٌ سخيفٌ ليس له أن يستمر و لا يحق لأحدٍ أن يفرضه على الآخرين بأى داعٍ من الدواعي، فمنتجوا لغات البرمجة الشهيرة الذين يزيدون في قواعدها يوماً بعد يوم مثل microsoft و لغتها الـ **C#** يجب عليهم أن يتوقفوا حالاً عن هذا و إلا كان لزاماً على معاصر المبرمجين مقاطعة أمثال تلك اللغات تماماً فيصير أمثال أولئك المُنتجين مثلاً لكل من يتحدى إرادة مجتمع المبرمجين و يجبرهم على اللهاث وراءه في عملية تعليمية تعذيبية الطابع لا نهاية لها.

الأسباب التي تجعل مُنتجي لغات البرمجة يفعلون هذا:

إذا ما فُكّرنا في هذه المسألة بعمقٍ لوجدنا أن الأسباب التي تجعل مُنتجي لغات البرمجة يزيدون من قواعد لغاتهم أو "يطورونها" ربما تختلف حسب المنهجية التي ينظر بها ذلك المُنتج إلى المُجتمع البرمجي نفسه، و لنا في لغة الـ **C#** خير مثالٍ على ذلك، فُمُنْتِج لغة الـ **C#** هو microsoft التي لا تسعى (مثلها مثل كل الشركات التجارية) إلا وراء الكسب المادى. و للحصول على المكسب المادى من وراء لغة الـ **C#** من المهم أن يكون لها أكبر عددٍ من المُستخدمين من المبرمجين مُختلفى الطباع و الأهواء، لذا نرى أن الـ **C#** تحوى داخلها من الصفات ما يجعلها تجميعاً من مُختلف أنواع التفكير البرمجية، و يزداد هذا الأمر وضوحاً يوماً بعد يوم بالزيادات التي تُوضَع فيها بدعوى التطوير و التحديث.

و قد ضربنا لذلك مثلاً بالـ **static typing** و الـ **dynamic typing**، و نُضرب له مثلاً آخر هو البرمجة الآمنة و البرمجة غير الآمنة أو الـ **safe programming** و الـ **unsafe programming**، و التي تُمكن مُبرمجي الـ **C#** من استخدام المُؤشّرات التي تُعوّدا عليها في الـ **C++** (بالطبع مع وجود بعض الاختلافات الضرورية). وهو الأمر الذي قد يُسعد بعضاً من

مُبرمجى الـ ++C و يُقنعهم بالفعل بالإنتقال إلى البرمجة بالـ #C، و لكنه فى نفس الوقت يجعل الـ #C بالنسبة لمن هم مثلى خدعة كبرى؛ لأننا فى الأصل لم ننتقل للـ #C إلا لأنها لغةٌ مُتطورةٌ و لا تحتوي على المُكوّنات التى يكرهونها فى عالم الـ ++C و على رأسها المُؤشّرات، فكيف بعد هذا نجد أن مايكرو سوفت "بِتَيْبِلِف" ⁷ !

عدم الخلط بين الإستقرار و الجمود:

على أن كل الكلام السابق الذى نستنتج فيه ضرورة الإستقرار كعاملٍ أساسى من الضرورى توافره فى لغات البرمجة القوية لا يجب أن يدفعنا إلى الظن بأن الجمود سيكون هو مَصير اللغة التى تطبقه على المدى البعيد و تُصر على هذا التطبيق؛ فالواقع أن الأمر غير ذلك نهائياً: لأن الإستقرار ليس معناه الثبات التام للغة عند نحوٍ مُعيّن منذ البداية كما قد يكون قد فهم الكثيرون من سابق الأقوال، بل المقصود بالإستقرار هو: عدم التغيير إلا عند الضرورة الملحة التى لا يُمكننا الإستجابة لها إلا بالتغيير الطفيف فى اللغة.

و أقول التغيير الطفيف لأن التغييرات الكبيرة يجب أن تدفعنا دفعا إلى التفكير فى تصميم لغة برمجةٍ أخرى مُفصلةٍ تماما عن اللغة التى صارت من طرازٍ أقدم، و تحوى داخلها المُكوّنات الجديدة التى نرغب فى ضمها إلى اللغة الأقدم.

و هذا هو الإستقرار الذى أرغب فى الوصول إليه فى لغات البرمجة القوية، و أرى أنه من أول الصفات التى يجب أن تتوفر فى أى لغة؛ فلغة البرمجة القوية جيدة التصميم هى فى حقيقة الأمر لغة البرمجة التى لا تحتاج إلى أى تعديلاتٍ على نحوها إلا على فتراتٍ مُتباعدة، و كما قلنا قبلاً فستكون تلك التغييرات قليلة العدد حتى لا يزداد نحو اللغة ضخامة، و الأهم أنها ستكون طفيفة لا تمس هيكل اللغة أو نموذجها البنائى بأى حال.

أما لغة البرمجة التى ستحتاج إلى تعديلاتٍ كثيرة أو جوهرية على فتراتٍ مُتقاربة فهى لغةٌ سيئة التصميم من البداية، و كان يجب ألا يتم التعجل فى تصميمها، أو كان يجب ألا يتم الإعلان عنها من الأصل حتى تنضج، و لو طبق هذا المبدأ البسيط على لغات البرمجة من البداية لأراحنا من كثيرٍ من أوجاع الرأس.

و فى النهاية فإننى أخص موقفى فى تلك النقطة الجوهرية من مَبْحَثِ تصميم لغات البرمجة فى البنود التالية:

- يجب أن تكون اللغة مُصمّمة على "نارٍ هادئة" حتى لا نحتاج إلى التغيير الكبير فيها فيما بعد.

7 كلمة عامية مصرية معناها (تُخدع) أو (تُغش)، مع إعطاء إحياءٍ قوى أن الفاعل يعتبر الآخر أحقماً مُغفلاً.

- يجب أن تُضم اللغةُ المُكوّنات التي لا يُمكن الإستغناء عنها و لا يُمكن أن تحل محلها مُكوّناتٌ أخرى تم ضمها بالفعل للغة، أى أن الأولوية في الضم ستكون للمُكوّنات التي لا غني عنها و التي تُغني عن غيرها بينما يتم رفض ذلك الغير.
- إذا ما كانت هناك تطوراتٌ في البرمجة تجعل مُكوّنًا جديدًا جديرًا بالضم إلى اللغة فيجب أن يتم حسم مسألة كَوْن هذا المُكوّن الجديد لا يهدم مُكوّناتٍ أخرى أم لا، فإن كان لا يفعل و كان سهل التأقلم مع المُكوّنات الموجودة بالفعل: فيمكن ضمه إلى اللغة بعد فترة استقرارٍ جيدةٍ لا تقل في رأيي عن عامٍ كاملٍ عن التغيير الضروري السابق مباشرة.
- على مدى حياة اللغة يجب أن يكون هناك حدٌ يتم بعده رفض ضم أى مُكوّنٍ للغة مهما كانت أهميته، و هو ما يمكن أن نسميه " حد التُّخمة "، و أقدرُ أنا هذا الحد بحجم الكتاب الذي يشرح نحو اللغة على نحوٍ مُحترِفٍ مُختصر، فلو زاد حجم الكتاب على الخمسين صفحةً من ورق الـ (A4) المكتوب بخطٍ و سَطٍ لا صغيرٍ و لا كبيرٍ (حجم شرح قواعد إبداع) فيمكنني أن أعتبر أن هذه اللغة قد بلغت حد التُّخمة، و يجب التقليل من كل ذلك الكم من القواعد التي لا داعي لها.
- و رغم أن الأمر يخضع للتقدير الشخصي إلى حدٍ كبيرٍ إلا أنه بالحس الشخصي يُمكن التوصل إلى نتائج شديدة الشبه عند النظر إلى لغات البرمجة المُختلفة من عيني أشخاصٍ مختلفين إستنادًا إلى المعيار السابق للتُّخمة، أى أنه رغم كَوْن المعيار غير فائق التحديد إلا أنه عمليًا سيكون ذا قوةٍ توحيدٍ قياسيةٍ كبيرةٍ بين المُبرمجين و الناقدين للغات البرمجة.
- عندما تبلغ اللغة حد التُّخمة و يكون هناك مُكوّناتٌ لا بد من وجودها في لغة البرمجة الجيدة لتلائم الأفكار الجديدة في العمل البرمجي، أو عندما نحتاج إلى تغيير أحد الأفكار الرئيّسة في لغة البرمجة، أو حتى تكون هناك مُكوّناتٌ عَفَى عليها الزمن و يجب التخلص منها، فيجب علينا أن نُفكر في إنتاج لغة برمجةٍ جديدةٍ تستلهم الجيد الذي في اللغة القديمة و تضع الجديد المرغوب في ضمّه محل القديم المرغوب في تركه.
- و هذا يجعل تصميم اللغات أسهل، و تطورها أكثر قابليةً للفهم، كما أنه يُعطى للراغبين في البرمجة باللغة على نفس الشكل القديم لها الفرصة لفعل ذلك، و لا يُجبرهم على التغيير للشكل الجديد ما دام الشكل الجديد قد صيغ على شكل لغةٍ برمجةٍ أخرى مختلفة، لهم كل الحق في التقرير لأنفسهم إن كانوا سيتعلمونها و يستخدمونها أم لا.

الأمن:

ليس المقصود بالأمن هنا الحماية ضد البرمجيات الضارة باختلاف أنواعها من فيروسات و برمجيات تجسس و أحصنة طروادة، بل المقصود به: وجود رقابة من مُصمِّم اللغة على المُكوّنات التي تُضم لها، بحيث يُضمن كونها غير قابلة لأن تكون مصدراً للأخطاء أو القلاقل في البرامج التي تُستخدم فيها، و كذا ضمُّ المُكوّنات التي تُساعد المُبرمج على التغلب على الأخطاء التي تظهر أثناء عمل البرنامج مثل مُعالجة الأغلط.

أى أن الناحية الأمنية في اللغة تتكون على الأقل من عنصرى: تلافى المُكوّنات السيئة و ضمُّ المُكوّنات المُفيدة، و على هذا فيمكننا أن نرى أن عنصر الأمن يصب في ناحية عنصر البساطة و الإستقرار؛ حيث يضع معايير تُحد من الأعداد التي يتم قبول ضمِّها إلى اللغة من المُكوّنات البرمجية، فيُحافظ على كون اللغة في الحد الأقصى المقبول لها من الحجم على الأقل، كما يُضمن أن القواعد التي ضُمَّت إليها هي بالفعل الأجدر بالضم.

و يُمكننا أن نضع الأمن اللغوى بتوسع و شمولٍ في ثلاثة عنا صر هي:

- وجود مُكوّنات تُسهّل العمليات كثيرة الاستخدام، و التي قد يُخطئ فيها المُستخدم إذا ما كانت تتطلب كثيرا من الجهد نظرا لكثرة تكرارها، و بالتالى فإن تسهيل العملية على المُبرمج بتوفير المُكوّنات الجاهزة في لغة البرمجة التي تقوم بالعملية كاملة بالنيابة عنه لا يخدم فقط الناحية الإنتاجية في العملية البرمجية، بل كذلك يخدم بشكلٍ أساسى عملية تبادى الأخطاء البرمجية.
- إستبعاد المُكوّنات أو التعبيرات البرمجية التي تُسبب البلية و الارتباك و من ثم الأخطاء البرمجية بكل أنواعها من أخطاء نُحوية و أخطاء زمن تشغيل و أغلطي منطقية، و مهما جادل المُجادلون في فائدة مثل تلك المكوّنات أو التعبيرات فيجب أن يُقنوا أنه يُمكن الإستغناء عنها؛ لأنها نتاج فكر بشرى وجدها حلا لما واجهه من مشاكل، و ليست تنزيلا من السماء يجب قبوله كما هو، و يُمكن بالقليل من التفكير (أو حتى بالكثير منه) إبتكار مُكوّناتٍ أو أساليبٍ جديدة تُغنى عن تلك المُكوّنات و التعبيرات المُربكة.
- وجود المُكوّنات التي مهمتها الأساسية مُعالجة الأخطاء في زمن التنفيذ (و التي يتكون منها أسلوب مُعالجة الأغلط) فى الـ `java` و الـ `C#` و كثير من اللغات الأخرى، و هذه المُكوّنات تجعل عملية البرمجة متعة حقيقية لأنها تُقرّب الأمر من التفكير العالى للبشر إلى

أقصى الحدود، فبدلاً من البحث اليدوى عن كل الأغلط المتوقَّع حدوثها وكتابة الأحداث اللازمة لمعالجتها: أصبح لدينا تلك الصياغة التى يُمكننا بها ببساطةٍ تحديد الإجراء المناسب لكل نوعٍ من أنواع الأغلط.

الألفة:

في مجلة (مجتمع لينوكس العربى) وبالتحديد فى العدد رقم 7 الصادر فى رمضان 1430هـ (الموافق أغسطس 2009م) كانت هناك قصة قصيرة من ضمن سلسلة قصص قصيرة تتحدث عن مغامرات مبرمج مخضرم فى لغة بيرل perl، و اسم تلك السلسلة هو (من مغامرات المحقق وميرت فونلى)، و كانت قصة ذلك العدد تسمى (سطر بلغة بيرل) حيث فيها تعرّض واحد من المبرمجين المبتدئين لمشكلة ما، فطلب من المحقق (وميرت فونلى) مساعدته على حلها، و هنا فكّر المبرمج الخبير بعض الشئ و من ثم خرج ببرنامج بلغة perl يحل المسألة كلها، و الأهم أنه من سطر واحد!

و الحقيقة أننى منذ أن قرأت تلك القصة (و حتى الآن) و أنا أضعها فى رأسى مثلاً للإدعاء و الصحافة؛ لأن القصة تحاول القول أن لغة perl قوية و ذات إمكانيات شديدة القوة لدرجة أنه يمكنها أن تحل مشكلة صعبة بسطر برمجي واحد. و لو كان الأمر هكذا لما كان هناك ما يضر فى الأمر و كنتُ واحداً من أكثر الناس إسراعاً للبرمجة بـ perl، و لكن الأمر كان على خلاف ذلك تماماً، فنظرة واحدة إلى البرنامج ذى السطر الواحد و هو:

```
perl -MDigest::MD5=md5 -
Owe'@a=@ARGV;@h{map{md5($ )}<>}=@a;@b=values%h;print"@b\n" '*
```

هذه النظرة الواحدة تكفى لنرى أننا قد خدعنا و دُلس علينا حينما زعم المؤلف أن هذا برنامج حاسوب؛ بل كان (و اعذرونى فى التعبير) نبش الدجاج أو طلاس السحرة، فالبرنامج السابق لا يحوى من مواصفات البرامج المهندسة جيداً شيئاً و لا حتى على أقل القليل منها، فهو:

- صعب القراءة و الفهم إلى درجة كبيرة على الخبير بلغة perl؛ لتراكب التعبيرات بما جعله بالفعل شبيهاً بأحجية السحرة و طلاسهم، فما باننا بالمبرمجين البسطاء الذين هم عامة المبرمجين؟!، و الدليل على صعوبة فهم البرنامج أن شرحه الموجود فى القصة كان أكبر من البرنامج بمراحل، حيث و صل إلى ما يُقارب الصفحتين الكبيرتين بينما لم يأخذ البرنامج نفسه إلا سطراً واحداً!

- صعب الصيانة و التطوير؛ لأن تعقيده سيأخذ من المتابع له وقتاً كبيراً لفهمه أولاً، ثم سيأخذ الوقت الأطول للتغيير فيه عند الرغبة فى التطوير، و لأنه معقّد فإن التغيير فيه مغامرة محفوفة بالمخاطر لن ألج (عن نفسى) مثلها أبداً إلا مرغماً.

ولأنه من أول مواصفات البرامج الجيدة هو أن تكون واضحة للقارئ؛ بحيث تكون فى النهاية سهلة الفهم و سهلة التطوير قدر الإمكان: فإن البرنامج السابق يرسب فى معايير البرمجيات الجيدة بمتتهى الجدارة.

و لكننا نظلم البرنامج إذا فكّرنا أن المشكلة فيه هو فقط؛ لأن المُشكِلة الأساسية فى اللغة التى كُتِبَ بها البرنامج نفسه و هى لغة perl. فهذه اللغة من أكثر اللغات التى تحتوى على رموز و علامات تبعدها عن الشكل القريب من الفهم البشرى و تقريبها من الشكل الطلسمى.

و كثرة العلامات التى تعمل عمل البديل للنص البرمجى أمرٌ لا يُساعد إلا على إنتاج برامج خرافية لا تُصدّق فى تعقيدها و شكلها الأسطوريين، تماما كالبرنامج السابق الذى نرى الأمرين قد تحقّقا فيه. و صلب المُشكِلة هو أن العلامات و الرموز التى لا عمل لها فى الحياة العادية للإنسان قد أخذت حجماً مُبالغاً فيه من صياغات اللغة و قواعدها، على حساب الكلمات النَّصِيّة العادية التى هى أقرب للفهم و الإدراك البشريين، و السبب الوحيد الذى قد يدفع إلى هذا (عند مُصممي ذلك النوع من اللغات) هو كَوْن تلك الرموز و العلامات أسرع فى الكتابة و القراءة، و بالتالى فإنها تُؤدّى إلى كِبَر الإنتاجية و هو هدفٌ يسعى إليه الجميع.

لكن المُشكِلة أن من يفتنّ بهذا الرأى يرتكب ثلاثة أخطاءٍ كبيرةٍ هى:

- إعتقد أن البرامج صغيرة الحجم هى بالضرورة سهلة التقبّل و الإستيعاب، و قد قلنا أن هذا خطأً لأن الإسراف المرصى فى العلامات الغريبة على العقل البشرى فى حياته اليومية سيجعل البرامج (بالنسبة له) إلى تعاويدٍ سحريةٍ و طلاسم، فيقضى أمامها أغلب الوقت لفكّها و فهم مُراد السّاحر (أقصد المُبرمج) منها.
- إعتقد أن الإنتاجية تتأثر أول ما تتأثر بسرعة الكتابة! و هو تصورٌ طفولى لا يجب أن يخطر إلا ببال المُبتدئين من المُبرمجين، الذين لا يرون من البرمجة إلا ما يرونه فى مسائل المنهج الدراسى الذى يدرّسونه فى الجامعة، أو مسائل الإمتحانات بأفكارها البسيطة المُكرّرة لا أكثر و لا أقل. فأصبحت كل البرمجة بالنسبة لهم معرفة المطلوب منهم، و من ثم الجلوس أمام الحاسوب و البدء فى الكتابة بسرعةٍ ليُمكنهم الحل و المُراجعة قبل إنتهاء الوقت المحدد.
- أهمل وجود الأدوات البرمجية المُساعدة الحالية التى تجعل بإمكانياتها القوية اللغات التى تُسرّف فى إستخدام الرموز البسيطة و اللغات التى تُسرّف فى إستخدام الكلمات الطويلة على قَدَم المُساواة، و عن تجربةٍ مع لغة الـ C# و بيئة التطوير المتكاملة الـ Visual Studio .NET أقول أن أمر الكتابة السريعة كان بالنسبة لى أمراً منطقياً بدهياً لما تُوفّره لى بيئة الـ Visual studio .NET من إمكانياتٍ مُساعدة، و اقتراحاتٍ فى كل خطوةٍ

أخطوها، و هو ما كان يَجْعَلُنِي أُرَكِّزُ جُلَّ تَفْكِيرِي (بل كله) على الأفكار و الخوارزميات التي سأستخدمها في البرنامج لِيُؤَدِّيَ مهمته بكفاءة و قوة.

بل إن الأمر فاق هذا إلى المُسَاعَدَةِ في إنتاج الكود للأجزاء التي تَحْتَاجُ إلى وقتٍ كبيرٍ من المُبرمج لعملها، بينما يُمكن للحاسوب مُسَاعَدَةُ المُبرمج على القيام بها بكفاءة و سرعةٍ شديدين، مثل مُصَمِّمَاتِ الواجهات الرسومية GUI designers التي تُوَشِّكُ بيئات البرمجة المُتكاملة جميعها على الإِتِّفَاقِ على ضَمِّهَا فِيهَا؛ لما تُزِيحُه من عبء تكوين شكل النوافذ باستخدام الكود مباشرة، بينما باستخدام مُصَمِّمَاتِ النوافذ فإن كل ما علي المُبرمج هو أن يَسْحَبَ المُكوِّنَ الذي يُريدُه و يَضَعُه في المكان الذي يشاء ببساطة و سلاسةٍ شديدين.

و إنني لألاحظ أن إهمال تلك الإمكانيات يُوشِكُ أن يكون آفةً في جَمْعٍ كبيرٍ من المُبرمجين المُخْضَرِّمِينَ، بما يُوحى بأنهم يَتَصَوَّرُونَ أن استخدام مثل تلك الإمكانيات سَيُقَلِّلُ من كفاءتهم، لأنها أدواتٌ للمبرمجين المبتدئين الذين لا يُمكنهم الإعتماد على أنفسهم كل الإعتماد في بناء و كتابة أكوادهم!. و هي النظرة التي أضُمُّهَا بحماسٍ إلى قائمة الأمور التي تُثير جنوني في عالم البرمجة؛ فهذه الأدوات معلومٌ من النظر و الخبرة بالضرورة أنها ذات أثرٍ بالغ القوة في العمل و الإنتاج البرمجي في المشاريع الكبيرة، و تُؤَدِّي إلى طفراتٍ إنتاجيةٍ كبيرة، مما يعنى أنه من العقل السليم استخدامها و الإستفادة منها بأقصى حدٍ مُمكن. و بالتالي يكون التصور السابق أسخف من أن يَخْطُرَ ببال أحدٍ من الأصل فما بالك باعتناقه و تطبيقه في مجال تصميم لغات البرمجة؟! و من ثم فَرَضُ نتائجه علي الجيل الجديد من المُبرمجين!

عدم الخلط بين الألفة و الإطناب المرضى:

مَدُّحُنَا السَّابِقِ فِي كَوْنِ اللُّغَةِ الْبَرْمَجِيَّةِ قَرِيبَةً مِنَ الْكَلَامِ الْبَشَرِيِّ لَا يَجِبُ أَنْ يَدْفَعَنَا إِلَى الظن أن القُرْبَ الكبير من الطريقة البشرية العادية في الحديث أمرٌ مُحَبَّبٌ في البرمجة؛ بل الأمر على العكس من ذلك من حيث أن ذلك القُرْبَ الكبير يُحوِّلُ البرنامج إلى عَجِينَةٍ من الكلمات التي لا يُمكننا أن نُحدِّدَ من بينها ما نُريدُ من معلوماتٍ إلا بعد عناءٍ و جهد. فالمسألة تَحْتَاجُ إلى تَوَسُّطٍ في الإختيار عند تصميم اللغة البرمجية بين اختصار و صِغَرِ حِجْمِ الرموز و العلامات و بين وُضُوحِ و مَفْهُومِيَّةِ الكلمات النصية.

و من الأمور التي تُضَبِّطُ هذه المسألة و يُمكن الإحتكام إليها القواعدُ التالية:

- قاعدة ضَمِّ المألوف من العلامات، فلا يُضَمُّ رمزٌ أو تُضَمُّ علامةٌ إلا إذا كان/ت مما أَلْفُه المُستخدِمُونَ في تعاملاتهم مع النصوص العادية، مثل الفاصلة ، و النقطة . و القوسين () ، والتقليل قدر الاستطاعة من الرموز و العلامات التي يَقلُّ التعامل بها أو يَعدَمُ في الحياة العادية مثل ~ و *

- استخدام الرموز و العلامات المألوفة (لو احتجنا لضمّهما) في نفس الوظيفة التي كانت تُستخدمُ فيها في اللغة العادية، مثل استخدام النقطة ▪ في نهاية الأوامر البرمجية و استخدام النقطتين القائمتين ▪ عند بدء تعبيرٍ أو تركيبٍ من الجُمَلِ.
- استخدام الكلمات بدلاً من الرموز في الأماكن المناسبة لهذه الرموز في الأحيان التي تكون فيها للكلمات القدرة على القيام بوظائفٍ زائدةٍ لا يُمكن للعلامات القيام بها، فمثلاً في إبداع استخدمتُ الكلمات المفتاحية و أو ليس للإستخدامات المنطقية في اللغة، على الرغم من أنه كان بالإمكان استخدام العلامات المشهورة في لغات البرمجة الإنكليزية مثل && || ! في ذلك و كانت ستؤدّي دور الكلمات المحجوزة، إلا أن هناك وظيفة لا يُمكن لهذه العلامات أدائها، و هي تحويل النص البرمجي إلى لغةٍ قريبةٍ من اللغة العادية مما يزيد من مقروئية الكود بما لا يُوصف بالنسبة للمبتدئين و الأطفال الصغار، الذين يُمكن أن تُستخدم اللغة في تعليمهم البرمجة بإذن الله تعالى.

المُكوِّنات التي رُفِضَ ضَمُّها إلى اللفظة

أُرفقتُ هذا الجزء بالرسالة لرغبتني في تقديمه كتطبيق عملي و شرح واقعي لما تُلزم به المبادئ التصميمية (التي سبق و أن أُبديتُ اقتناعي بها في الجزء السابق) كَلِّ من يَسِيرُ بِمُقْتَضَاهَا، و الأثر العميق الذي ستركه على عقله عند نقد المُكوِّنات المعروفة و المتوافرة حالياً في اللغات البرمجية الأخرى، و التي لا شك في كونها بيئة خصبة للتعلم و النقد بالنسبة له.

لذا وضعتُ هنا أهم المُكوِّنات البرمجية التي رُفِضَتْ ضَمُّها للغة **إبداع** تأثراً كما قلتُ قبلاً بقناعاتي التصميمية السالفة الشرح، و من الأمور التي سيلاحظها كل من له علاقة احتكاكٍ بعائلة لغات الـ C أن مُعظم هذه المُكوِّنات تُوجد في تلك العائلة، أو على الأقل في مُعظم أفرادها إن نجا منها واحدٌ عند مناقشة مُكوِّنٍ ما من ضمن ما ذُكر هنا.

و السبب الرئيس هو أنني قد تلقيتُ مُعظم تعليمي الجامعي (و ليس التعليم الذاتي) على تلك العائلة، و لم يَشُدَّ عن تلك القاعدة إلا لغة الـ **visual basic** التي دَرَسْتُها في الفرقة الإعدادية بشكلٍ طفيفٍ للغاية لم يُؤثِّرَ فيَّ أبداً كما أثَّرت فيَّ لغات تلك العائلة، و كذلك لغة الـ **fortran** التي كرهتها بعمقٍ حتى اليوم، و لكنني لم أر فيها مُكوِّناتٍ تُرفض من الأصل؛ و ذلك لكونها من الأجيال الغاية في القِدَمِ برمجيًا و بالتالي ليس هناك ما يُرفض فيها لأنها تُمثِّلُ كلها كما مرفوضاً (كما سيتضح لأي مقارنٍ بين **إبداع** و الـ **fortran**)، على الرغم من أنني قد تأثرتُ بها تأثراً بسيطاً للغاية في مسألة تخص التعامل مع عنا صر الجداول في **إبداع**.

كما أن عائلة الـ C تضم مجموعةً من اللغات الأشهر في العالم البرمجي كله على اختلاف أذواقه و مشاربه، بدايةً من العالم السفلى بكل كهنته اللذين تُمثّلهم لغتا الـ C و الـ ++C، وُصولاً إلى العالم الأعلى بكل طياريه و تُمثّلهم الـ java و الـ C#. و هو الطيف الذي يجعلها تضم معظم مُبرمجي العالم (و لا عجب؛ فهن أربع لغاتٍ تحتل كلّ منهن مكانةً فائقةً القوة بين لغات البرمجة الأقوى). و أنوّه هنا أن القارئ سيجد كلاماً مُكرّراً في أثناء النقاش هو للتنبيه على الأهمية القصوى لما يُناقش أولاً، ثم لكي يكون بإمكان القارئ الكريم قراءة هذا الجزء مُفرداً بدون أن يفقد الكثير من النقاط المحورية مما سلف ذكره، بل يكون بإمكانه اشتفافها من بين السطور على الأقل.

و المُكوّنات المذكورة في هذا الجزء هي على الترتيب:

- 1- الواجهات interfaces.
- 2- الهياكل structs.
- 3- جُملة goto.
- 4- المؤشّرات pointers.
- 5- الإجراءات procedures.
- 6- الوسائط delegates.
- 7- المهمات tasks.
- 8- المُغلّقات Properties.
- 9- المُفهرّسات indexers.
- 10- نموذج الـ Python للأصناف و الإجراءات،
- 11- الـ unions الدامجات.

الواجهات interfaces:

نبذة بسيطة:

الواجهات أو interfaces فى الـ java و الـ C# هى مُكوّنٌ شبيهٌ بالصنّف العادى فيما عدا أنه قد لا يحتوى على بناءٍ implementation لأي من (أو كل) الإجراءات المُعرّفة فيه، أى أنه فى الواجهة يُمكن للمبرمج أن يُعلن عن المُتغيّرات التى يُريدها و يضع رؤوس الإجراءات التى يُريدها كما يفعل مبرمج الـ C/C++ فى ملفات الترويسة header files لا البناء الخاص بالإجراء، و بعد ذلك يُمكن للمبرمج استخدام تلك الواجهة فى الوظائف التى وُضعت لها.

أسباب وجودها و الرد عليها:

• أنها تعطى الهياكل إمكانية الوراثة:

بدايةً فإن وجود الهياكل نفسها يُعد عيباً فى اللغة فى اعتبارى كما سأوضح فيما يلى، لذا ففى إبداع ليس لدينا (الهياكل) من الأصل، كما أن هذا الكلام خداعٌ مُحضٌ لأنها وراثته مُزيّفة؛ فالوراثة ما هى إلا إمكانية أن تكتب الكود مرّةً و احدهً ثم بجملهٍ بسيطةٍ تقوم بإعادة استخدام ذلك الكود مرّةً ثانيةً و ثالثةً و غيرهن بمنتهى الحرية مع الإستفادة بالتطوير المُستقبلى فى الكود الأصل. أما ما يحدث مع الواجهات فهو أن المبرمج يضع تعريفاتٍ للمكوّنات الداخلية للصنّف فى واجهةٍ مُعيّنة، ثم يقوم فيما بعد عند "وراثة" تلك الواجهة ببناء كل تلك المكوّنات من البداية بدون أى توفيرٍ للوقت أو الجهد!، فأين هى تلك الوراثة المزعومة؟!.

إذا يتّضح لنا أن هذه الميزة ما هى إلا فقاعةٌ فارغةٌ تُقال و لا حقيقة لها، و السبب الوحيد لقولها هو دعم السبب الحقيقى لاستخدام الواجهات، و جعل الأسباب كثيرةً تملأ ناظرى القارئ لتقنعه بيسرٍ بأهميتها، فى حين أنه ليس هناك سببٌ حقيقى يستحق النظر إلا واحداً أو اثنين على الأكثر.

• تُعطى الأُصناف خاصية الوراثة المتعددة التى لا تُوفّرها لها الأُصناف المُجرّدة:

يلحق هذا السبب بسابقه من حيث أنه سببٌ مُزيّفٌ؛ فكما قلنا فى السطور الماضية فإن تلك الوراثة (إن كان لنا أن نسمّيها وراثته) زائفةٌ لا حقيقة لها. ثم لما ذا لم يتم دعم الوراثة المُتعددة الحقيقية ما دامت لها فوائدٌ هائلةٌ إلى هذه الدرجة؟!.

لقد تم دعم الوراثة المتعددة في **إيداع** بالفعل، و بمقارنته بدعم الواجهات في اللغات الأخرى يمكننا أن نرى أنهما استهلكا مجهوداً قريباً لدعمهما، كما أن لهما تقريباً ذات الدرجة من الإضافات لقواعد اللغات اللاتي يُضَمَّان إليهن!

فلمْ بُذِل ذلك الجهد و خُصِّصَت تلك القواعد لدعم الواجهات و لم يُبذَلْ و يُخَصَّصَ لدعم الوراثة المتعددة؟!

السبب تجدونه عند أصحاب توجه دعم الواجهات بدلاً من الوراثة المتعددة.

• تُعْطَى الصِّنْفِ أَوْ الْأَصْنَافِ خِصَائِصَ مَا كَانَتْ لِمَتَلِكْهَا:

قَرَأْتُ هَذَا السَّبَبَ فِي مَرْجِعِ إِنْقِلِيزِي يَتَحَدَّثُ عَنِ لُغَةِ الـJava. و في الحقيقة فإنني لا أفهم حتى الآن ما يقصده المؤلف من هذا الكلام، فإن كان يقصد ميزة إعطاء الهياكل و الأصناف ميزتي الوراثة و الوراثة المتعددة على الترتيب فقد رددنا على هذا الزعم.

• إِجْبَارُ الصِّنْفِ الْجَدِيدِ الْبَانِي لَهَا عَلَى بِنَاءِ كُلِّ الْإِجْرَاءَاتِ الَّتِي فِيهَا:

و أنا أتساءل: لِمَ هَذَا التَّكْلِفُ مَا دَامَتْ هُنَاكَ الْأَصْنَافُ الْمُجَرَّدَةُ الَّتِي تُؤَدِّي هَذَا الدَّورَ بِكُلِّ قُوَّةٍ وَ فَعَالِيَةٍ؟!

• تُمَثِّلُ دَوْرَ الْبَرُوتوكُولِ بَيْنَ الْأَصْنَافِ الْمَخْتَلِفَةِ. وَ لِذَلِكَ فَبَعْضُ لُغَاتِ الْبَرْمَجَةِ الْكَائِنِيَّةِ

التَّوَجُّهُ تَسْمِيهَا (البروتوكول) بَدَلًا مِنْ الْوَاجِهَةِ:

يُمْكِنُ لِلْأَصْنَافِ الْمُجَرَّدَةِ أَنْ تَقُومَ بِنَفْسِ الدَّورِ بِمُنْتَهَى الْكِفَاءَةِ، وَ أَكْثَرُ أَنْ هَذَا الْقَوْلُ يَشْتَرِكُ مَعَ سَابِقِهِ فِي أَنْ مُصَمِّمِي اللُّغَاتِ الَّتِي تَدْعُمُ الْوَاجِهَاتِ إِخْتَارُوا الْحُلَّ الْأَصْعَبَ لِمُشْكِلَةٍ كَانَتْ يُمْكِنُ حَلُّهَا بِمُنْتَهَى الْبَسَاطَةِ!

الهياكل `structs`:

نبذة بسيطة:

هي مكوّنات في الـ C و الـ C++ و الـ C# تُشبه إلى حد كبير للغاية الأصناف، حيث تُمثّل تجميعاً منطقيّة من المتغيّرات، بالإضافة إلى الإجراءات التي تعمل على تلك المتغيّرات. و على الرغم من هذا فهي تُخالف الأصناف في بعض الخصائص كما سأوضّح فيما يلي.

أري ألا يُسمَح بها للأسباب التالية:

- تُدخل المُبرمج في تفا صيل يُفترض أن تظل مقصورةً على المُبرمج الباني للمُترجم و لا تُتاح لمُستخدم اللغة؛ فالسبب الرئيس لدعمها في لغات مثل الـ C و الـ C++ أنها تُعطى المُبرمج القدرة على تحديد ما إذا كان يُريد إنشاء الكائن الجديد في منطقة المُكدّس `stack` في الذاكرة أم في منطقة الـ `heap`، و كما قلنا أن هذا يُعدّ تعديلاً واضحاً على السلطات التي يُفترض أن تكون للمُترجم أو المُفسّر مع نظام التشغيل.

فمُترجمٌ مهندسٌ جيداً و مُبرمجٌ على مُستوى عالٍ من الحرفية و الإتقان يُغني عن كل هذه الزيادات في القواعد، و يُغني المُبرمج عن التدخل في أمورٍ يشتكى أغلبية المبرمجين من اضطرابهم إلى خوض غمارها رغم عدم رغبتهم في أغلبية الوقت إن لم يكن كله في فعل هذا. و القول بأن هناك من المُبرمجين من تُسعد هذه الخاصية في اللغات التي تدعمها تجعلني أقترح على هذا المُبرمج العودة للبرمجة بلغات التجميع، فالأخيرة ستُعطيه من القوة الأكثر و سيجد مُبرمجين يقولون أن البرمجة بها تُسعدهم أكثر من سواها !

أما ما أظنه السبب وراء وجود هذه الصفات في اللغات التي تُوجد بها: فهو أن ظروف خروج لغة مثل الـ C كان هو السبب الرئيس وراء كل ما حدث؛ حيث أن هذه اللغة قد أنتجها دينيس ريتشي Dennis Ritchie حين احتاج إلى لغة تُعطيه كل القوة التي تُعطيهها له لغات التجميع (أو على الأقل أغلبها). في حين تكون لها بساطة لغات المستوى الأعلى و **محموليّتها** و قابلية أكوادها العيش لفترةٍ أطول و الإستفادة المتجددة من التقنيات الجديدة في برمجة المترجمات، و ذلك ليكتب بها نظام التشغيل الجديد الذي يرغب في بنائه، و هو نظام اليونكس `unix`.

و كان أمامه خياران لأسلوب التعامل مع القوة التي تُتيحها له اللغة: فالأول **سلوب الأول** و هو الأكثر مباشرةً كان أن يضع كل تلك القوة في يديه هو على هيئة أدوات واضحة يُستخدمها كما يُستخدم المتغيّرات و الإجراءات. أما **الأسلوب الثاني** فهو أن يضع كل تلك القوة في بنية

المُترجم ذاته ثم يُكَيِّفُهَا لِتُنْتِجَ لَهُ المُنْتِجَاتِ الأَفْضَلِ تَقْنِيَا، بِخَوَارِزِمَاتٍ ذَكِيَّةٍ يُمَكِّنُهَا أَنْ تَسْتَغْلَ الفرصَ التحسينية في الكود و تقوم بجعل الكود أفضل أداءً و استغلالاً للموارد.

ولأن دِينِسَ كان مُعتاداً على البرمجة باستخدام الأسلوب الأول (و هو توافر كل الأدوات بين يديه)؛ لأنه عا صر كل الأجيال الأولى للغات البرمجة منذ لغة الآلة و حتى اللغات عالية المستوى مُروراً بلغات التجميع، و لأن الأسلوب الأول هو الأسلوب الأسرع في البناء؛ فقد وقع اختياره عليه.

و لما كان مُعا صروه لهم نفس ظروفه التي دفعته إلى اختيار ذلك الأسلوب فقد وافقوه أو على الأقل وافقه معظمهم على اختياره هذا.

و لهذا كله فإن الغلطة لم تكن هنا، بل كانت في اللحظة التي كان بإمكان المُبرمجين من الأجيال التالية الإستغناء عن هذا الأسلوب و العودة إلى الأسلوب الأفضل و لكنهم لم يفعلوا، و كان على رأس هذه اللحظات لحظة إنتاج لغة الـ C++؛ فقد أدخلت هذه اللغة إمكانيات البرمجة كائنية المنحى إلى لغة الـ C و لكنها احتفظت في نفس الوقت بكل عيوب الأخيرة التي نجمت عن الأسلوب الأسرع في التحكم في القوة البرمجية، و لم يُحاول مُطَوِّرُهَا التخلص من تلك العيوب (بل توسّع أكثر من ذلك في وضع مهام كان يُفترض على المُترجم القيام بها على كاهل المُبرمج العادي!).

- تزيد من قواعد اللغة بدون فائدة، بل إن الشبه بين قواعدها و قواعد الأصناف يجعلها مصدر بلبلة للمُبرمج، و أكثر للمُتعلِّمين الجدد للغة التي تدعمها؛ فعلى المُبرمج أن يتذكر أن الهيكل لا يرث و لا يُورث على العكس من الصنف الذي يرث و يُورث، و عليه أن يضع في ذهنه أن الهيكل و الصنف يشتركان فقط في الوراثة من الواجهات، هذا بالطبع بجانب تذكر أن الهيكل حينما يُنشأ فإنه يُنشأ في منطقة المُكَدَّس في الذاكرة بينما يُوضَع الصنف في منطقة الـ heap. و غيرهن من الإختلافات التي لا سبيل للتخلص منها إلا بالتخلص من التفرقة بين الصنف و الهيكل من الأصل.

فإذا ما ضمنا الهيكل و الواجهة إلى بعضهما البعض لوجدنا أنهما ليسا إلا تنوعاً على الصنف، و لكن مع بعض الإختلافات؛ في الوظيفة كما في حالة الواجهة، أو في الخصائص الداخلية كما في حالة الهيكل.

و قد كان بإمكان مُطَوِّرِي لغات البرمجة الجديدة إن أرادوا دمج هذه الأنواع الثلاثة من المُكوّنات في مُكوّن واحدٍ أشمل و أعم (كما فعلت في إبداع) فيتفادوا كل ذلك الرّخم من القواعد المُربكة، و كذا كل البلبلة التي تنتج عن التشابه بين تلك القواعد نظرياً و اختلاف التعامل العملي معها.

و لكن ما حدث هو ما نُؤَه إليه البروفيسور C. A. R. HOARE في ورقة علمية مَاتَعَة تحت عنوان (ملاحظات حول تصميم لغة البرمجة) قَدَمَهَا عام 1973، حيث قال فيها ما ترجمته (ببعض التصرف):

بعض مُصَمِّمِي لغات البرمجة استبدلوا (النمذجة) بـ(البساطة) في تصميم اللغات، و يعنون بـ(النمذجة) أن المُبرمج إذا لم يستطع فهم اللغة كاملةً فإنه يكفيه فهم جزءٍ منها فقط.

ربما يكون هذا معقولاً بالنسبة للبرامج التي تعمل كما قصد منها المُبرمج، و لكن إذا كان برنامج لا يعمل و كان (قَدَرًا) يستخدم صفةً في اللغة لا يعلمها المُبرمج: فإنه سيقع في مشاكل صعبة. فإذا ما كان مُوقَّفًا نوعاً ما فإن البرنامج المُصَحَّح سيتحقق من خطئه و لكنه لن يكون قادراً على فهم الرسالة التشخيصية، خلاف ذلك فإنه لن يكون ذى فائدة.

و إذا ما أُضيف إلى تعقيد لغة البرمجة تعقيدُ بنائها، و تعقيدُ بيئة عملها و كذا تعقيدُ معايير العمل لاستخدامها: فإنه ليس من المُفاجئ أنه عند مجابهة مهمة برمجية معقدة فإن العديد من المُبرمجين لا يستطيعون النجاة.

فقد اعتقد مُطَوِّرُوا مُعْظَم اللغات الحالية أنه بإمكان المُبرمج إتقان جزءٍ فقط من اللغة و عدم الحاجة إلى إتقانها كلها، و لكن الرد عليهم ما قاله البروفيسور HOARE و أدرجناه فيما سبق.

جُمْلَةُ goto :

من أشد الأمور التي تُحَيِّرُنِي فِي عَالَمِ تَصْمِيمِ لُغَاتِ الْبِرْمَجَةِ أَنْ هُنَاكَ أُمُورًا مَعِينَةً ظَلَّ كَثِيرٌ مِنَ الْعُلَمَاءِ وَالْمُحْتَرِفِينَ مِنَ الْمُبْرِمَجِينَ يُنَوِّهُونَ إِلَى خَطَرِهَا وَيُحَدِّثُونَ مِنْهَا بِشَدَّةٍ، ثُمَّ رَغْمَ هَذَا يَقَعُ فِيهَا مُصَدِّمُ اللُّغَاتِ بِكُلِّ بَسَاطَةٍ! . وَمِنْ أَشَدِّ الْأَمْثَلَةِ وَضُوحًا مَسْأَلَةُ جُمْلَةِ goto الَّتِي اخْتَصَّهَا بَعْضُ الْعُلَمَاءِ وَهُوَ **edsger w.dijkstra** بِرِسَالَةٍ عِلْمِيَّةٍ خَاصَّةٍ لِلتَّحْذِيرِ مِنْهَا وَمِنْ خَطَرِهَا، وَهِيَ الرِّسَالَةُ الْمُسَمَّاةُ (جُمْلَةُ goto تُعْتَبَرُ ضَارَةً). بَلْ وَقَدْ اشْتَهَرَ بَيْنَ الْمُبْرِمَجِينَ مُصْطَلَحُ (مَعْرُونَةُ الْأَسْبَاقِيَّتِي) لِيَصْفُوا بِهِ الْبِرَامِجَ الَّتِي تُفْرِطُ فِي اسْتِخْدَامِ جُمْلَةِ goto فَتَتَحَوَّلُ إِلَى عَجِينَةٍ مِنَ الْأَوَامِرِ الَّتِي يَصْعَبُ فَهْمُ عَمَلِهَا حَتَّى عَلَى مُبْرِمَجِهَا إِنْ غَابَ عَنْهَا لِحِظَاتٍ !

وَقِيلَ عَنْ فَائِدَةِ عَدَمِ وُجُودِ جُمْلَةِ (goto) فِي الْبِرَامِجِ⁸ :

وَإِذَا كَانَ بِالْإِمْكَانِ قِرَاءَةُ الْبِرَامِجِ مِنْ أَعْلَى إِلَى أَسْفَلٍ بَدُونِ الْحَاجَةِ إِلَى الْقَفْزِ عِبْرَ الْجُمْلِ الْمَتَنَاثِرَةِ نَتِيجَةً لاسْتِخْدَامِ جُمْلِ (goto) الْمُرِيكَةِ: فَإِنَّ التَّرْكِيزَ سَيَكُونُ أَكْبَرَ لِأَنَّهُ سَيَنْصَبُ عِنْدئذٍ عَلَى جِزْءِ الْكُودِ الْحَالِي، مِمَّا يُعْطَى الْفُرْصَةَ لاسْتِغْلَالِ الذَّاكِرَةِ اللَّحْظِيَّةِ جَيِّدًا وَعَدَمِ تَكْلِيفِهَا مَتَابَعَةَ أَجْزَاءِ أُخْرَى مِنَ الْكُودِ بِطَرِيقَةٍ تُشَبِّهُهَا وَتُهْدِرُ قَوَاهَا، وَهُوَ نَفْسُ السَّبَبِ الَّذِي يَجْعَلُ الْمُبْرِمَجِينَ الَّذِينَ لَا يَسْتِخْدَمُونَ هَذِهِ الْجُمْلَةَ فِي بِرَامِجِهِمْ أَقْلَ عَرْضَةً لِلْوُقُوعِ فِي الْخَطَأِ، وَنَتِيجَةً لِهَذَا فَقَدْ نَجَحَتْ الْبِرْمَجَةُ الْهَيْكَلِيَّةُ فِي السَّبْعِينَاتِ، حَيْثُ أَنَّهَا تَقُومُ عَلَى الْبِرْمَجَةِ فَقَطْ بِاسْتِخْدَامِ الْجُمْلِ الشَّرْطِيَّةِ وَالْجُمْلِ التَّكْرَارِيَّةِ مِمَّا يَجْعَلُ الْبِرَامِجَ أَقْلَ أَخْطَاءًا.

وَفِي الْوَرَقَةِ الْعِلْمِيَّةِ الْمَعْنُونَةِ (جُمْلَةُ goto تُعْتَبَرُ ضَارَةً) نَوَّهَ الْمَوْلَفُ (**edsger w.dijkstra**) إِلَى أَنَّ جُمْلَةَ goto دَعْوَةٌ صَرِيحَةٌ لِلإِرتْبَاكِ وَالْبَلْبَلَةِ، حَيْثُ قَالَ مَا تَرْجَمْتَهُ (بِبَعْضِ التَّصْرُفِ):

الاسْتِخْدَامُ الْجَامِحُ لِعِبَارَةِ Goto لَهُ عَوَاقِبٌ فُورِيَّةٌ مِنْ حَيْثُ أَنَّهُ يُصْبِحُ فِي غَايَةِ الصَّعُوبَةِ إِيجَادَ مَجْمُوعَةٍ إِحْدَاثِيَّاتٍ ذَاتِ مَعْنَى يُمَكِّنُنَا مِنْ خِلَالِهَا تَوْصِيفَ الْعَمَلِيَّةِ. عَادَةً فَإِنَّ النَّاسَ يَأْخُذُونَ فِي اعْتِبَارِهِمْ كَذَلِكَ قِيمَ مَتَغْيِرَاتٍ مُخْتَارَةً بِعِنَايَةٍ، لَكِنْ هَذَا خَارِجٌ عَنِ نِطَاقِ التَّسَاوُلِ لِأَنَّهُ لَا يُمَكِّنُ فَهْمَ مَعْنَى هَذِهِ الْقِيمِ إِلَّا بِالنَّظَرِ إِلَى التَّقَدُّمِ ! مَعَ جُمْلَةِ Goto يُمَكِّنُ لِلْمَرْءِ تَوْصِيفَ الْبِرَامِجِ بِشَكْلِ فَرِيدٍ بَعْدًا دِ يُحْصَى عَدَدُ الْأَحْدَاثِ الَّتِي نُقِذَتْ مِنْذُ بَدَايَةِ الْبِرَامِجِ. الصَّعُوبَةُ أَنَّهُ فِي إِحْدَاثِيٍّ كَهَذَا عَلَى الرَّغْمِ مِنْ تَفَرُّدِهِ أَنَّهُ لَيْسَ مَفِيدًا عَلَى الإِطْلَاقِ. فِي نِظَامِ إِحْدَاثِيٍّ كَهَذَا يُصْبِحُ مِنَ الْأُمُورِ الْمَعْقَدَةِ لِلْغَايَةِ تَعْرِيفِ كُلِّ تَلْكَ النِّقَاطِ لِلتَّقَدُّمِ.

8 من كتاب (software engineering) الفصل الثاني (human factors in)

(software engineering)، أما المؤلف فلم أستطع الحصول علي اسمه بكل أسف !؛

فالنسخة التي عندي ليس فيها غلاف الكتاب، كما أن البحث عنه علي الشبكة لم يجد نفعاً.

و قال كذلك ما ترجمته:

جملة **Goto** كما هو الوضع حالياً بدائيةً للغاية، و تُعدُّ بصورةٍ كبيرةٍ دعوةً لتحويل البرنامج إلى فوضى.

و من ثم كان من الطبيعي أن ينتهي به الأمر إلى النتيجة التي ذكرها في بداية رسالته، حيث قال ما ترجمته:

لعديدٍ من السنوات كنتُ مُعتاداً على ملاحظة أن كفاءة المبرمجين تُعتبر دالةً عكسيةً في كثافة عبارة **Goto** في البرامج التي يُنتجونها، و في توقيتٍ أحدثٍ اكتشفتُ لم لا استخدام **Goto** هذه التأثيرات الكارثية، و اقتنعتُ بأن جملة **Goto** يجب أن تُمحى من لغات برمجة المستوى الأعلى (أي: كل شيء فيماعدًا شفرة الآلة المبسوطة).

و على الرغم من أن هذه الرسالة نُشرت في مارس من عام 1968 م، و صَدَرَ بعدها مؤلِّفاتٌ أخرى تُحدِّر مثلها من المشاكل التي تُسببها جملة **goto**: إلا أن مُصمِّمي لغات البرمجة ظلوا بعد كل تلك السنين على عنادهم، و أبقوها في اللغات الجديدة التي كان يجب فيها التخلص من كل ما عُلم خطره على المبرمج. و لا يُمكنني بالضبط أن أحدد السبب الذي يجعل لغات برمجةٍ كائنية التوجه إلى أقصى الحدود مثل الـ **#C** تُبقى على جملةٍ أمريةٍ من العهود الأقدم للغات البرمجة، على الرغم من أن وجودها إضافةً إلى ما يُسببه من ارتباكٍ منطقي في البرامج فهو أيضاً يُسبب زيادةً في قواعد اللغة التي كانت تحتاج أصلاً إلى التقليل من كميتها المتورمة لا زيادتها!

المؤشرات pointers:

نبذة بسيطة:

تُوجَدُ في عائلة الـ C إلا الـ java و في العديد من اللغات الأخرى مثل الـ object pascal و غيرهن، و هي مُتغيّراتٌ من نوع خاص؛ حيث أن القِيم التي تَحْمِلُها داخلها تُمَثَّلُ عناوينا في الذاكرة إما لمُتغيّرٍ آخر أو لبداية جُمْلَةٍ نصيَّةٍ أو حتى لإجراءٍ من الإجراءات، و حينما يتم التعامل معها فإن المُبرمج يتعامل مباشرة مع الذاكرة و مُحتوياتها دون أي وسيطٍ منطقي. و التعامل بالمؤشرات يكاد يكون إجبارياً في الـ C/C++ اللاتي يُدخِلنهما في أي أمرٍ مهما كان صغيراً (و أقول هذا عن تجربة شخصية).

أري ألا يُسَمَحُ بها للأسباب التالية:

- تُدخِلُ المُبرمج في تفاصيل يُفترَضُ أن تظل مقصورةً على المُبرمج الباني للمُترجم لا مُستخدم اللغة، حيث أن إدارة الذاكرة و إعادة غير المُستخدم منها مرةً أخرى إلى حوزة نظام التشغيل من صميم عمل المُترجم و مُبرمجه، و الذي يجب ألا يتخلى عن هذه المهمة لصالح المُبرمج العادي أبداً.

و لكن كما قلنا قبلاً عند مناقشة أسباب التخلي عن استخدام الهياكل في لغة **إبداع** فإن المُبرمجين القدامى اختاروا أسرع الحلول للحصول على القوة القصوى في برامجهم، و قد كان بكل أسفٍ الحل الأقدر (و القذارة هنا بالطبع معنويةً بحتة)؛ فقد كَلَّفَ هذا الحل المُبرمجين الجدد الكثير؛ فبدلاً من الإنطلاق نحو مزيدٍ من التركيز على المهام التي تُستخدَم لغات البرمجة لحلها فقد أصبح المُبرمج يقضى كثيراً من وقته أمام اللغة مُحاولاً الهروب من الحُفر المُغطاة و الألغام المُخفية؛ للحصول على حقه الطبيعي في برنامجٍ خالٍ من العيوب و يعمل بكفاءةٍ و قوة!

و أصبح من المُعتاد بالنسبة للدارسين الجدد تبيهم على النقاط التي يجب دراستها بمُنتهى الحرص و التركيز، و أصبح من الأكثر اعتياداً طمأنة هؤلاء الجدد حينما يُخطئون بسبب تلك الألغام بالقول لهم أن "هذا يحدث حتى للمُبرمجين الكبار"، و كأن هذا شيءٌ يبعث على الطمأنينة لا الذعر!

و لى هنا أن أدرج قولاً قرأته في مجلة (مجتمع لينوكس العربي) للأخ (محمد نجم):

عندما اتذكر بداياتي في البرمجة
يسعني القول أنها كانت مأساة حقيقة فانا شققت طريقي في البرمجة بلغات معقدة كالسي وبعض الجافا ولي أسبابي فلم اكن اعلم شيء آخر سواهم و كنت صغير السن وقتها وغير متفرغ أيضاً . خليط الإحساس والإحباط ثم الغباء ثم الغضب ثم تنهي برنامج بسيط(من ٤٠ سطر او اكثر) وتجد أخطاء وتعجز عن إيجاد البعض الآخر هذا بالفعل كضيل يجعلني ارجع عن البرمجة ككل وربما ابدأ في البحث عن مجال آخر بعيداً عن هذا التعب(وجع الرأس).

و لتكتمل الصورة فإنى أدرج قوله عن البرمجة بلغاتٍ أخرى:

أنا بدأت بروبي حقيقة بعد العديد من المشاكل (وجع الراس) مع السي والجافا والشيل سكربتينج (الذي كان اسهالم لكنه لايفي بالغرض) والان انا مستمتع ببساطتها وقوتها معا اذا اردت برنامج ويب لدي Rails واذا اردت برنامج رسومي توجد العديد من المكتبات تفي بالغرض

و الأمر واضحٌ تماما، و يعكس بالفعل ما يحدث لكثير من المُبتدئين فى البرمجة و يُسبب خسارة كبيرةً فى نشء المُبرمجين و المُقبلين على دراسة هذا العلم.

• لا تكتفى بكل تلك البلبلة التى تُسببها بل تزيد من قواعد اللغة بشكل كبير، فتُصبح عملية تعلم اللغة أعقد و أطول و أكثر إرهاقا عما كانت ستُصبح عليه لو لم تكن تلك المُكوّنات موجودة فيها. و لا تكتفى بإرهاق مُستخدم اللغة فقط، بل تتجاوز ذلك إلى عقاب المُصمّم لها بطريقةٍ لو علم بها مُستخدم اللغة لانتشى شماتة و نام قرير العين؛ فالْمُصمّم الذى يجعل لغته البرمجية كبيرةً إلى حدٍ مُفرع ليس له أن يشتكى من صعوبة التأليف بين كل ذلك الزخم من القواعد، و ليس له كذلك أن يشتكى من صعوبة صنع المُترجم أو المُفسّر القوى لها.

أما تطوير اللغة و الإرتقاء بها تدريجياً فأمرٌ سيكون بالنسبة له ضرباً من ضروب الخيال العلمى؛ فإذا كان حال اللغة و هى فى أول إصداراتها على تلك الحال المُزرية التى و صفنا: فماذا سيكون حالها عند إضافة الجديد إليها؟!

إضافةً إلى ذلك فحتى لو استطاع إضافة الجديد إليها فأى مُبرمج مجنونٍ ذلك الذى سيتحمل كل تلك الإضافات و يتماشى معها؟!

أما المساكين ممن فرَضتْ عليهم ظروفهم الدراسية تعلّم تلك اللغة البرمجية فلن يجدوا مفرّاً بعد ذلك من تكملة دراستها كاملة و احترافها؛ فعلى الرغم منهم سيكونون قد قطعوا شوطاً لا بأس به فى تعلم اللغة و لن تكون خسارتهم عند تكملة دراستها أكبر من خسارتهم عند تركها نهائياً و تعلم لغةٍ جديدةٍ تماما، بل على الأرجح ستكون تلك الخسائر أقل و أهون و هو الأمر الوحيد الذى يجعلهم يَتمون مشوار التعلم هكذا (إن أتموه).

• مصدرٌ لا ينفد للأخطاء النحوية و التنفيذية و المنطقية فى البرامج التى تُستخدم فيها، و بالتالى فإن المُبرمج للمُترجم حينما يلقى بجزءٍ من مسئولية إدارة الذاكرة على عاتق المستخدم لتلك اللغة فإنه لا يُقلّل الأعباء التى تقع عليه كما يظن، بل إنه يزيدا أكثر و أكثر لحاجته إلى اكتشاف أنواع أكثر من الأخطاء و المساعدة على تنقيح البرامج حتى تخرج سليمة النحو و المنطق.

و كون المؤشرات مصدراً للتصعب في البرمجة لا يخفى على أحد؛ فهي أدوات مُربكة للغاية بطبيعتها، و عن تجربة شخصية أقول أن وجود المؤشرات في البرامج التي أكتبها في عملي كانت و لا تزال تزيد من صعوبة التكويد إلى الحد الأقصى مما يكلفنا من الوقت الغالي الكثير.

و أُضربُ مثلاً لهذا الموقف بالكود التالي لدالة تتعامل مع مكتبة الـ **OpenCV** حيث تأخذ منها متغيراً من نوع **(* IplImage)** حيث يُمثل هذا النوع التمثيل القياسي للصورة في مكتبة الـ **OpenCV** لتُعيد ما يحتويه من بيانات الصورة على هيئة بايتات متراصة يُشير إلى أولها مؤشر من نوع **(double*)**. و المُفترض أنها مسألة بسيطة للغاية و مباشرة و لكن استخدام المؤشرات فيها حوّل الأمر إلى مأساة.

حيث أن التمثيل **IplImage** يستخدم مُشيراً من نوع **(* Char)** ليُشير إلى أول بايت في بيانات الصورة التي يحتوي عليها، و يُسمى هذا المؤشر بـ **imageData**، و من ثَمَّ للحصول على هذه البيانات و التعامل معها بأي شكلٍ من الأشكال يجب على المُبرمج التعامل مع المؤشرات بكل ما يحمله هذا من معاني.

و بالنظر إلى كود الدالة **cvtImg2Dta** و المقارنة بين كل ذلك الزخم من الحسابات المُستخدم للوصول إلى بيانات الصورة في حالتنا هذه، و بين ما كان الوضع ليُصبح عليه إذا كانت البيانات موجودة على هيئة مصفوفة ثنائية الأبعاد **two dimensional array**: لوجدنا أن الوضع كان ليُصبح أسهل بكثير و أقل تعقيداً مما أجبرتنا عليه المؤشرات بالفعل.

```
void CvtImg2Dta(IplImage *img, double *slice)
{
    for( int y=0; y< img->height; y++ )// taking care of rows indexing
    {
        for( int x=0; x< img->width; x++ )// taking care of columns indexing
        {
            slice[img->nChannels*x+y*img->width*img->nChannels] =
                (double)img->imageData[img->nChannels*x+y*img->widthStep]/255.0;
            if(img->nChannels==3)
            {
                slice[(3*x)+(3*y*img->width) +1] =
                    (double)img->imageData[(3*x)+(y*img->widthStep) +1]/255.0;
                slice[(3*x)+(3*y*img->width) +2] =
                    (double)img->imageData[(3*x)+(y*img->widthStep) +2]/255.0;
            }
        }
    }
}
```

فبدلاً من كتابة:

((3*x) + (3*y*img->width) +1)

و ما شابهها من حساباتٍ كنا سنكتب مُمَيَّزَاتِ العنصر `element indexes` داخل المصفوفة مباشرةً و ينتهى الأمر.

و ليس الأمر بالبسيط كما قد يبدو؛ فقد تسببت كل هذه الحسابات المتداخلة المُربِكة فى تعطيل عملنا لفترةٍ لا بأس بها؛ بسبب وجود خطأ ما فى عمل الدالة احترنا فى سببه، حتى اكتشفنا أننا نسينا مُعاملاً مُعيَّناً من ضمن الحسابات المُربِكة تسبب عدم وجوده إلى عدم الإشارة بصورةٍ صحيحةٍ إلى العناصر المرغوب فيها!، و لا تسأل عن الحسرة و الغيظ اللذين أحسنا بهما حينما اكتشفنا أن ذلك الخطأ التافه هو السبب فى كل الريبة و الحيرة اللتان غمرتانا، و كل التأخر الذى حدث لعملنا.

لذلك أقول عن جدٍ و عن تجربةٍ شخصيةٍ واقعيةٍ أن الإصرار على وجود المؤشّرات فى لغات البرمجة الجديدة لا يخدم العلم البرمجى فى شئ، و لا يُقدّم الخير للأجيال الجديدة من المبرمجين الذين يُفترض بهم السير بهذا العلم الجليل إلى الأمام، و هم بطبيعة الأمر لن يفعلوا هذا إذا ما أصرّت الأجيال الأقدم من المبرمجين على فرض تركّتهم الثقيلة من الأدوات العتيقة و الأفكار المُربِكة التى ولى زمنها و لم يعد من المناسب بحالٍ من الأحوال أن تبقى فى الواقع المعيش للعلم البرمجى.

الإجراءات :procedures

نبذة بسيطة:

الإجراءات التي أقصدها هنا ليست هي الإجراءات التي في إبداع بالطبع، بل هي مكوّنات قديمة للغاية حيث يرجع ميلادها إلى زمن الأيام الأولى للبرمجة عالية المستوى، وهي تشبه الدوال **functions** تماماً في الفكرة الأساسية من حيث أنها تمثّل عباراتٍ من الكود تُنفَّذ بالترتيب من أعليّ إليّ أسفل في كل مرة يتم فيها استدعاء ذلك الإجراء، وتختلف عن الدوال في أن الدوال تُعيد قيمة واحدة أو قيمةً متعددةً أما الإجراءات فلا تُعيد أي قيمة.

ومن اللغات التي تدعم الإجراءات في الوقت الحالي: **Fortran** و **Ada** و **Pascal** وهي كلها لغاتٌ من الأجيال القديمة، وفي معظم اللغات البرمجية الحديثة تم الإستغناء عن الإجراءات بشكلٍ نهائي حيث لم يعد لها أي معنى (اللهم إلا في لغاتٍ مُعيّنة مثل **object pascal**)، وإنني أوافق مُصممي اللغات الحديثة في قرار استبعاد الإجراءات كمكوّن في لغات البرمجة الحديثة للأسباب التالية الذكر.

أري ألا يُسمَح بها للأسباب التالية:

هناك سببٌ وحيدٌ لعدم دعم الإجراءات في لغات البرمجة الحديثة هو أوضح من الشمس في مُنتصف السماء، أن الإجراء ما هو إلا دالةٌ لا تُعيد أي قيمة، ولا يختلف عن الدوال العادية في أي أمرٍ آخر سوى هذا، وبالتالي فأى لغة برمجة تدعم الدوال (أي كل لغات البرمجة الحديثة بطبيعة الحال) تدعم الإجراءات ضمناً لكن مع عدم ذكر كلمة إجراء بشكلٍ صريح.

ولكن على الرغم من أن هذا الأمر بدهيّ تماماً ومن غير المنطقي تطبيق غيره فإن هناك لغات برمجةٍ سارت عكس التيار تماماً و دعمت الدوال والإجراءات كلٌّ بشكلٍ مُنفصلٍ عن الآخر!، وهنا سأتكلم عن لغة ال **object Pascal** بعينها لمجرد ضرب المثل، ففيها يكون تعريف الإجراء كما في المثال التالي:

```

Procedure foo (var a: byte);
Begin
  Writeln (a);
  a := 10;
  Writeln (a);
End;

```


حيث أن الكلمات (Procedure) و (End) (Begin;) هي كلماتٌ محجوزةٌ ثابتةٌ يجب كتابتها في كل مرة، بينما الكلمة (foo) هي اسم الإجراء الذي يُمكن تغييره بحرية، و ما بين القوسين (التاليين للاسم) هي قائمةٌ بمُعَامِلَاتِ الإجراء التي ستُمرَّرُ له للعمل عليها، و يُوضَع الكود المُكوِّن لجسم الإجراء بين كلمتي (Begin) و (End;) المفتاحيتين.

و لو نظرنا إلى المثال التالي لتعريف دالةٍ من الدوال في الـ Pascal object:

```
Function w (a, b: integer): integer;
Begin
Read (a, b) ;
W: = a+b;
End;
```

- لوجدنا أن نفس القواعد السابقة تُطبَّق هنا أيضاً فيما عدا فرقتين اثنتين لا ثالث لهما:
- كتابة الكلمة المفتاحية (Function) بدلاً من الكلمة (Procedure) في بداية جُملة التعريف كما هو موضحٌ بالمثالين. وهذا الفارق هزلئى كما هو واضحٌ و لا يعنى أى شئ.
 - كتابة نوع الخَرْج الخاص بالدالة بعد أقواس المُدخَلات المُمرَّرة لها، و هو من النوع (Integer) في المثال السابق.

و فيما عدا هذين الفارقين فالأمر واحدٌ لا خلاف فيه.

و ما دام الأصل واحدٌ، و ما دامت الدالة تحتوى الإجراءَ داخلها بشكلٍ منطقي: فإن الأمر المثير للعجب و السخرية هو أن تدعم لغة برمجة المُكوِّن في نفس الوقت!؛ فلا هي فعلت مثل اللغات الأقدم التي لم يكن هناك ما يُسمَّى بالدوال في وقتها و بالتالى فعلت الصواب حينما دَعمت الإجراءات؛ لأنها دَعمت مُكوِّناً قوياً بمعايير عصرها، و لا هي فعلت مثل لغات البرمجة الحديثة التي اكتفت بالمُكوِّن الأعم و الأشمل و الأقوى الذي هو الإجراء؛ و بالتالى فعلت الصواب بتقليل قواعد اللغة و تخفيف اللبس و البلبلة من على كاهل المُبرمجين.

و السبب الوحيد الذي يُمكن أن يُقدِّمه مُصمِّمو تلك اللغات أنهم أرادوا من وراء ذلك المُحافظة علي التوافق العكسي backward compatibility؛ لأن تلك اللغات الحديثة هي في حقيقة الأمر تطويرٌ للغاتٍ أقدم، فمثلاً تُعد لغة الـ Pascal object تطويراً للغة الـ Pascal التي تحتوي بدورها علي الإجراءات، و لما كانت برمجياتٌ كثيرةٌ مكتوبةٌ بالـ Pascal فإنهم اعتبروا أنه من الخسارة ترك كل

تلك البرمجيات و عدم دعم ترجمتها بشكلٍ طبعيٍّ في لغتهم الجديدة، و نفس الحال تكرر مع لغتي الـ ++C/C.

و لكنني حتي في هذه الحالة أرفضُ ضم الإجراءات؛ لرفضني لمبدأ التوافق العكسي أصلاً؛ فلغات البرمجة الحديثة يجب أن تنفصل عن القديم لتُقَدِّم ما يتوافق مع أحوال الواقع البرمجي الحالي.

و يُمكن التغلب علي عدم التوافق بين برامج اللغتين القديمة و الحديثة عن طريق بناء برنامج لتحويل الأكواد بينهما آلياً، و هو الأمر الموجود بالفعل للغاتٍ تختلف بشدةٍ عن بعضها البعض (مثل الـ java و الـ objective C) فبالتالي سيكون أسهل بمراحلٍ عند عمله للتحويل بين لغتين شديديتي التشابه مثل الـ object Pascal و الـ Pascal.

الْوَسَائِطُ delegates:

نُبذةٌ بَسِيطةٌ:

الوسائط (و هنا سأركز علي الوسائط كما توجد في لغة الـ C#) هي عبارة عن مكوّناتٍ من نوع خاص تسمح بتسجيل استدعاءاتٍ عامّةٍ لدوالٍ لها مُعاملاتٍ واحدةٌ و أنواع خَرَجٍ واحدةٌ كذلك، بحيث أنه يُمكن ربط الوسيط الواحد مع أكثر من دالة، و حينما يتم استدعاء ذلك الوسيط (كما تُستدعى أية دالةٍ أخرى) يقوم الوسيط باستدعاء الدوال المرتبطة به، و يُمرّر لها المُعاملات التي تم تمريرها له عند استدعائه. و لا يَمنع التشابه بين مؤشّرات الدوال **functions pointers** و الوسائط من كون الوسائط نوعاً جديداً؛ حيث أن مؤشّرات الدوال لا يُمكنها أن تقوم ببناء أكثر من دالةٍ في نفس الوقت و تقتصر في عملها على استدعاء دالةٍ واحدة. و بالتالي تكون الفائدة المرجوة من ورائها صغيرة نوعاً ما، أما الوسائط فيمكن لها (على الأخص تلك التي يُمكنها أن تشير إلى دوالٍ لا تُعيد أي قيمة) أن تشير إلى أكثر من دالة.

أري ألا يُسمح بها للأسباب التالية:

الوسائط (من حيث الفكرة الأساسية) مكوّناتٌ في غاية القوة و الفائدة، و لكن المُشكلة أن الشكل الذي قدّمته لغة الـ C# للوسائط كان في غاية الهُزال و الضعف.

و ضعف صياغات الوسائط في الـ C# يتجلى في الصور التالية:

- تقييد المُعاملات التي ستمرّر للدوال التي سيستدعيها الوسيط بأن تكون هي نفسها المُعاملات التي تمرّر للوسيط ذاته، و بالتالي لا يُمكنني اختيار تمرير مُعاملاتٍ مُختلفةٍ حتي لو كانت هناك حاجةٌ ما سّةً لذلك. و هذا العيب أدى إلى عيبٍ آخر هو:
- يجب أن تكون مُعاملات الدالة التي تُضم إلى الوسيط مُوافقةً لمُعاملات الوسيط، أي أن الوسيط يُشير إلى نوع واحدٍ من الدوال (هذا بالإضافة إلى التقييد بنوع الخرج المُؤخّذ بين الوسيط و إجراءاته). و هكذا فقد الوسيط ميزة العموميّة التي تُتيح له أن يُشير إلى أي نوعٍ من أنواع الدوال مهما كانت مُعاملاتها و خَرَجها.

المهمات :tasks

نبذةٌ بسيطةٌ:

المهمة **task** هي مُكوّنٌ برمجيٌّ قائمٌ بذاته في لغة **Ada**، وهي في الأصل برنامجٌ يُشبه البرنامج الأساسي في كل شيءٍ إلا أنه يتم تنفيذه على التوازي مع البرنامج الرئيس. أي أنها تقوم بنفس عمل خيوط التنفيذ **execution threads** في اللغات الأخرى و لكن بشكلٍ أكثر اندماجا داخل اللغة نفسها وليس بشكلٍ منفصلٍ عنها في مكتبة اللغة.

أري ألا يُسمَح بها للأسباب التالية:

في الحقيقة فإنني لا أعترض على المهمات كمكوّنٍ أساسيٍّ في لغات البرمجة الحقيقية ولا أجادل في هذا على الإطلاق؛ فأنا من أكثر الناس اقتناعاً بأهمية دعم العمليات المتوازية التي تُركّز التقنيات الحديثة في تصميم المُعالجات على دعمها بتعدد أنوية تلك المُعالجات في الشريحة الواحدة وغيرها من التقنيات الأخرى.

ولكنني أرى فقط أنه من الأفضل عدم وضع هذا المُكوّن في صلب اللغة مباشرةً و وضعه في المكتبة القياسية الخاصة بها قدر الإمكان. فمثلاً في **java** و **C#** يُمكنك إذا ما احتجت إلي خيوط التنفيذ المُتعددة أن تمد يدك إلى المكتبة و تُعرّف خيط تنفيذٍ جديد، و تربط الخيط الجديد إما بصنفٍ جديدٍ بينى **implements** واجهة **interface** مُعيّنة كما في **java** (و هو نموذجٌ مُتمزّتٌ بعض الشيء في وجهة نظري). أو تربطه باسم دالةٍ ما كما في **C#** (و أظن أنه النموذج الأفضل والأقوى (الأسرع).

ولكن يظل الأمر سيان بالنسبة للمُستخدم النهائي للغة؛ حيث ستظل المهمات أو خيوط التنفيذ مُتوافرةً للإستخدام بين يديه ليستخدمها كما يشاء و متى يشاء مهما اختلف اسمها أو مكان اعتمادها في اللغة، إنما الفرق الوحيد بين اعتماد المهمات كمكوّنٍ لغويٍّ مُباشرٍ أو كمكوّنٍ في المكتبة يظهر مع المُبرمج الباني للمُترجم؛ حيث أنه لو وُضعت المهمات في صلب اللغة فسيكون على مُبرمج المُترجم وضع ذلك في حسابانه في مرحلة المسح **scan** و التحقق المنطقي **semantic analyzing** و إنتاج الكود الوسيط، بينما سيكون الأمر أخفّ لو كانت المهمات جزءاً من المكتبة البرمجية لا اللغة نفسها؛ فعلي الأقل سيختفي جزء المسح و التحقق المنطقي و يندمج ضمن قواعدٍ أخرى مبنيةً بالفعل.

و سيزداد الفرق معه لو كان قد قرر بناء الماسح و المُحقق المنطقي بيديه لا باستخدام البرامج التي تُنتجها باستخدام صياغات **BNF** أو **EBNF** للغة؛ ففي هذه الحالة يكون زيادة مُكوّنٍ أو تعبيرٍ برمجيٍّ واحدٍ عبئاً على التصميم و البناء لمُكوّنات المُترجم الأولى التي تعتمد على نحو اللغة بشكلٍ تامٍ في عملها (أعني الماسح **scanner** و السابر **parser**).

و نظراً لأننى قد قررتُ بالفعل عدم استخدام الأدوات المُساعدة فى بناء الواجهة الأمامية لمُترجم **إيداع** (علي الأقل في الإصدارات الأولي منه؛ و ذلك للعديد من الأسباب الهامة للغاية) فإننى من هذه النوعية من المُبرمجين الذين يهتمهم بالفعل تقليل مُكوّنات اللغة إلى الحد الأدنى (بالطبع إذا لم يكن هذا ضاراً) لتخفيف الضغط التصميمي و البنائى الواقع على كواهلهم.

المُغْلَفَاتِ properties:

نُبذةٌ بسيطةٌ:

هى نفسها الـ `properties` فى الـ `C#` و التى يُترجمها الكل بـ (الخصائص) و لكننى أُسميها بالمُغْلَفَاتِ لسبب سبلى ذكره، وهى نوعٌ خاصٌ من الإجراءات؛ حيث أنها تأخذ شكلاً شبيهاً لها فى الوظيفة و التعريف، و لكنها تُستخدَم كما تُستخدَم المتغيرات؛ فهى حلقةٌ وسيطةٌ بينهما. و وظيفتها الأساسية هى الإحاطة المنطقية بمتغير أو مجموعةٍ من المتغيرات لتمثيلها أمام العالم الخارجى و تتعامل معه بالنيابة عنها.

و مثالٌ على إشارها و استخدامها فى الـ `C#` ما يلى:

```

class Data {
    FileStream s;

    public string FileName {
        set {
            s = new FileStream(value, FileMode.Create);
        }
        get {
            return s.Name;
        }
    }
}

Data d = new Data();
d.FileName = "myFile.txt"; // invokes set("myFile.txt")
string s = d.FileName;    // invokes get()

```

أرى ألا يُسمح بها للأسباب التالية:

يُمكن باستخدام الإجراءات فعل نفس الشئ و ذلك كما فى المثال التالى بإبداع:

ص = العملاء ()

العملاء (ص + 1)

إجراء (رقم س) العملاء:

س = عدد.العملاء

إجراء العملاء (رقم س):

عدد.العملاء = س

و لكن مع الحصول على الفوائد التالية:

- الحفاظ على نحو اللغة أبسط و أصغر مما كان سيُصبح عليه لو تم ضم المُغْلَقَات إلى اللغة.
 - القدرة على تحديد مُستويات وُصولٍ مُختلفةٍ لكلٍ من: إجراء التغيير فى القيمة، و إجراء الحصول على القيمة.
- مثل البرنامج التالى:

إجراء عام (رقم س) العملاء:

س = عددالعملاء

إجراء خاص العملاء (رقم س):

عددالعملاء = س

المُفَهَّرَات indexers:

نبذة بسيطة:

هي مُكوّنٌ أظنه جديداً تماماً تقدّمه الـ **C#** كطريقةٍ جديدةٍ للتعامل مع الأصناف، و شرح مايكرو سوفت يقول أن المُفَهَّرَات تُمكننا من التعامل مع الصنّف كأنه مصفوفة **array**، لكنني أرى أن هذا الكلام قد يبدو صحيحاً في الوهلة الأولى و لكن إمعان النظر فيه يُؤدّي إلى الإيمان بخطئه التام، لأن المُفَهَّرَات ما هي إلا نوعٌ من المُغلّفات **properties** غير المُسمّاة و التي تُستدعى عن طريق اسم صنّفها مباشرة. و بالنظر إلى المثال التالي على المُفَهَّرَات سنجد أن التعامل مع الصنّف كان مُماثلاً للتعامل مع المُغلّفات لا مع المصفوفات:

```
class MonthlySales {
    int[] product1 = new int[12];
    int[] product2 = new int[12];
    ...
    public int this[int i] { // set method omitted => read-only
        get { return product1[i-1] + product2[i-1]; }
    }

    public int this[string month] { // overloaded read-only indexer
        get {
            switch (month) {
                case "Jan": return product1[0] + product2[0];
                case "Feb": return product1[1] + product2[1];
                ...
            }
        }
    }
}

MonthlySales sales = new MonthlySales();
...
Console.WriteLine(sales[1] + sales["Feb"]);
```

و التشابه الوحيد بين التعامل مع المصفوفات و الصنّف هو استخدام قوسى المصفوفات و هما [] فقط لا غير، بينما تشبه المُفَهَّرَات الخصائص فيما يلي:

- 1- بها شقٌّ يُنقَد عند قراءة القيمة التي يُعبّر عنها العنصر (قسم **get**)،
- 2- بها شقٌّ يُنقَد عند إسناد قيمة للعنصر (قسم **set**)،
- 3- يُمكن وضع الشّقين (أيهما أو كلاهما) فى الكود ليُنْتِج مُفَهَّرَسٌ قابلٌ للقراءة و الكتابة، أو للقراءة فقط، أو للكتابة فقط.

إذاً فالتشابه بين المُغْلَفَاتِ و المُفْهَرِّسَاتِ ليس مُجَرَّدَ تشابهٍ بل هو تطابق، و الفرقان الوحيدان: أن المُغْلَفَاتِ تُسْتَدْعَى باسمها بينما يتم استدعاء المُفْهَرِّسِ باستخدام اسم الكائن نفسه، و كذلك السماح بتمرير مُعَامِلَاتٍ لِلْمُفْهَرِّسِ للعمل عليها بينما لا يُسَمَّحُ بهذا للخصائص العادية، إذاً فالْمُفْهَرِّسَاتِ ما هي إلا مُغْلَفَاتٌ غير مُسَمَّاةٍ تقترب من الإجراءات أكثر من المُغْلَفَاتِ العادية.

أرى ألا يُسَمَّحُ بها للأسباب التالية:

ليس للمُفْهَرِّسَاتِ أي فائدةٍ تتفرد بها و لا يُمكن القيام بها بشكلٍ آخرٍ ربما يكون بنفس البساطة أو أكثر بساطة، فبإمكاننا إظهار إجراءٍ يَفْعَلُ كل ما تَفْعَلُ المُفْهَرِّسَاتِ بدون إِتْخَامِ اللُّغَةِ بِمُكَوِّنٍ جديدٍ يزيد من قواعدها بشكلٍ كبير (و هي لا تحتل من الأصل)، و يُحْمِلُ المبرمجين المُتَعَلِّمِينَ لها عبء إدراك الفارق بين المُكَوِّنَاتِ المُخْتَلِفَةِ المُتَشَابِهَةِ.

و بصدق: فإن هناك نقطةً أخرى تزيدني غيظاً من الـ **C#** هي الإحساس بأن مُصمِّمِيهَا يُرِيدُونَ جَعْلَ النَّاسِ يُحْسِنُونَ أن الـ **C#** تُقَدِّمُ تقنياتٍ جديدةٍ تختلف عن كل ما سبق (و خاصة عن الـ **java** التي تُمَثِّلُ لمايكروسوفت نفس ما يُمَثِّلُهُ ابن الصُّرَّةِ للزوجة!) حتى و إن كان مضمون تلك التقنيات يُشَكِّلُ مُشْكَلَةً لا حلاً، و عبثاً إضافياً لا تخفيفاً و دعماً!

نموذج الـ Python للأصناف و الدوال:

على الرغم من أن لغة الـ Python تُمَثَّلُ بالنسبة لى لغةً من أفضل اللغات (بل أفضلها) من حيث شكل البرنامج المكتوب بها، إلا أنها تحتوي على قراراتٍ تصميميةٍ أجد أنه من الواجب الرد عليها لأنها تُمَثَّلُ قراراتٍ سيئةٍ بما فيه الكفاية لمن كان له قناعاتٍ تصميميةٍ كقناعاتي.

و من أَوْضَحَ هذه القرارات الغريبة هي نظرة اللغة إلى الأصناف و الدوال؛ حيث أن هذه النظرة الغريبة ترى في الأصناف ما يُشبه الدوال و ترى في الدوال ما يُشبه الأصناف، فنرى أن الأصناف في الـ Python عبارة عن حاويةٍ لمجموعةٍ من الأوامر المُتتَابِعَةِ التي تُنفَّذُ بِشَكْلِ مُتتَالٍ واحداً تلو الآخر، و أن المُكوِّنات الداخلية للأصناف مثل الدوال و المُشَيِّدات و خلافهما ما هي إلا عباراتٍ تنفيذيةٍ بدورها، و يعنى هذا أنه يُمكننى أن أقوم بكتابة عباراتٍ إسناد قيم و استدعاءاتٍ لإجراءاتٍ داخل صلب الصنف و ليس داخل مُشَيِّداته أو دواله فقط.

و مثال على هذا:

```
Class class1:
    def func1():
        # code of func1
    A = func1() + 5
    def func2():
        # code of func2
```

حيث يتم في البداية تعريف دالةٍ تُسمَّى `func1`. ثم تم استخدامها لإسناد قيمةٍ إلى متغيرٍ يُسمَّى `A`. ثم تلى ذلك تعريف دالةٍ أخرى تسمى `func2`.

و تكتمل الصورة حينما نعلم أنه يُمكننا في الـ Python تعريف دالةٍ داخل دالةٍ أخرى، و استخدام الدالة الداخلية عن طريق كتابة اسم الدالة الحاوية لها و إتباعها بقوسى دالةٍ نكتب فيهما المُعاملات المُمَرَّرة للدالة الداخلية. و تُعرَفُ الدوال المُحتوية بالإسم `closures`.

و مثال على هذا:

```
def func1(a):
    def func2(b):
        return a+b
    return a
```

و بالنظر إلى صفات الأصناف و الدوال في الـ Python سنرى أن لكليهما الصفات التالية:

- اسم الدالة و اسم أى كائنٍ من الصنّف هما مؤشّران يُشيران للدالة أو الكائن، أى أنه يُمكن التغيير فى هوية الدالة أو الكائن التي/الذى يُشير إليه/ها الاسم.

مثل البرنامج التالي:

```
from math import *

def print_calc(f):
    print "log(%s)=%s, exp(%s)=%s" % (f, log(f), f, exp(f))

print_calc(1.0)
log, exp = exp, log      # evil code! swap the objects the names refer to
print_calc(1.0)
```

حيث أنه فى الأمر قبل الأخير جعلنا الاسم (log) يُشير إلى كود الدالة (exp)، بينما جعلنا الاسم (exp) يُشير إلى كود الدالة (log)!. و النتيجة التالية لاستدعاء الاسمين قبل التبديل و بعده تدل على أثر التبديل و نجاحه:

```
log(1.0)=0.0, exp(1.0)=2.71828182846
log(1.0)=2.71828182846, exp(1.0)=0.0
```

- كلاهما يُمثّل كتلةً من الأوامر التي تُنفَّذ بشكلٍ تّابعي.
- كلاهما له القدرة على احتواء تعريفاتٍ لأصنافٍ داخله، أى أنه يُمكن للدالة أن تحتوى على تعريف صنّفٍ من الأصناف داخلها، لتعود و تستخدمه بصورةٍ طبيعيةٍ كما يحدث فى حالة الأصناف المُتفرّعة فى باقى لغات البرمجة.

مثال:

```
def func1(a):
    Class class1:
        def func2():
            # code of func1
        A = func2() + 5
        def func3():
            # code of func2
```

- كلاهما له القدرة على احتواء دوالٍ داخليةٍ يتم استدعاؤها عند الحاجة إليها عن طريق اسم المُكوّن الحاوي لها، مع الفارق في أن الدالة تحتوي داخلها دالةً داخليةً واحدةً فقط بينما للـصِّنْفِ تعديد الدوال، وكذلك فإن الدوال التي في الصِّنْفِ يتم استدعاؤها باسمها بينما تُستدعى الدالة الداخلية باسم المُتغير الذي استدعى دالتها الحاوية.

مثال 1:

```
def func1(a):
    def func2(b):
        return a+b
    return a
```

```
a = func1(5)
```

```
b = a(10)
```

مثال 2:

```
Class class1:
    def func1():
        # code of func1
    A = func1() + 5
    def func2():
        # code of func2
```

```
Class1 inst
```

```
inst.func1()
```

و من الصفات السابقة نرى أن هناك تقريباً لا معنى له بين صفات مُكوّنين برمجيين يجب إبقاء كل منهما مُنفصلاً عن الآخر، و عدم إثارة البلبلة التي لا داعي لها بجعل كل منهما يحمل شيئاً من صفات الآخر بشكل لا يُحقق أية أهدافٍ في حقيقة الأمر.

فالدوال هي في الأصل كتلة من الأوامر التي تُنفَّذ بشكلٍ تتابعي بحيث يتم استدعاؤها عند الحاجة إليها؛ فيُوقَّر وجودها تكراراً الأكواد و مساحة البرنامج النصية، و يزيد القدرة على اكتشاف الأخطاء و السرعة في ذلك و غيرهن من العوامل الأخرى، أما الأصناف ففي الأصل لم تنشأ إلا لتكون حاويةً تجمع بين مجموعة من المُتغيرات و مجموعة من الدوال التي تعمل عليها، و عن طريق الإستنساخ من الأصل يتم الإستفادة

من كود الصنف أقوى استفادةً عن طريق عزل المُحتَوِيَّاتِ الداخليَّةِ للكائنات عن المُبرِمْجِ النهائي و غيرها من الفوائد.

فما معنى وجود دالةٍ تحوى دالةً أخرى داخلها، أو صِنْفٌ يحوى أوامراً تُنفَّذُ بشكلٍ مُتسلسلٍ و كأنها كتلة أوامرٍ فى صلب دالةٍ من الدوال؟!!

و ما الفائدة التى ستعود علينا من ذلك المزج و التقريب الغريبين؟،

و أى ضررٍ سيعود علينا عند التفريق بينهما كما تفعل بقية اللغات؟

الدَّامِجَاتُ unions:

نُبذةٌ بسيطةٌ:

الـ unions في الـ C و الـ C++ و ما حذا حذوهما من اللغات: هي مُكوّناتُ الفكرة الرئيسة من ورائها هي إتاحة الفرصة لاستخدام نفس الحَيِّزِ من الذاكرة بأي نوع من الأنواع المختلفة، يعنى أن الدَّامِجِ عبارة عن حاوية تُشبه الصِّنْفِ من حيث أنه يتم داخله تعريف مُتغيّرات، و لكنه يختلف عنه في أن الصِّنْفِ يُعتبر كل مُتغيرٍ داخلي مُنفصلاً عن المُتغيّرات الأخرى من حيث حَيِّزِ الذاكرة المحجوز له، بينما فى الدَّامِجِ يتم إعطاء كل تلك المُتغيّرات مساحةً مُشتركةً فى الذاكرة تبدأ من نفس النقطة و تنتهى حسب مساحة أكبر المُتغيّرات حجماً، و لكن لا يُمكن لأي مُتغيرٍ أن يتعامل إلا مع جزءٍ من تلك الذاكرة المُشتركة يتناسب مع نوع ذلك المُتغير.

يعنى لو أننا كتبنا ما يلى:

```
// declaring_a_union.cpp
union DATATYPE // Declare union type
{
    char  ch;
    int   i;
    long  l;
    float f;
    double d;
} var1; // Optional declaration of union variable

int main()
{
}
```

فإن معنى هذا أننا نعرِّف مُتغيراً يُسمى var1 من النوع DATATYPE يُمكننا التعامل معه على أساس:

أنه من النوع char بالإسم ch.

أو النوع int بالإسم i.

أو النوع long بالإسم l.

أو النوع float بالإسم f،

أو النوع double بالإسم d.

و القيمة التى نتعامل معها كل مرة هى نفس القيمة و لكن مع اختلاف المساحة التى نتعامل معها منها.

أرى ألا يُسمَحُ بها للأسباب التالية:

أرى أن الأسباب التى يسوقها الشارحون للبرمجة بلغتى الـ C و الـ C++ فى كتبهم لوجود الدَّامِجَاتِ و مدى فائدتها لا تُسمِن و لا تُغنى من جوعٍ أمام التحقيق و التدقيق؛ فما وجدتُ منها غير:

- أن الدَّامِجَاتِ تُوفَّرُ فِي اسْتِخْدَامِ الذَّاكِرَةِ حِينَمَا نُرِيدُ التَّعَامُلَ مَعَ قِيَمَةٍ مَا بِأَكْثَرَ مِنْ أُسْلُوبٍ، فَبِدَلَا مِنْ: تَعْرِيفِ مُتَغَيِّرَاتٍ مُخْتَلِفَةِ الْأَنْوَاعِ بِشَكْلِ مُنْفَصِلٍ، وَ أَقْوَمِ كُلِّ مَرَّةٍ حِينَمَا أَحْتَاجُ إِلَى التَّعَامُلِ مَعَ الْقِيَمَةِ عَلَى أَنَّهَا مِنْ نَوْعٍ مَعِينٍ بَوْضَعِهَا فِي الْمُتَغَيِّرِ الَّذِي يَنْتَمِي إِلَى ذَلِكَ النَّوْعِ، وَ مِنْ ثَمَّ أَقْوَمُ بِإِجْرَاءِ الْعَمَلِيَّاتِ عَلَى ذَلِكَ الْمُتَغَيِّرِ، أَقُولُ: بَدَلًا مِنْ كُلِّ هَذَا فَإِنِّي أَقْوَمُ مُبَاشَرَةً بِوَضْعِ تَعْرِيفٍ لِدَامِجٍ يَدْمِجُ بَيْنَ تِلْكَ الْأَنْوَاعِ فِي مَكَانٍ وَاحِدٍ لِأَتَمَكَّنَ مِنَ التَّعَامُلِ مَعَ الْقِيَمَةِ الْمُخَزَّنَةِ فِي دَاخِلِهِ بِأَيِّ شَكْلِ أُرِيدُهُ.

و هذا سببٌ لا أظنه موجوداً اليوم؛ فهذا القول كان صحيحاً حينما كانت الأجهزة التي تتم البرمجة لها لا تحتوي من الذاكرة إلا على 64 كيلوبايت أو ما يُشابهها، يَسْتَغَلُّ نِظَامُ التَّشْغِيلِ الْقَلِيلَ مِنَ الْكَيْلُوبَايْتَاتِ مِنْهَا وَ تَسْتَغَلُّ الْبَرَامِجُ الْعَادِيَّةُ الْبَاقِي، وَ هَذِهِ مَوْا صِفَاتٌ قَرِيبَةٌ مِنْ مَوْا صِفَاتِ أَجْهَزَةِ الـ PDP11 الَّتِي كَانَتْ أَوَائِلَ بَرَامِجِ الـ C مَكْتُوبَةً لَهَا، وَ مَا عَا صَرَهَا مِنَ الْأَجْهَزَةِ. وَ لَكِنَّا الْآنَ نَعِيشُ فِي زَمَنِ مُخْتَلِفٍ لَيْسَ مِنَ الْمُمْكِنِ فِيهِ أَنْ تُوَهَّمَنِي أَنَّكَ حِينَمَا تَكْتُبُ بَرَامِجًا كَالسَّابِقِ سَوْفَ تُوفَّرُ عَلَيَّ الْكَثِيرُ مِنَ الذَّاكِرَةِ الْمُهْدَرَةِ، بَلِ الْعَكْسُ أَنَّكَ سَتُحْصَلُ عَلَى بَرَامِجٍ غَرِيبٍ سَيُّ التَّكْوِينِ مُقَابِلِ الْحَصُولِ عَلَى بَضْعَةِ بَايْتَاتٍ لَا تَسَاوِي أَيَّ شَيْءٍ بِالْمُقَارَنَةِ مَعَ أَقْلِ الْإِمْكَانِيَّاتِ فِي سَعَاتِ ذَوَاكِرِ الْحَاسُوبِ الْمَوْجُودَةِ الْيَوْمَ.

إِذَا فَهَذِهِ الْفَائِدَةُ الَّتِي كَانَتْ مَوْجُودَةً لِلدَّامِجَاتِ قَدْ انْتَهَتْ إِلَى غَيْرِ رَجْعَةٍ وَ كَانَتْ مِنَ الْمُحْتَمِّمِ عَلَى مُصَمِّمِي لُغَاتِ الْبَرْمَجَةِ التَّنْبَهُ إِلَى هَذَا وَ إِعْيَادِ ذَلِكَ الْمَكُونِ الزَّائِدِ عَنِ اللُّغَاتِ الْجَدِيدَةِ، لَا ضَمَّهُ إِلَيْهَا كَمَا حَدَثَ مَعَ الـ C++ الَّتِي سَارَعَتْ إِلَى ضَمِّهِ إِلَيْهَا وَ كَأَنَّهُ كَنْزٌ ثَمِينٌ.

- الدَّامِجَاتُ أَكْثَرَ كِفَاءَةً مِنْ اسْتِدْعَاءِ الْإِجْرَاءَاتِ الَّتِي تُحَوَّلُ بَيْنَ أَنْوَاعِ الْمُتَغَيِّرَاتِ، أَيُّ أَنَّهُ فِي كُلِّ مَرَّةٍ نَحْتَاجُ فِيهَا إِلَى تَحْوِيلِ قِيَمَةٍ مَا مِنْ نَوْعٍ إِلَى نَوْعٍ آخَرَ: فَبِدَلَا مِنْ اسْتِدْعَاءِ إِجْرَاءَاتٍ لِلتَّحْوِيلِ بَيْنَ أَنْوَاعِ الْمُتَغَيِّرَاتِ فَإِنَّا نَسْتُخْدِمُ الْمُتَغَيِّرَاتِ الْمَوْجُودَةَ فِي الدَّامِجِ مُبَاشَرَةً، وَ نَسْتُخْدِمُ مِنْهَا ذَلِكَ الَّذِي لَهُ النَّوْعُ الَّذِي نُرْغَبُ فِي التَّعَامُلِ مَعَ الْقِيَمَةِ عَلَى أُسَاسِهِ.

وَ هَذَا قَوْلٌ آخَرٌ أَرَاهُ غَيْرَ مَنْطِقِي فِي عَالَمِ الْيَوْمِ؛ حَيْثُ أَنَّ مِثْلَ هَذِهِ التَّوْفِيرَاتِ الْغَرِيبَةِ لَا تُقَدِّمُ تِلْكَ النَّتَائِجَ الْخَارِقَةَ الَّتِي تُغْرِي بِبَدَلِ الْجُهْدِ وَ التَّضْحِيَةِ لِلْحَصُولِ عَلَيْهَا، فَمَا الَّذِي سَيُضِرُّ الْبَرَامِجَ إِذَا مَا اسْتَدْعَى إِجْرَاءَ تَحْوِيلٍ عَشْرَ مَرَّاتٍ أَوْ أَكْثَرَ؟!، وَ مَا التَّأخِيرُ الَّذِي سَيَحْدُثُ عِنْدَئِذٍ؟!، وَ كَيْفَ سَيُؤَثِّرُ ذَلِكَ التَّأخِيرُ عَلَى زَمَنِ تَنْفِيذِ الْبَرَامِجِ بِصُورَةٍ وَ لَوْ طَفِيفَةً؟!

أين الإبداع فيها؟

هذا الجزء مُخَصَّصٌ لشرح الأسباب التي جعلتني أصمِّم **إداع** بالشكل الذي ظهرت عليه في النهاية، فرغم أن هناك شرحاً لقواعد التصميم التي أقتنع بها في فصل سابق، و أن هناك شرحاً للمكونات التي رُفِضَتْ ضَمَّهَا لِ**إِدَاع** و الأسباب التي دفعته إلي ذلك: إلا أن الحاجة لهذا الجزء ما سَتُهُ لِلغَايَةِ؛ و ذلك لأنه حتى عند ذكر الأصول الفكرية لأي منهج من المناهج فسيحدثُ اختلافٌ في تفسير تلك الأصول و تنزيلها علي الواقع، و الأمر الوحيد الذي سيمنعُ مثل ذلك التشتت التفسيري هو ضرب الأمثلة العملية لذلك المنهج حينما يتم تنزيله علي الواقع العملي بواسطة من وَّضَع تلك الأصول.

و قد التزمتُ هنا قَدْرَ الإمكان بالبساطةِ الشديدة حتى لا يتضخم حجم الكتاب بشكل لا يُطاق، و الإكثار من الأمثلة العملية حتى تُصبح الأمور أكثر سهولةً من حيث الاستيعاب. و كذا فقد أدرجتُ أموراً ربما يري القاري أنها ليست بالأهمية التي يجب معها الإهتمام بها و أفراد مكانٍ لها، و لكنني فعلتُ هذا كتوضيح لمبدأ أن لغة **إِدَاع** ليست مجرد تجميعاً من المكونات وُضِعَتْ مع بعضها البعض لتصبح لغة برمجة عادية، إنما هي لغةٌ صُمِّمَتْ بِمُنْتَهَى العناية حتى تتوافق كل مكوناتها مع بعضها البعض و تنسجم في تناغم جميل. و ربما لا تظهر الحكمة من وراء جعل جزءٍ ما من اللغة علي الشكل الذي هو عليه بشكلٍ مَبَاشِرٍ، و لكن معرفة الكثير من التفاصيل الأخرى تجعل الذهن يتنبه إلي ما فاته و يُدرك التوليفة التي جعلت الأمور تُصبح هكذا.

أسباب جعل لغة إبداع لغة تفسيرية بجانب وجود مُترجم لها:

- للإستفادة من فلسفة الـ **Java** التي تنص على كتابة الكود مرّةً واحدةً و تشغيله في أى مكان، و بالتالى: ننتج البرنامج و نترجمه مرّةً واحدة، و نرفع نسخةً واحدةً على الشبكة، و يُحمّل المُستخدم النهائي نسخةً وحيدة، إذا رغبتنا في هذا.
- تُمكننا من نقل البرنامج بين أنظمة التشغيل بحريةً و ذلك بدون أن يضارّ المُبرمج في أى شيء؛ إذ أنه يتعامل مع البيئة التشغيلية التي سيتم إعطاؤها القدرة على أنظمة تشغيلٍ أخرى، و كذلك تحديثها للتعامل مع الإصدارات الجديدة من أنظمة التشغيل التي تم تحديثها، و هذا يعنى حياةً أطول للبرامج و محموليةً عاليةً جداً بدون أدنى تعبٍ من جانب المُبرمج.
- إذا ما أردنا بعد ذلك تطوير لغة برمجةٍ عربيةٍ أخرى مُتخصّصةً في مجالٍ مُعيّن (كلغةٍ للتعبيرات و التعاملات الإحصائية على سبيل المثال) فإن إنتاجها سيكون أسهل؛ و ذلك لأننا لن نحتاج حينئذٍ إلا إلى صُنع الواجهة الأمامية من مُترجم تلك اللغة و مُحوّلٍ إلى الصيغة الوسطى الخاصة بالآلة الوهمية لإبداع فقط، دون تجسّم عناء صُنع مُترجمٍ كاملٍ.
- هذا هو التطور الطبيعي في علوم الحاسب البرمجية؛ التي ظلت رُويداً رُويداً تُبعد البرامج عن التعامل المباشر مع العتاد الصلب و تُسند هذا الدور إلى النواة **kernel**، و لم يبق في الوقت الحالى إلا إبعاد البرامج عن التعامل المُباشر مع المُعالج و هو الأمر الذي يتحقق باستخدام المُفسّرات. و هكذا نكون في مأمنٍ من أغلاط المُبرمجين في التعامل مع المُعالج في التطبيقات التي لا يهتمهم فيها كيفية تنفيذ أوامرهم في المُعالج ما دامت تُنفذ بكفاءةٍ و قوة.
- في حالة استخدام بيئات التشغيل: فإن التطورات التقنية الجديدة لن تكون مُطبّقةً فقط على البرامج الجديدة كما في حالة الترجمة منذ البداية، بل ستطبق بأثر رجعيٍّ على البرامج القديمة التي تعمل على الإصدارات الحديثة من بيئة التشغيل.

فوائد وجود نظام أنواع ثابت **static typing**:

- إكتشاف أخطاء استخدام الحَاويات (أي المُتغيّرات و الثوابت و كائنات الاصناف و كائنات الألقاب) في زمن الترجمة **compile time** بدلاً من زمن التنفيذ **run time**.
- تخفيف العبء عن زمن التنفيذ؛ و ذلك لأن إختبارات الأنواع و صحة استخدامها تتم في هذه الحالة في زمن الترجمة مرّةً واحدة، بينما في حالة نظام الأنواع المُتغيّر **dynamic typing** تتم الإختبارات في زمن التنفيذ في كل مرّة يتم فيها استخدام الحَاوية.

- إمكانية استغلال معلومات الأنواع (المُتوافرة في زمن الترجمة) للمزيد من تحسين استغلال الذاكرة و/أو المُعالج و أخذ ذلك في الحسبان أثناء زمن الترجمة.

فائدة وُجوب تعريف الحاويات يدويا :

- يُعد نوعاً من التوثيق الواضح التلقائي.
- يُكسب الكود مقروئيةً أعلى.
- تخفيف العبء من علي كاهل المُترجم و مُبرمجه؛ حيث أنه أبعد عنه عبء استنتاج الأنواع المُختلفة بنفسه، مع العلم أنه لم يحمل عنه عبء استنتاج مساحات الحاويات المُختلفة التي يُمكن أن تمتلك مساحةً مُتغيرةً حسب ظروف الإستعمال (مثل المُتغيرات الرقمية).
- في كثير من الأحيان في اللغات التي تُتيح للمُبرمج استخدام المُتغيرات بدون تعريفها يدوياً سُنضطر لمحاكاة عملية التعريف، مثل الموقف الافتراضي التالي:

س = { } \ لتعريف جدول فارغ لاستخدامه داخل تعبير بينما /
بينما < شرط، منطقي > :
< كود، بينما >

حيث احتجنا في هذه الحالة إلي تعريف جدول قبل الدخول إلي صلب تعبير بينما؛ لأننا لو قمنا بتعريفه داخل التعبير فسيتم اعتباره جدولاً مُختلفاً في كل لفة ! و بالتالي يأخذ قيمة غير التي نتوقع له أن يأخذها.

و لذلك نُضطر إلي إسناد قيمة جدول فارغ إلي الإسم (س) حتي يعلم مُترجم اللغة أن (س) ما هو إلا جدول نرغب في تعريفه. و مثل هذا المواقف يُمكن رؤيته في لغات مثل python و ruby و غيرهن من لغات التنويع المُتغير.

- يُمكن جعل التعريف و الإسناد في سطرٍ واحدٍ كمحاكاةٍ للُّغات الديناميكية و بالتالي لن يأخذ التعريف أسطراً زائدة و لن يجعل الكود أطول، كما يُمكن جعل التعريفات في أعلى الأوامر التنفيذية كتوثيق واضح كما يحدث إجباراً في لغة object pascal.
- في اللغات التي لا يتم فيها تعريف الحاويات داخل الأصناف في أسطرٍ مُنفردة (مثل لغتي الـ python و ruby) فإنه لمعرفة الحاويات داخل صنفٍ ما: سُنضطر لقراءة أكواد إجراءاته كلها، أو ستعتمد علي برنامج يُساعدك علي هذا، أو ستحتاج لتوثيقاتٍ خارجية تُخبرك عمّا تُريد، أما في اللغات التي تُجبرك علي التعريف اليدوي فلن تحتاج لفعل أي من هذا لأن الكود في أعلى الصنف يُغنيك عنه.

فوائد عدم تعبير الأنواع البسيطة عن الإحجام :

- يُؤدِّي إلى قِلَّةِ عدد الكلمات المفتاحية في اللغة. فالأنواع `int` و `uint` و `float` و `double` و `short` و `decimal` وغيرهن الكثير تم اختصارها في النوع رقم، بينما تم اختصار الأنواع `String` و `char` في النوع نص.
- عدم التقييد بمفاهيم الذاكرة الحالية (مثل الإعتماد علي عدد البايتات في تحديد المساحة) و بالتالي الانفصال التام عن تكوين الآلة التي يعمل عليها البرنامج. إلا عند استخدام أكواد التجميع التي ستتغير إذا تغيرت البنية العتادية بما لا يُؤثِّر علي نحو اللغة نفسها، و كذا بما لا يُؤثِّر علي توحيد معني كل قاعدة في اللغة مهما اختلف نظام التشغيل أو المُعالج اللذين تعمل عليهما برامجهما.
- تقليل عدد القواعد في اللغة من حيث:
 - عدم الحاجة لقواعد التخصيص `casting`.
 - عدم الحاجة لوجود الأكواد العامة `generic codes` بكل ما تحويه من صعوبات و تعقيدات جَمَّة، فالتعميم موجودٌ عملياً في الأكواد؛ فليس هناك تفریق بين الأحجام و لم يُجعل كلُّ حجم نوعاً مُختلفاً عن الآخر.
- التناغم مع التفكير البشري الطبيعي عالي المُستوي الذي لا يهتم إلي حدٍ كبير بالتحديد الفائق للأمر، و بالتالي زيادة سرعة التعلُّم للغة. مع ترك أمر هذه التحديدات للمُترجم أو المُفسِّر لكي يهتم هو بها حسب حالة الآلة التي يعمل عليها و ظروف العمل.
- زيادة المقروئية.
- زيادة المكتوبية.

سبب جعل تعريفات الألقاب و الإجراءات و الأصناف في نهاية الملفات الرئيسة. و بعد الاستخدام الفعلي لها:

- لو كان العكس هو ما حدث (أي تعريف تلك المُكوّنات قبل استخدامها): فسيكون مؤلماً للمُبرمج في كل مرّة يفتح فيها برنامجاً أن يُقلِّب الكود بحثاً عن بداية أوامر البرنامج التي تُوجد أسفل التعريفات، لكن في حالة **إبداع** فبمُجرّد فتح الملف الرئيس يجد المُبرمج بداية الأوامر دون جهدٍ منه.
 - و في لغات `python` يلجأ المُبرمجون لعمل إجراء له في العادة الإسم `main` ثم يستدعونه في بداية الأوامر بالأسفل، و في داخل هذا الإجراء يضعون أوامر المسار الرئيس للبرنامج كاملة كما هو الحال مثلاً في لغة `C`. أي أنهم يُحاكُون أسلوب اللغات الوظيفية و الإجرائية للتغلب علي تلك المُشكلة.
- مثال:

```
def main :
```

```
# main program's code
```

```
def func1 :
    # function1's code
    .
    .
    .
main()
```

- بينما حَلَّلْنَاهَا بِشَكْلِ أَسْطٍ وَ لَا يَتَعَارَضُ مَعَ بَقِيَةِ الْأَسْبَابِ التَّالِيَةِ.
- هَذَا يَتَّفِقُ مَعَ الطَّرِيقَةِ التَّلَقَّائِيَةِ الَّتِي يَعْمَلُ بِهَا الْمُبْرِمِجُ؛ حَيْثُ عِنْدَمَا يَقُومُ الْمُبْرِمِجُ بِكِتَابَةِ خَوَارِزْمٍ لِحَلِّ مُشْكِلَةٍ مَا فَإِنَّهُ يَقُومُ بِـ:
 - كِتَابَةِ الْخَطُوطِ الْعَرِيضَةِ الْعَامَّةِ، وَ يَسْتَعْمِدُ فِيهَا " صِنَادِيقِ سَوْدَاءِ " black boxes
 - عِبَارَةً عَنِ مَكُونَاتٍ مَوْجُودَةٍ، أَوْ سَيَقُومُ بِإِنْشَائِهَا بِالْكَامِلِ فِيمَا بَعْدَ.
 - فَكُّ الْخَطُوطِ الْعَرِيضَةِ فِي الْمَرْحَلَةِ السَّابِقَةِ وَ تَوْضِيحِ كَيْفِيَةِ عَمَلِ كُلِّ صِنْدُوقٍ أَسْوَدٍ تَمَّ اسْتِعْمَالَهُ مِنْ قَبْلِ.

و هذا ما يحدث في **إبداع**، حيث :

- يَقُومُ الْمُبْرِمِجُ بِكِتَابَةِ الْمَسَارِ الْعَامِّ لِلْبَرْنَامِجِ بِاسْتِعْمَالِ إِجْرَاءَاتٍ وَهَمِيَّةٍ لَمْ تُكْتَبْ بَعْدَ؛ لِأَدَاءِ الْمِهَامِ الْكَبِيرَةِ وَ/أَوِ الصَّعْبَةِ وَ/أَوِ الْمُتَكَرِّرَةِ، وَ أَصْنَافٍ وَهَمِيَّةٍ لْتُمَثِّلَ دَوْرَ هِيََاكِلِ الْبَيَانَاتِ الَّتِي يَحْتَاجُ إِلَيْهَا فِي عَمَلِهِ، وَ كَذَلِكَ أَلْقَابٍ وَهَمِيَّةٍ.
 - يَقُومُ الْمُبْرِمِجُ بِكِتَابَةِ أَكْوَادِ تِلْكَ الْإِجْرَاءَاتِ وَ/أَوِ الْأَصْنَافِ وَ/أَوِ الْأَلْقَابِ فِي الْمَرْحَلَةِ التَّالِيَةِ بَحَيْثُ يَكْتَمِلُ الْكُودُ تَمَامًا.
- و نلاحظ هنا التماثل التام بين الأسلوبين و الذي يجعل طريقة **إبداع** مُنَاطِرَةً لِعَمَلِ التَّلَقَّائِيِّ الْبَشَرِيِّ، وَ هَذَا (مِنْ جَدِيدٍ) يَجْعَلُ اللُّغَةَ أَسْهَلَ فِي التَّعَلُّمِ لِلْأَطْفَالِ الصِّغَارِ وَ الْمُبْتَدِئِينَ.

أسباب إمكانية جعل تعريفات الألقاب و الإجراءات و الأصناف مختلطة، و عدم إجبار المبرمج علي تعريف الألقاب سويًا و الإجراءات سويًا و الأصناف سويًا:

- تسهيلات علي المبرمج؛ ففي حالة وُجُوبِ ضَمِّ الْمُتَشَابِهَاتِ فَسَيَكُونُ مِنَ الْمُرْهَقِ عِنْدَ تَعْرِيفِ مَكُونٍ جَدِيدٍ الْبَحْثُ عَنِ مَكَانِ الْفِئَةِ الَّتِي يَنْتَمِي إِلَيْهَا ذَلِكَ الْمَكُونُ لِكِتَابَةِ تَعْرِيفِهِ هُنَاكَ، وَ فِي كُلِّ مَرَّةٍ نَعْرِفُ فِيهَا إِجْرَاءًا جَدِيدًا سَنَضْطَرُّ لِلْبَحْثِ عَنِ مَكَانِ تَعْرِيفِ الْإِجْرَاءَاتِ لِكِتَابَةِ ذَلِكَ الْجَدِيدِ هُنَاكَ، وَ كَذَا مَعَ الْأَصْنَافِ وَ الْأَلْقَابِ!
- زِيَادَةٌ غَيْرُ مُبْرَرَةٍ لِلقَوَاعِدِ لِأَنَّهَا لَا تَأْتِي بِفَائِدَةٍ مُعَيَّنَةٍ لَا يُمَكِّنُ الْاسْتِغْنَاءَ عَنْهَا.
- لِإِعْطَاءِ الْمُبْرِمِجِ الْقُدْرَةَ عَلِي ضَمِّ الْمَكُونَاتِ مُخْتَلِفَةِ الْأَنْوَاعِ وَ الَّتِي تُرْتَبِطُ وَظَائِفِيًا مَعَ بَعْضِهَا الْبَعْضُ، بَحَيْثُ عِنْدَ التَّعْدِيلِ فِي كَيْفِيَةِ الْقِيَامِ بِتِلْكَ الْوِظَائِفِ لَا تَكُونُ هُنَاكَ حَاجَةٌ لِلذَّهَابِ بَعِيدًا، وَ كَذَا عِنْدَ

الحاجة لتطبيق أمر من الأوامر علي المكونات المُختلفة التي تُؤدّي وظيفة مُعيّنة (مثل تحويلهن إلي تعليقٍ لسببٍ من الأسباب).

أسباب وجوب تعريف أي صنف خارج الملف الرئيس في ملف باقية مستقل له نفس اسمه:

- في البرامج الصغيرة لن يحتاج المُبرمج إلي ملفات باقاتٍ علي الإطلاق، لأنه سيضع الأصناف التي يحتاجها في الملف الرئيس لصغر حجم الكود كله، و بالتالي لن يُشكّل هذا الأمر له أي فارق.
- في البرامج المُتوسّطة و الكبيرة سيكون من الأفضل جعل كل صنفٍ في ملفٍ مُستقلٍ لما يلي:
 - ضخامة الأصناف في البرامج الكبيرة تجعل من الأفضل توزيع كل صنفٍ في ملفٍ مُنفصلٍ لضمان صغر حجم الأكواد في كل ملفٍ قدر الإمكان.
 - سرعة الوصول إلي الصنف المرغوب في الوصول إليه.
 - المُساعدة القصوي للمُترجم في أداء عمله بسرعة و بساطة (و هذا يرتبط ببناء المُترجم الداخلي).
 - المُساعدة علي إعادة الإستعمال القصوي بمُجرّد نسخ الملف و لصقه في المشروع الجديد.
 - تماشيًا مع مبدأ "تشابه الافتراضي مع الملموس" حيث في إبداع تُوجد القاعدة: "الملف = صنف، و المُجلّد = باقة" و التي تُسهّل التعلّم علي المُبتدئين و صغار السن، لأن هؤلاء لن يكون بإمكانهم إستيعاب المفاهيم المُجرّدة (مثل: الباقات) إن لم ترتبط في أذهانهم بشيءٍ ملموس يرونه بأعينهم (مثل: المُجلّدات).

أسباب جعل تعبير (من - إلي - بقيمة -) بهذا الشكل:

- في هذا التعبير فإن قيمة الزيادة (التي تُوضَع بعد كلمة بقيمة) تُجمَع علي الحدّ السابق في المُتسلسلة series لإنتاج الحدّ الجديد، و سبب ذلك:
 - هذه هي الطريقة الوحيدة لإنتاج المُتسلسلات في الأحيان التي لا يكون هناك اسمٌ للحدّ السابق نعتد عليه عند وضع تعبير قيمة الزيادة (مثل حالة: بينما - في من - إلي - بقيمة -).
- حيث في تعبير (بينما) يُمكن وُضَع:

بينما أ في من 1 إلي 100 بقيمة أ+2 :
\ الكود هنا /

لكن في حالةٍ مثل :

رقم {1} الأرقام = {من 2 إلي 6 بقيمة 2}

لا يكون هناك رمزٌ للحدِّ القديم لوضعه في مُعادلة حساب قيمة الحدِّ الجديد، و بالتالي يجب أن يتم هذا الأمر ضمناً.

○ يُمكن محاكاة الحالة التي لا يكون فيها الحدِّ الجديد خاضعاً لصيغة:

$$\text{الحد الجديد} = \text{الحد القديم} + \text{قيمة انتقالية}$$

و ذلك عن طريق جعل القيمة الإنتقالية كما يلي:

$$\text{القيمة الانتقالية} = - \text{الحد القديم} + \text{قيمة إنتقالية داخلية}$$

و بالتالي تكون مُعادلة حساب الحدِّ الجديد قد تحوّلت لما يلي:

$$\text{الحد الجديد} = \text{الحد القديم} + (- \text{الحد القديم} + \text{قيمة انتقالية داخلية})$$

$$\text{الحد الجديد} = \text{قيمة انتقالية داخلية}$$

مثال: البرنامج التالي:

```

رقم الرقم
رقم {3} الأرقام = {
| { من 1 إلي 31 بقيمة قيمة () } { 3 2 1 } { 3 2 1 }
| { من 2 إلي 200 بقيمة 2 } { { 3 2 1 } }
بينما الرقم في الأرقام :

```

أكتب نص سطر ("قيمة حد الأرقام الجديد هي " + إلي نص (الرقم))

إجراء (رقم الحد الجديد) قيمة :

$$\text{رقم مشترك الحد القديم} = 1$$

$$\text{الحد الجديد} = - \text{الحد القديم} + 2 \times \text{الحد القديم} + 1$$

$$\text{الحد القديم} = 2 \times \text{الحد القديم} + 1$$

له الخُرج:

قيمة حد الأرقام الجديد هي 1.0

قيمة حد الأرقام الجديد هي 3.0

قيمة حد الأرقام الجديد هي 7.0

قيمة حد الأرقام الجديد هي 15.0

قيمة حد الأرقام الجديد هي 31.0

قيمة حد الأرقام الجديد هي 1.0

-
-
- قيمة حد الأرقام الجديد هي 200.0
- قيمة حد الأرقام الجديد هي 1.0
- قيمة حد الأرقام الجديد هي 2.0
- قيمة حد الأرقام الجديد هي 3.0

فهنا حينما أردنا عمل صَفٍ له القيمة التالية { 1 3 7 15 31 } أي أن :

$$\text{الحد الجديد} = \text{الحد القديم} \times 2 + 1$$

كتبنا الكود:

$$\text{رقم } \{1\} \text{ الأرقام} = \{ \text{من } 1 \text{ إلي } 31 \text{ بقيمة قيمة } () \}$$

إجراء (رقم الحد الجديد) قيمة انتقالية :

$$\text{رقم مشترك الحد القديم} = 1$$

$$\text{الحد الجديد} = \text{الحد القديم} + 2 \times \text{الحد القديم} + 1$$

$$\text{الحد القديم} = 2 \times \text{الحد القديم} + 1$$

أسباب جعل تعبير (من - إلي - بقيمة -) محصوراً علي القيم الرقمية فقط دون كائنات الأصناف و كائنات الألقاب:

- التعبير لا يُستخدم فقط في تعبير بينما و لو، بل يُستخدم كذلك في إعطاء الجداول قيمة. مثال:

$$\text{رقم } \{1\} \text{ الأرقام} = \{ \text{من } 10 \text{ إلي } 100 \text{ بقيمة } 10 \}$$

- وبالتالي مَدُّ قدرات التعبير لتشمل كائنات الأصناف و الألقاب سيجعل استخدامه في إعطاء قيم لجداول كائنات الأصناف و الألقاب صعباً جداً و له قواعد مُعقَّدة، لذا كان الهرب من كل هذا.
- عدم القدرة علي استخدام هذا التعبير مع كائنات الأصناف و الألقاب لا يعني عدم القدرة علي أداء نفس العمليات عليهما، و المثال التالي يوضح هذا:

$$\text{الكائن} = \text{أول. كائن}$$

$$\text{بينما الكائن} \# \text{ آخر. كائن} :$$

$$\text{\ / افعال شيئاً ما /}$$

$$\text{الكائن} = \text{الكائن_التالي}$$

و لو كان بالإمكان كتابة هذا بتعبير (من - إلي -) لكان كالتالي:

بينما الكائن في من أول. كائن إلي آخر. كائن بقيمة الكائن = الكائن_التالي :
 \ افعال شيئاً ما /

و الفرق بين الكودين هو سطران لا أكثر، و في حالة كون التعبيرات التي تلي كلمات من و إلي و بقيمة مُعقَّدةً فربما نحتاج إلي جعل كل واحدٍ منها في سطرٍ مُستقلٍ، مثل الكود التالي:

بينما الكائن في من تعبير. الحصول. علي. القيمة. الإبتدائية
 |
 إلي تعبير. الحصول. علي. القيمة. النهائية
 |
 بقيمة تعبير. القيمة. الانتقالية :
 \ افعال شيئاً ما /

و نظيره المُكافيء له باستخدام الطريقة العادية هو:

الكائن = تعبير. الحصول. علي. القيمة. الإبتدائية
 بينما الكائن # تعبير. الحصول. علي. القيمة. النهائية:
 \ افعال شيئاً ما /
 تعبير. القيمة. الانتقالية

فترى هنا أن الفرق تلاشي. مع الإحتفاظ لكلا التعبيرين بالوضوح و البُعد عن التعقيد.

سبب عدم ضم ما يُمائل المُعامل المنطقي XOR كُمعاملٍ قياسي في اللغة:

لأن المُعامل # يقوم بذات الوظيفة!، و انظروا إلي جدولي الصواب لكلٍ منهما لتروا أن الأمر صحيح:
 جدول صواب المُعامل #

صحيح	صحيح	خطأ
صحيح	خطأ	صحيح
خطأ	خطأ	خطأ
خطأ	صحيح	صحيح

جدول صواب المُعامل XOR

صحيح	صحيح	خطأ
صحيح	خطأ	صحيح

خطأ	خطأ	خطأ
صحيح	صحيح	خطأ

و هما مُتماثلان كما هو واضح.

لكن الأمر الغريب و المُثير للسخرية أن لغات برمجة شهيرة تُقدِّم المُعامل XOR مع أنه يمكنها الاستغناء عنه كما فعلنا في إبداع كتجلٍ جديدٍ من تجليات عدم محاولة جعل تلك اللغات تتبع نظام " حَمِيَّة " مناسب !

يُمْكِنُ كِتَابَةُ :

لو س في { 7 6 5 4 3 2 1 } :

\ أوامر /

و لا يُمكنُ كتابة :

لو س في { { 6 5 4 } { 3 2 1 } } :

\ أوامر /

و ذلك لأنه من غير المنطقي كتابة الصيغة الثانية بينما الأولى تؤدي الغرض و لا تُرهق مُبرمج المُترجم. كما أن إجبار المُبرمج علي جعل كوده بالصيغة الأولى و ليس بالصيغة الثانية يجعل الكود أكثر مقروئية. خذ الجدول الذي في المثال التالي حتي تُصبح الصورة أوضح :

لو س في { { { 18 17 16 } { 15 14 13 } { 12 11 10 } } { { 9 8 7 } { 6 5 4 } { 3 2 1 } } } :

\ أوامر /

و الآن قارن بينه و بين المثال التالي :

لو س في { 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 } :

\ أوامر /

هل رأيتَ الفرق ؟

سبب الإبقاء علي مُعاملات الوصول و الاستنساخ في المسار العام للبرنامج و الإجراءات رغم أنها لن تشكل فارقا :

- عدم إرهاق المُبرمج بمعرفة أنه :
 - لا يَصِحُّ استخدام مُعامِلات الوصول في المسار العام أو الإجراءات. بل يَصِحُّ في الأصناف فقط.
 - لا يَصِحُّ استخدام الاستنساخ في المسار العام، بل يَصِحُّ في الإجراءات و الأصناف فقط.
 - الثبات يُمكن استخدامه في أي مكان.
- أما في حالة الإبقاء علي إمكانية كتابة الصفات التي يُريدها المُبرمج في أي مكان باختلاف المعاني حسب السياق فيكون عليه أن يدرك تلك المعاني (البَدْهِيَّة البسيطة) فقط، و ليس عليه التنبه لما يَصِحُّ و ما لا يَصِحُّ.
- عدم إرهاق مُبرمج السَّابِرِ parser بحالاتٍ و قواعدٍ ليس لها داعٍ بينما يُمكن تسهيل الأمر عليه و المُستخدِم في ذات الوقت.

سبب جعل إعطاء القيم الافتراضية لمُعامِلات الإجراءات عن طريق ترتيبها. و ليس بالاسم (named parameters) :

- لو تم عمل مكتبة استُخدمت فيها طريقة الـ named parameters ثم استخدمها الناس بالفعل، ثم قمنا بتغيير أسماء المُعامِلات (أو حتي واحدٍ منها فقط): فسوف يحدث انهيارٌ لأكواد المُبرمجين التي تستخدم الإجراءات المُعتمِدة علي هذه الطريقة في المكتبة !
- تسهيلات علي مُبرمج السَّابِرِ و المُبرمج مُستخدِم اللغة؛ لأنه حتي لو دَعَمْنَا الطريقة الثانية فسنكون مُضْطَرِّين لدعم الطريقة الأولى !، أي أن الطريقة الثانية لا تُمَثِّل إلا زيادةً في القواعد يُمكن الإستغناء عنها، فلمَ لا نستغنى عنها إذا؟!
- وجود الطريقة الأولى فقط أسهل في الشرح و التعلُّم من شرح الطريقتين الأولى و الثانية.

سبب الاستغناء عن جملة do - while :

الحق أنه كان قراراً سهلاً جداً لأن التعبير التالي يُؤدِّي نفس الوظيفة:

بينما صحيح :

\ أوامر /

لو \ شرط / :

أنهي ()

و ما دام هناك البديل السهل الذي لا يُؤدِّي إلي زيادة عدد القواعد: فلمَ ندرج قاعدةً أُخري إضافية إلي اللغة؟!

من الأسباب التي تجعل من الأفضل عدم ضم المُعامِلات المنطقية الحساسة مثل | & في لغة الجافا :

- تُؤدِّي إلى الوقوع في أخطاءٍ منطقيةٍ قد لا ينتبه المُبرمج إلى سببها بسهولة؛ نتيجةً لعدم تنفيذ المُعاملِ للشِّقِّ الثاني من العملية المنطقية و الذي قد يكون استدعاءً لإجراءٍ له أعراضٌ جانبية **side effects**، و بالتالي لا يتم استدعاء الإجراء و هو ما يتسبب في عدم تنفيذ الأعراض الجانبية مع أن المُبرمج قد اعتمد علي أنها ستُنَفَّذ! و هذا واحدٌ من أعوص المشاكل في الإكتشاف؛ نظراً لأن عين المُبرمج تمر مرور الكرام علي موضع الغلط دون الإلتباه إليه (و أقول هذا عن تجربةٍ واقعية).
- يُمكن محاكاة هذا السلوك التحسيني بوضع القواعد التالية في المُترجم و المُفسِّر:
 - إذا كان في العملية استدعاءً لإجراءٍ فسيتم تنفيذه لا محالة؛ لتجنب افتقاد أعراضه الجانبية.
 - إذا لم يكن هناك استدعاءاتٍ لإجراءاتٍ فسنُطبق طريقة المُعاملات الحَسَّاسة، و بالتالي لو حَوَّل المُبرمج الكود التالي:

لو الإجراء (5 10) < 15 و س :

كود /

إلي ما يلي:

رقم ص = الإجراء (5 10)

لو ص < 15 و س :

كود /

فسيستفيد من التحسينات المُمكنة.

سبب عدم طرح إجراءاتٍ حرّةٍ قياسيةٍ يتم استدعاء كل واحدٍ منها عند: قراءة قيمة حاوية. أو تعديل

قيمة حاويةٍ من الحاويات. كرد فعلٍ لما فعله المبرمج :

مثلاً يكون الإجراءين المُرتبطين بالمتغير (الرقم) كما يلي:

رقم الرقم

إجراء حر تعديل.الرقم :

كود /

إجراء حر قراءة.الرقم :

كود /

بحيث تُحاكي هذه الإجراءات عمل المُغلِّقات **properties**.

الجواب:

- سيتم استدعاء إجراءات التعديل بعد التعديل الفعلي في الحاوية، وهذا أقل قوة من الحالة التي يتم فيها فحص القيمة قبل إسنادها للحاوية (مثلما يحدث في حالة الأغلفة أو إجراءات `get` و `set` العادية التي سنُسَمِّيها الإجراءات المُغَلِّفة منذ الآن فصاعداً).
 - ولُمحاكاة الأسلوب الأخير سوف تتعقد الإجراءات الحرة، وتتعدّد كيفية عملها، وكذلك كيفية إسناد القيم للحاويات نفسها بما يجعل تكلفة الأمر أكبر بكثير من فائدته التسهيلية!
 - يُمكن للإجراءات المُغَلِّفة القيام بهذا الدور بُمُنْتَهِي الكفاءة بدون الحاجة لقواعدٍ إضافية في اللغة و المُترجم و المُفسِّر، و في نفس الوقت تحتفظ هذه الإجراءات بميزة أنه يُمكن إعطاء كل واحدٍ فيها مُستوي و صولٍ و/أو نوع و/أو استنساخ مُختلِف عن الإجراء الآخر و/أو عن الحاوية التي يتعلّق بها عمله.
 - سيَتسبب وجود هذه الإجراءات الحرة القياسية في عملٍ عبءٍ مُخيف في زمن التنفيذ؛ نظراً لحاجتنا الماسّة لتوفير القدرة علي التعامل معها دوماً في زمن التنفيذ لكل الحاويات التي في المكتبات أو في برنامج المُستخدِم، و فكرة (فحص وجود التعامل مع هذه الإجراءات في كود البرنامج لتحديد: هل يوجد سببٌ لوضع كود لها في زمن التنفيذ أم لا؟) سوف يُؤدّي تطبيقها إلي بطء عملية الترجمة بشدة، بمقدارٍ يتناسب مع حجم المكتبات المُستخدمة و عدد الحاويات فيها! و هو الأمر الذي يستحيل قبوله عملياً.
 - سوف تظهر أسئلةٌ مثل:
 - هل التعامل مع حاويةٍ داخل كائنٍ صِنْفٍ يُعتبر تعاملًا مع كائن الصنف، بحيث يُتطلّب معه استدعاء رد الفعل الخاص بكائن الصنف؟
 - هل التعامل مع عنصرٍ في جدولٍ يُعتبر تعاملًا مع مُؤشِّر الجدول، بحيث يُتطلّب استدعاء رد الفعل الخاص بمؤشِّر الجدول؟
- و للإجابة عن هذه الأسئلة سوف نُضطر لوضع المزيد و المزيد من القواعد، مما يعني تعقيد نحو اللغة بشدة.

من محاسن وجود الوراثة المتعددة مع أفراد كل صنفٍ خارجي في ملفٍ مُنفصل:

في حالة وجود صِنْفٍ فيه عددٌ كبيرٌ من الإجراءات، و تنقسم هذه الإجراءات إلي مجموعاتٍ وظيفيةٍ مُختلفة؛ فإنه يُمكن وضع الإجراءات الخاصة بكل مجموعةٍ وظيفيةٍ في صِنْفٍ خاصٍ بها، ثم تتم وراثة كل تلك الأصناف في صِنْفٍ واحدٍ يُشكّل هو البديل العملي للصنف القديم المُتخَم.

من فوائد جعل تعليقات إبداع علي الهيئة \ تعليق / :

- صِيغَةُ مُوَخَّدة يُمكن استعمالها في حالة السطر الواحد أو الأسطر المتعددة، و بالتالي أسهل في الحفظ.
- يُمكن لتعليقٍ احتواء أي عددٍ من التعليقات الأخرى، حيث يتم قراءة عدد العلامات \ و انتظار عددٍ يُماثلها من علامات / و هذا يُؤدِّي إلي سهولة تحويل أي مساحاتٍ من الأكواد إلي تعليقٍ دون التضارب الذي يحدث في حالة عائلة الC مع علامات /* */

فوائد منع الأعراض الجانبية في إجراءات الملف الرئيس :

- القدرة علي تنفيذ الإجراءات علي التوازي مع عدم وجود ذاكرةٍ مُشتركةٍ يُخشِي منها علي إطلاق الحرية غير المتناهية للإجراءات في التوازي.
- إعتقاد الإجراءات علي مُعاملاتها فقط يُمكننا من استخدام تحسينات وقت الترجمة، مثل:

رقم ص = 10

رقم س = مضروب(ص)

إجراء (رقم س) مضروب(رقم ص) :

لو ص = 1 أو ص = 0:

س = 1

غيره :

س = ص × مضروب(ص-1)

فبدلاً من أن يتم تنفيذ هذا الإجراء في زمن التنفيذ فيمكن للمُترجم تنفيذه في زمن الترجمة توفيراً للوقت.

- المُساعدة علي جعل الإجراءات كياناتٍ قائمةٍ بذاتها يُمكن نقلها من مكانها بسهولة؛ حيث يُساعد ذلك علي البرمجة و التصميم بما يُشبه الحالات التي لا نعرف فيها شكل البرنامج النهائي، و إنما نعرف بعض الخصائص التي نريد بناءها و بالتالي نتمكن من بناء الإجراءات الخاصة بها، و عندما نريد ترتيب الإجراءات علي نحوٍ مُختلفٍ يُمكننا ذلك بدون القلق من كون الإجراءات مُرتبطاً بمكانه بسبب الأعراض الجانبية التي تنتج عنه أو عن أحد الإجراءات الأخرى.
- تُتيح لنا إمكانية استخدام الحساب المُصغَّر lazy computation متي أردنا ذلك و وجدناه أنفع للأداء، حيث أنه ما دام الإجراء ليس له أي تأثيرٍ إلا علي مُخرجاته، و لا يُؤثِّر بالأعراض الجانبية علي مُكوِّنٍ آخر؛ فإن ذلك معناه أن عدم استدعائه لن يُشكِّل مُشكلةً للبرنامج.

و رغم أن الإجراء قد يكون له أعراضٌ جانبيةٌ حتى و لو كان لا يُؤثِّرُ إلا علي مُخرجاته (مثل تغيير حقل بياناتٍ في كائنٍ صِنْفٍ، أو استدعاء إجراءٍ ذي أعراضٍ جانبيةٍ) إلا أن ذلك سهل التعامل معه بالقاعدة التالية :

(لو كان الإجراء لا يُؤثِّرُ إلا علي مُخرجاته، و كل مُدخلاته من النوع المُمرَّر بالقيمة: فيمكننا استخدام الحساب المُصغَّر معه لو رغبتنا في ذلك).

القاموس

بعض الإِصطلاحات العربية المُستخدمة ليست هي الأشهر في ترجمة مُرادفاتها التقنية الإنجليزية، وإنما استخدمتها لأنها أدق و أوضح من حيث المعني، بينما الترجمة المشهورة حَرْفِيَّةٌ أكثر من اللازم.

الإِصطلاح الإنجليزي المُكافئ	الإِصطلاح العربي
class	صنف
Class Instantiating	استنساخ صِنْف
enumeration	لقب
productivity	إنتاجية
comment	تعليق
method	إجراء داخل صِنْف
native code	كود أصلي
managed code	كود محكوم
Object oriented programming	برمجة كائنية
functional programming	برمجة وظيفية
interpreted language	لغة تفسيرية
compiled language	لغة مُترجمة
Pointer	مُؤَبِّر
compiler	مُترجم
variable	مُتغيِّر
exceptions handling	مُعالجة الأغلط
operator	مُعامل
property	مُغلف
interpreter	مُفسِّر
writability	مكتوبية
constructor	مُشيد
kernel	نواة (في نظام التشغيل)
structs	هياكل
interface	واجهة
inheritance	وراثة

delegate	وَسَيْط
procedure	إِجْرَاء
Procedure call	اسْتِدْعَاءُ إِجْرَاء
Variable definition	إِشْهَار (تَعْرِيف) مُتَغَيِّر
parameter	مُعَامِل (خُلْ إِجْرَاء)
Reserved word	كَلِمَةٌ مَحْجُوزَةٌ
space	مَسَافَةٌ
indentation	إِزَاحَةٌ
namespace	حَيِّزُ أَسْمَاء
Multiple inheritance	وَرَاثَةٌ مُتَعَدِّدَةٌ
Code block	مَسَاحَةٌ تَكْوِيدِيَّةٌ = كِتْلَةٌ مِنَ الْكُودِ
Procedures overloading	تَحْمِيلُ الْإِجْرَاءَاتِ
Exception throwing	إِلْقَاءُ غَلَطٍ
abstracting	التَّجْرِيدُ
recompile	إِعَادَةُ تَرْجُمَةٍ
index	مُمَيِّزٌ
Side effects	الْأَعْرَاضُ الْجَانِبِيَّةُ
Structured query languages (SQL)	لُغَاتُ الْإِسْتِعْلَامِ الْمُهَيِّكَلِ
Digital image processing	مُعَالَجَةُ الصُّورِ الرَّقْمِيَّةِ
header	تَرْوِيسَةٌ
Track bar	عَمُودُ تَعَقُّبٍ
browser	مُتَصَفِّحٌ
portability	مَحْمُولِيَّةٌ
GUI applications	بَرَامِجُ بَوَاجِهَةٍ رَسُومِيَّةِ
Console applications	بَرَامِجُ بَوَاجِهَةٍ نَصِّيَّةِ
Clients applications	بَرَامِجُ عَمَلَاءِ
server	خَادِمٌ
Servers applications	بَرَامِجُ خَوَادِمِ
static typing	تَنْوِيعٌ ثَابِتٌ
Dynamic typing	تَنْوِيعٌ مُتَغَيِّرٌ

Safe programming	برمجة آمنة
unsafe programming	برمج غير آمنة
GUI designer	مُصمِّم واجهة رسومية
struct	هَيْكَل
task	مهمة
indexer	مُفَهِّس
union	دَامِج
implementation	بِنَاء
Header file	ملف ترويسة
stack	مُكَدِّس
Backword compatibility	توافق عكسي
Functions pointers	مؤشرات الدول
Compile time	زمن الترجمة
Run time	زمن التنفيذ
Generic codes	الأكواد العامة
series	مُتَسَلِّسِلَة
lazy computation	الحساب المُصَغَّر

المصادر

- learning OpenCV (computer vision with OpenCV library). By: gary bradski & adrian kaebler. O'REILLY.
- Professional C Sharp 3rd Edition – Wrox.Press -2004 by:
 - a. Simon Robinson
 - b. Christian Nagel
 - c. Jay Glynn
 - d. Morgan Skinner
 - e. Karli Watson
 - f. Bill Evjen
- C Sharp, from Wikipedia, the free encyclopedia.
- C# How To Program (Redistilled In One Book) by: Deitel.
- Slides named "Introduction to C#" Anders Hejlsberg, Distinguished Engineer Developer, Division Microsoft Corporation

OMS

- (الكامل في السى شارب) لِمُؤَلِّفٍ حَقِيقَةً لَا أَعْرِفُ اسْمَهُ لَكِنَّهُ يَضَعُ لِنَفْسِهِ الرَّمْزَ
- Introduction to C# The New Language for Microsoft .NET, by H.Mössenböck, University of Linz, Austria.
- Beginning Programming with Java for Dummies, 2nd Ed – [Wiley], by: Barry burd.
- Introduction to Java programming: comprehensive version, by: Y. Daniel Liang.—6'Th Ed.
- Comparison of C Sharp and Java, From Wikipedia, the free encyclopedia.
- Java (programming language), From Wikipedia, the free encyclopedia.
- (برمجة الحاسب)، المملكة العربية السعودية، المؤسسة العامة للتعليم الفني و التدريب المهني، الإدارة العام لتصميم و تطوير المناهج.

- C++ Data Structures 3rd.Edition, by: Nell dale, university of Texas, Austin.
- كتاب (Visual Basic للجميع) الإلكتروني للمؤلف: تركي العسيري.
- كتاب (إبدأ مع لغة أوبجكت باسكال Object Pascal)، للأخ: معتز عبد العظيم الطاهر (أبو ياس).
- كتاب (الخطوة الثانية مع أوبجكت باسكال - صناعة البرمجيات)، للأخ: معتز عبد العظيم الطاهر (أبو ياس).
- كتاب (بايثون بلمسة)، للمؤلف: مصطفى فرحات.
- ترجمة كتاب (a byte of python) للمؤلف: Swaroop C H. ترجمة: أشرف علي خلف.
- F# Eye for the C# guy, by: Leon Bambrick, secretGeek.net
- Why F# is like it is, by: Robert Pickering, ALTI
- البرمجة بلغة باسكال)، للمؤلف: وجدى عصام عبد الرحيم
- Eiffel Analysis, Design and Programming Language, Standard ECMA-367, 2nd Edition - June 2006
- unix haters handbook, Edited by Simson Garfinkel, Daniel Weise, and Steven Strassmann, Illustrations by John Klossne. Published by :IDG Books Worldwide, Inc.
- الورقة العلمية (جملة goto تعتبر ضارة goto statement cosidered (harmeful). للبروفيسور: edsger w.dijkstra.
- الورقة العلمية (ملاحظات حول تصميم لغة البرمجة hints on programming (language design) للبروفيسور C. A. R. HOARE.
- مجلة (مجتمع لينوكس العربى)، العدد 7 الصادر فى رمضان 1430هـ (الموافق أغسطس 2009م).
- The Measured Cost of Conservative Garbage Collection by B. Zorn

- كُتِبَ (مقدمة إلى لغة ada)، إعداد: مهندسة تولاي شاهين، جامعة حلب، كلية الهندسة المعلوماتية.
- The Book (ADA PROGRAMMING), by Wikibooks contributors, Developed on Wikibooks the open-content textbooks collection.
- موقع: stackoverflow.com
- مواضيع مختلفة في [.Wikipedia, the free encyclopedia](https://en.wikipedia.org)

السَّيرَةُ الذَّاتِيَّة



البيانات الشخصية:

الإسم: م. وائل حسن محمد على عبدالمنعم (كنيتي: أبوإياس)

الجنسية: مصري

محل الميلاد: أسوان- كوم أمبو

الحالة الإجتماعية: أعزب

تاريخ الميلاد: 22/10/1987

الموقع الشخصي:

<http://afkar-abo-eyas.blogspot.com>

البريد الإلكتروني:

Wael_hasan_87@yahoo.com

المؤهلات الدراسية و التخصص:

جامعة جنوب الوادي

كلية الهندسة و التكنولوجيا بأسوان

قسم كهرباء

شعبة حاسبات و نظم

مشروع التخرج: (game development using C# and XNA):
(library).

تقدير المشروع : إمتياز التقدير العام: جيد

التخصص الحالي: مهندس برمجيات في مجال تصميم و بناء لغات البرمجة، (بناء لغات البرمجة يتضمن تصميم و بناء المُترجمات compilers و المُفسِّرات interpreters و غيرهما من الأدوات الأخرى).

و مُصمِّم لغة البرمجة العربية الاحترافية (إبداع) ، و باني المُفسِّر القياسي (أبداع
obde3) لها.

المهارات الحاسوبية:

- لغات البرمجة:

- Java
- C#
- C++
- Python
- JavaScript
- Object Pascal
- Fortran
- matlab
- VB.NET
- ruby

- نظام التشغيل المُفضَّل هو القنول/لينوكس GNU/linux , و خاصةً توزيعة kubuntu.

- أقوم منذ شهر رجب لعام 1433هـ (الموافق لشهر يونيو لعام 2012م) بنشر مقالاتٍ تقنيةٍ علي موقع (وادي التقنية Itwadi.com).

خبرة العمل :

• نوفمبر، 2010 إلى سبتمبر، 2011

بأحثٍ مُساعدٍ في مركز أبحاث الذكاء الاصطناعي و الروبوت Center for Artificial Intelligence and Robotics - C.A.I.R.O بكلية العلوم بأُسوان.

• سبتمبر، 2011 إلى الأبد إن شاء الله عز و جل

مُتفرِّغٌ لمشروع (البرمجة بإبداع).

أدوات إنتاج هذا الكتاب



- توزيعة kubuntu من نظام التشغيل قِنُو لِينُوكَس GNU/Linux.

- برنامج LibreOffice writer لكتابة الوثائق.



- برنامج Gwenview لعرض الصور و المعالجة البسيطة لها.

- برنامج gimp لتعديل الصور باحتراف.



- برنامج kolourpaint لتعديل الصور بشكل بسيط.



رِسَالَةُ الْبَرْمَجَةِ بِإِبْدَاعِ

(رسالة البرمجة بإبداع[™]) هي رحلة علمية مُشوّقة يتم فيها التحدث باستفاضة عن: مشروع البرمجة بإبداع[™] و الدوافع وراءه و أهدافه المُتنوعة. شرح لغة البرمجة العربية الاحترافية إبداع[™] علي نحو مُختصر. ثم شرح الأسباب المنطقية وراء ذلك التصميم بالتفصيل في عشرات الصفحات التي أظن أنها ستكون مُمتعة لمن يُحب البرمجة و الابتكارات الجديدة فيها؛ لأنها تطوف في العوالم البرمجية المُختلفة و تُريكم ما وراء الكواليس قدر الإمكان حتي يتبين وجه التعارض أو التوافق معه.

م. وائل حسن محمدعلي (أبو إياس)