مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (30) العدد (4) 2008
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (30) No. (4) 2008

تطوير آلية لقياس أداء لغات البرمجة المختلفة عند استخدامها في بناء الأنظمة

الدكتور عبد الرحمن حسين*
الدكتور جبر حنا **

(تاريخ الإيداع 10 / 4 / 2008. قُبِل للنشر في 2008/11/2)

□ الملخّص □

قمنا في هذا البحث بدراسة أداء لغات البرمجة، وذلك بهدف وضع آلية اختيار علمية من خلال تطبيقات مختلفة، تمكننا من خلال هذا الأداء اختيار لغة البرمجة المناسبة للنظام. وتبين لنا من خلال عدة مقارنات لكتابة البرنامج نفسه بعدة لغات أن هناك اختلافات زمنية في الترجمة وفي التنفيذ من أجل ظروف أداء موحدة. ثم قمنا باستخدام مترجمات مختلفة للغة نفسها، وكذلك من أجل لغات مختلفة وهذا بدوره قادنا إلى أن اختيار لغة البرمجة له تأثير كبير وأساسي على تكاليف كتابة البرامج ودورة حياة البرامج "life-cycle software" ولذلك يجب أن تُختار لغة البرمجة بالأهمية نفسها التي تعطى لعملية اختيار الكيان الصلب الأساسي. قمنا بتطبيق أداء الخوارزمية من خلال لغات البرمجة وقياس أداء تجهيز البرامجِ المختلفةِ من خلال المترجمات للغات مثل الباسكال أو سي، وذلك لمُقَارَنَة سلوكِ خوارزميةَ الإختيار.

الكلمات المفتاحية: المترجم– لغات البرمجة– خوارزميات الاختيار– دورة البرامج.

* أستاذ مساعد–قسم التحكم والأتمتة– كلية الهندسة الكهربائية والالكترونية– جامعة حلب–حلب–سورية.
** أستاذ مساعد– قسم الحاسبات والتحكم الآلي– كلية الهندسة الكهربائية والميكانيكية– جامعة تشرين– اللاذقية –سورية.

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (30) العدد (4) 2008
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (30) No. (4) 2008

Developing A Mechanism of Measuring the Performance of Different Programming Languages for Building Systems

Dr. Abdul Rahman Hussian [*]
Dr. Jabr Hanna [**]

(Received 10 / 4 / 2008. Accepted 2 / 11 / 2008)

□ ABSTRACT □

This paper studies the performance of programming languages in order to devise a scientific mechanism of choosing that helps us choose the appropriate programming language for the system. We show this through writing the same program using different languages. We also use different compilers for the same language; then we find out that choosing a programming language has a great effect on the software life cycle, so choosing a programming language must be as equally important as choosing the basic hardware. So we apply algorithms to measure the performance of different language such as Pascal or C in order to compare the performance of the algorithm of choice.

Keywords: Compiler, Programming languages, Algorithm of choice, software life-cycle

[*] **Associate Professor, Department of Control and Automation Engineering, Faculty Electrical & Electronic Engineering, University Aleppo, Aleppo, Syria.**
[**] **Associate Professor, Department of Computer and Automatic Control, Faculty Mechanical & Electrical Engineering, Tishreen University, Lattakia, Syria.**

مقدمة:

إن عملية اختيار لغة البرمجة عالية المستوى غالباً ما يتم دون دراسة بل يعتمد على معرفة المبرمج بإحدى اللغات البرمجية، ولا تتم أي مناقشة لصلاحية الاختيار، وتشير بعض الدراسات إلى أن الشركات المنتجة لمترجمات اللغة لا تتردد في الإشارة إلى أن اللغة تصلح لجميع الأعمال البرمجية [5] ولا تقدم أي مقارنات لآلية قياس الأداء بل تعتمد على انتشار اللغة بين المبرمجين.

ولكن مع التزايد المستمر لتكاليف البرامج، فإن القرارات المتخذة لاختيار اللغة، يجب أن تستند على الاعتبار الصحيح للإيجابيات والسلبيات التقنية للغات البرمجة المتوفرة والمراد اختيارها.

بالرغم من أنه في بعض الأوقات التي تُحدِّد فيها حالة ما إمكان الاختيار للغة برمجة واحدة أو اثنتين فقط، فإنه يوجد العديد من الحالات الأخرى التي يكون فيها مجال اختيار لغة البرمجة واسعاً جداً [3].

نبين في الجدول التالي بعض اللغات وشعبية عملية الاختيار، ولا يرتبط ذلك بعملية قياس الأداء.

Complete list	Description	Why to use it?
C# (C Sharp)	Concurrent to java, runs on .Net or compatible platforms (Portable net, Mono). More - Code	Easier than C++. May share resources with other languages under a common runtime.
C	C offers great freedom, and lot of debugging challenge thanks to pointers and memory management. More - Code	Suffering slow programming to make fast programs. For system programming.
C++	C++ is C plus objects, an extended library, templates. More - Code	System programming as C but allow larger project, or applications.
Pascal	Old language (1970), improved with objects, imposes a strictly structured programming. More - Code	Teaching, or client/server apps with Delphi and Kylix.
Java	Was designed to be portable and to replace C++. More - Code	Cross-platform applications (but slower than native ones). For web services programming .
Assembler	This is near the machine language and the fastest. You should never use it, as older programmers did.	Making drivers or industrial processing.

Perl	A scripting interpreted language. Readabillity and ease of use are not the goal. Perl - Code	Mainly used by networks administrators and for small CGI scripts.
Scheme	Scheme is a modernized version of Lisp. Scheme - Code	Artificial intelligence and scripting.
SQL	Language of data management. MySQL site	For databases queries.

أهمية البحث وأهدافه:

قمنا في هذا البحث بدراسة المنهجية "methodology" العلمية لاختيار لغة البرمجة عالية المستوى مع المترجم "compiler" المناسب.

إن أي تطبيق بشكل عام لا يحتاج فقط إلى اختيار لغة البرمجة بل يحتاج أيضاً إلى اختيار مترجم اللغة.

والاعتبارات السائدة في اختيار لغة البرمجة من أجل معالج مصغر ما هي:

1- توفر اللغة.
2- المتطلبات المفروضة.
3- التردد في تعلم لغة برمجة جديدة.
4- القلق المُبَالغ به تجاه كفاءة لغة البرمجة.
5- تقليل تكاليف الصيانة الدورية للبرامج .

طريقة البحث ومواده:

تتضمن دراستنا للمنهجية المستخدمة لاختيار لغة البرمجة:

1- التعرف بشكل اوسع على إمكانات لغات البرمجة المتوفرة .
2- استخدام الاختبار القياسي "benchmark testing".
3- حساب مدى الكفاءة "figure of merit" لتقييم الجودة لكل لغة.

يستند الاختبار القياسي على التطبيق المحدد (الذي يراد اختيار لغة البرمجة المناسبة له)، وحتى يكون حساب مدى الكفاءة صحيحاً يجب أن تتوفر القدرة على تمييز الأهمية النسبية للعوامل المتعددة المستخدمة في الحساب.

بالإضافة إلى استخدام الاختبار القياسي لتقييم لغات البرمجة بشكل نهائي فإنه يُستخدم أيضاً في الاختبار الأولي.

- **تجهيز قائمة الاختيار:**

في بداية الدراسة تشكل قائمة تحوي مئات من لغات البرمجة والمترجمات تدعى قائمة الترشيح "candidate list".

هذه القائمة الأولية الضخمة من لغات البرمجة يمكن أن يتم تخفيضها من خلال ثلاثة معايير أساسية نوردها فيما يلي:

– المعيار الأول: توفر المترجم compiler availability

يتم حذف لغة البرمجة وفق هذا المعيار في حال عدم توفر مترجم لهذه اللغة، وبتطبيق هذا المعيار يتم حذف عدد من اللغات من القائمة. واللغات المتبقية تصبح من اللغات الأساسية في قائمة الترشيح.

وبالتالي يرتبط اختيار المترجم بنوع التطبيق المراد اختيار لغة البرمجة المناسبة له.

– المعيار الثاني: اختلافات اللغة البسيطة minor language variations

ويعني هذا المعيار عملية حذف اللغات (من قائمة الترشيح) التي تتوافق مع اللغات المشهورة بعد إجراء بعض التعديلات البسيطة على مترجماتها .

–المعيار الثالث: قلة الوضوح في المُلاءمة clear lack of suitability

إن قلة الوضوح في مُلاءمة اللغة للغرض المطلوب، يؤدي إلى حذف عدد من اللغات، وذلك إما لأنها غير مناسبة للتطبيق، أو لأنها ملائمة لعدد قليل من الاستخدامات. بعد حذف بعض اللغات وفق المعايير السابقة، فإن اللغات المتبقية في قائمة الترشيح تمر بالمراحل التالية من الاختبارات:

أ– الاختبار الأولي Initial testing :

إن اللغات المتبقية سوف يتم إخضاعها إلى دورة الاختبار الأولي، والذي بموجبها يتم حذف لغات البرمجة تبعاً لأحد الأسباب الأربعة التالية:

(1)– السبب الأول هو البطء في زمن التنفيذ "execution time"

إن كل المترجمات أو المفسرات تحتوي على برامج اختبار قياسية لقياس زمن التنفيذ، وبالتالي يتم حذف لغات البرمجة ذات زمن التنفيذ البطيء من قائمة الترشيح.

(2)– السبب الثاني هو أن مصادر الدعم لهذه اللغات تكون محصورة في مصدر وحيد.

(3)– السبب الثالث وجود عدد من المشكلات في بعض لغات البرمجة يتم استبعادها أيضاً ومن هذه المشاكل:

a– عدم جاهزية إصدار اللغة من قبل المصدر عند وقت الدراسة.

b– إذا كانت لغة البرمجة لا تقدم فوائد أكثر من لغة أخرى، لذلك يتم الحكم عليها أن قابلية قراءتها ضعيفة.

c– إذا كانت اللغة ضعيفة في عملية التوثيق " documentation".

(4)– إذا أظهر التحليل أن بعض لغات البرمجة ذات اختلافات بسيطة في الجوهر مع لغة ما، عندئذ يتم حذف هذه اللغات من القائمة لأنها تكافئ هذه اللغة.

ب– مدى الكفاءة Figure of merit(FoM) :

يتم حساب مدى كفاءة كل لغة برمجة تم ترشيحها عن طريق تحديد المميزات التقنية "technical features" الهامة للغات المتبقية، وعن طريق التكرار ثلاث مرات لعملية تدعى طريقة دلفي "Delphi method" يتم تحديد أوزان نسبية "weights" لهذه المميزات التقنية.

يتم الحصول على النتائج من طريقة دلفي [1] بالشكل التالي:

1– يتم تحديد الأوزان النسبية.

2– تعطى نتيجة (من 0 حتى 1.0) لكل مميزة من كل لغة برمجة مدروسة.

3– وأخيراً تعطى كل لغة نتيجة نهائية عن طريق جمع نتائج الأوزان النسبية لكل مميزة تقنية.

يعطى مدى كفاءة لغة البرمجة بالعلاقة:

$$FoM = \sum_{i=1}^{N} W_i S_i$$

حيث:

W_i: هي عامل الوزن من أجل الميزة التقنية "i".

S_i: نتيجة اللغة من أجل الميزة التقنية "i" .

ج- **متطلبات عملية القياس:**

إن لمميزات اللغات المتبقية في قائمة الترشيح ثلاثة متطلبات لعملية القياس هي:

1- الكفاءة الزمنية "time efficiency".

2- الكفاءة الحجمية "space efficiency" .

3- كفاءة زمن الترجمة "compile-time efficiency" .

لكي نقيس هذه المميزات نتبع الخطوات التالية:

1- علينا أولاً أن نصمم ونطور برنامج قياسي "benchmark program".

2- ينفذ هذا البرنامج من قبل كل اللغات المرشحة.

3- تعطي نتيجة التنفيذ القياس لمميزات اللغات وفق المتطلبات الثلاثة لعملية القياس الآنفة الذكر.

وهكذا وبعد التحليل الأولي وحذف بعض لغات البرمجة وفق المراحل الموصوفة سابقاً، يتم تحليل اللغات المتبقية بتفصيل أكبر، وبما أن اللغات المتبقية لديها برامج قياسية مكتوبة ومنفذة، وتتصف بمدى كفاءة محسوبة، فلا بد من إدخال بعض المعايير Selection criteria الإضافية.

إن تطوير معايير القياس لتقييم لغة البرمجة عملية هامة، فبدون أي من خواص اللغة مثل زمن التنفيذ، وحجم البرنامج الذي تم توليده، وزمن الترجمة لا يمكن المقارنة بين لغات البرمجة.

هذه الخواص هي بعض من الخواص العديدة الأخرى التي يجب أن تدخل في عملية التقييم لاختيار لغة البرمجة.

إن المنهجية العامة تسمح بإدراج عدد كبير من المعايير، وتصنف هذه المعايير إلى:

- **معايير الإدارة management criteria**

هناك ثلاثة معايير للإدارة يتم من خلالها تقييم لغة البرمجة، تتعلق هذه المعايير بالمشاريع البرمجية وأنظمة التشغيل.

1. زمن وكلفة التطوير Development time and cost :

بشكل واضح يجب أن تكون التكلفة والزمن أصغر ما يمكن، بالإضافة إلى أنه من أجل أية ميزانية معطاة، فإن تخفيض التكلفة يؤدي إلى تطوير أو إضافة إمكانات خاصة، وهذا المعيار يؤدي إلى تحسين المميزات التقنية للغة البرمجة، والتي بدورها تؤدي إلى تسهيل تطوير البرامج، وتطوير برامج أسهل وتبسيط التطويرات للأدوات البرمجية الإضافية.

2. فعالية النظام System effectiveness :

تتعلق الفعالية بشكل خاص بكيفية عمل البرنامج النهائي (الذي تم تشفيره عن طريق لغة معينة)، وبمعنى آخر تحديد الفعالية التي يعمل بها النظام ككل.

يمكن قياس هذا المعيار عن طريق:

- السهولة في عمل النظام.
- عدم وجود أخطاء.
- أزمنة تنفيذ سريعة.

3. الصيانة الدورية Life-cycle maintenance:

تتعلق الصيانة الدورية بعملين:

1- أعمال التصحيح.

2- أعمال التحسين (التطوير).

ولذلك فإن اللغة التي تؤدي إلى برامج قابلة للتعديل والامتداد "extensible and modifiable software" سوف تخفض تكاليف الصيانة الدورية كما ستؤدي إلى تحسين فعالية التشغيل عن طريق الاستجابة السريعة للتعديلات الضرورية للمستخدم.

هذا المعيار سوف يؤدي إلى تحسين المميزات التقنية للغة البرمجة كقابلية القراءة والغنى بمجموعات متعددة الاستعمال من تراكيب المعطيات.

النتائج والمناقشة:

- **المعايير التقنية technical criteria**

وهي المعايير المتأصلة في لغة البرمجة، ومترجمها، وبالنظام البرمجي الداعم لها.

لكي يتم تقييم اللغة بدقة عن طريق المعايير التقنية ومعايير الإدارة، يجب وضع قائمة من المميزات التقنية المستخدمة في الحكم على كل لغة برمجة مرشحة.

إن الهدف من طريقة دلفي "Delphi method" الموصوفة سابقاً هو الحصول على رتبة الدرجة للمميزات التقنية.

لكي نستخدم طريقة دلفي يتم أولاً تحديد الدرجة الأولية للمميزات التقنية " technical features" التي تحتوي على أهم العوامل"factors" في عملية تقييم لغة البرمجة.

وقد تم توزيع المعايير التقنية على ثلاث مجموعات ذات أهمية نسبية في الدرجة.

- الدرجة الأولى First order:

1- تمثيل المعطيات data representation :

يجب أن يتم الحكم على لغة البرمجة من خلال الإمكانات التي توفرها لتمثيل المعطيات، وقابليتها لدعم القيم الصحيحة وقيم الفاصلة العائمة، وتراكيب المعطيات المعقدة كالسجلات "records"، والقوائم المرتبطة "linked lists" ، والملفات، والأشجار "trees ".

وبالاعتماد على مسائل البرمجة، فإن هذه القدرات تختلف في درجة الأهمية فيما بينها.

2- تراكيب التحكم Control structure:

إن لغات البرمجة التي تسمح بالتراكيب:

DO-WHILE, IF-THEN-ELSE, CASE , BEGIN-END, RAPEAT-UNTIL

هي لغات مفضلة أكثر من تلك التي لا تسمح مباشرة بمثل هذه التراكيب مثل لغات التجميع.

من الضروري الأخذ بعين الاعتبار إمكان وجود التداخل في تراكيب التحكم وقابلية قراءة البرنامج المشفر من القمة للأسفل بدلاً من القراءة من الأسفل للأعلى.

3- برمجة الأنظمة systems programming:

إن برمجة الأنظمة هي تطوير وإنتاج برامج تعمل مع الترجمة، والتحميل، والإشراف، والصيانة، التحكم، وتشغيل الحواسب.

كما أن الاختلاف بين برمجة التطبيقات وبرمجة الأنظمة، يتطلب قابلية العكس أو القلب "ability to invert"، والإزاحة "shift"، والحجب "mask"، وتدوير الكلمة "rotate word" ، والبايت " byte "، وكميات البت" bit "، والمقدرة على معالجة المعطيات، ووضع القيم في الشكل المطلق، ووصف عناوين الذاكرة.

إن اللغات الأفضل لبرمجة الأنظمة، هي التي تسمح بكتابة أنظمة العمل المعقدة دون الرجوع إلى لغة التجميع (Assembly language).

4- بيئة البرنامج الداعمة program support environment:

يجب أن يتم الحكم على لغة البرمجة من خلال بيئة البرنامج الداعمة . فتوفر وسهولة استخدام المحررات"editors"، والمنقحات الرمزية "symbolic debuggers"، والمحاكيات "emulators"، والبرامج المرفقة، والمكتبات، وبرمجة ذاكرات "PROM"، هي من العوامل الهامة في اختيار لغة البرمجة.

- الدرجة الثانية Second order:

من أجل العديد من تطبيقات البرمجة، فإن بعض مميزات اللغة هذه يجب أن تؤخذ بالاعتبار مع مميزات الدرجة الأولى:

1- قابلية التحميل على المعالج الهدف Target CPU transportability:

إن قابلية التحميل تتعلق بمقدرة لغة البرمجة على إنتاج شيفرة لغة الآلة لأكثر من معالج واحد.

2- مدى الاستخدام extent of use:

إن لغات البرمجة المستخدمة بشكل واسع هي اللغات ذات المترجمات المدعومة من قبل شركات كبيرة وهي مرغوبة أكثر من اللغات المدعومة من قبل شركات صغيرة.

وبشكل فعلي يوجد عاملين اثنين:

- لغات برمجة مستخدمة بشكل واسع ومألوفة من قبل مصدر ضخم من المبرمجين.
- ومترجمات مدعومة من قبل شركات ضخمة ووكالات حكومية ومدعومة لمدى طويل.

3- قابلية التعلم learnability:

تتعلق قابلية التعلم بسهولة تعلم لغة البرمجة.

4- التوثيق documentation :

بالإضافة إلى تعليمات المستخدم الأساسية فإن التوثيق الجيد يجب أن يتضمن قوائم للكلمات المحجوزة، ورسائل خطأ مصغرة مرتبة حسب الترتيب الأبجدي أو العددي.

5- كفاءة الزمن time efficiency :

يتم قياس الزمن عن طريق قياس زمن التنفيذ للبرنامج القياسي.

6- كفاءة الحجم space efficiency :

يمكن قياس كفاءة الحجم عن طريق حجم الشيفرة التي تم توليدها للبرنامج القياسي، ورزم الدخل/ الخرج الخاصة المُولدة وقت التشغيل.

7- ترابط لغة التجميع assembly language linkage:

أحياناً من الضروري ربط وحدات لغة التجميع ضمن وحدات تتم ترجمتها بشكل منفصل، والانتقال إلى وحدات ذات شيفرات عالية المستوى.

8- قابلية القراءة readability:

إن إمكان القراءة وفهم الشيفرة في لغة البرمجة مطلوبة من أجل الصيانة، والتطوير.

ويمكن القول بأن إمكان القراءة ذات أهمية أكبر من إمكان الكتابة.

9- استخدام مجموعة أوامر المعالج الذي تم اختياره كمعالج هدف.

- الدرجة الثالثة Third order:

1- تعدد المهام multitasking:

إن خاصية تعدد المهام في لغة البرمجة تسمح للمبرمجين بتوصيف عدة مهام يمكن أن تنفذ على التوازي.

2- خاصية الإعادة return وخاصية التعاودية recursion

وغالباً ما تستخدم هاتين الخاصتين في حالات معالجة المعطيات الديناميكية.

3- كفاءة زمن الترجمة compile-time efficiency:

بالطبع يتم قياسه خلال ترجمة البرنامج القياسي، وهذا الزمن الذي تم قياسه هو الزمن اللازم للترجمة وتحويل البرنامج إلى ملف هدفي قابل للتنفيذ.

4- قابلية نقل بيئة البرنامج الداعمة transportability of program support environment: يمكن أن تقدم فائدة مرة أخرى في مرونة العمل كما سنبين ذلك في التعديلات.

5- القدرة على برمجة ذاكرة ROM أو ما يسمى "ROMable":

إن القدرة على برمجة ذاكرة "ROM" أي تحميل الشيفرة الهدف تعتبر خاصية مرغوبة في تطبيقات الحاسبات الصغرية.

- **التعديلات**

سوف تحسن التعديلات التالية عوامل أو معايير التقييم. وهذه التعديلات هي:

1- تمثيل المعطيات data representation:

وتضم عدة عوامل، والتي تنفصل إلى العناصر الثلاثة التالية:

أ- تراكيب المعطيات "data structure": مثل المصفوفات، السجلات، المؤشرات، .... الخ

ب-نوع المعطيات "data type": القدرة على تحديد أو تعريف أنواع جديدة ودرجة فحص النوع.

ت-مجال المعطيات "data scope": تركيب الكتلة "block structure"، تمرير البرامترات " parameter passing"، التخصيص الديناميكي "dynamic allocation"،... الخ.

2-وصول (ولوج) الآلة machine access:

إن برمجة النظام وترابط لغة التجميع يرتبطان بعلاقة قوية جداً، حيث يمكن أن يتم دمجها، وهذا الدمج مهم جداً، إلا أنه من الضروري وجود إمكان إجراء التعديلات والدمج من جديد.

3- قابلية نقل البرامج:

إن المميزة المتعلقة بقابلية نقل البرامج يجب أن تكون مضمنة في بيئة البرنامج الداعمة. والأكثر أهمية، أنه يجب أن يشير الموضوع إلى كامل الطيف من الأدوات البرمجية المتوفرة لتطوير المعالج الصغري.

هذا العامل له تأثير على الدراسة، ويعتبر مهماً في الاختيار النهائي بسبب إمكان استخدام بيئة البرنامج الداعمة مع الأقراص الضخمة ونظام الملفات الهرمي مع مستويات الحماية المتعددة، وأدوات التحكم للتشكيلة المؤتمتة.

- **القياس**

وهو الاختبارات القياسية التي تستعمل لمقارنة أداء الحاسبات، والمعالجات، والدارات، والخوارزميات. يشير القياس "benchmark" إلى البرنامج المصمم لمقارنة نظام حاسوبي مع نظام آخر، وهو الاستخدام الشائع للبرامج القياسية. بعض المترجمات تقوم بإدخال شيفرة زمن التشغيل "runtime code" داخل البرامج التنفيذية (برامج الهدف) distenation لتدعم وظائف الدخل/الخرج ووظائف أخرى.

بالإضافة إلى أن بعض المترجمات تقوم بإدخال كميات كبيرة من رزم زمن التشغيل "runtime packages" داخل شيفرة الهدف، بينما مترجمات أخرى تدخل رزم صغيرة أو لا تدخل على الإطلاق.

لكي يتم حذف بعض هذه المترجمات المختلفة، يتم تطوير إصدار خاص للبرنامج القياسي، وفي الحقيقة لقد طورت أربعة إصدارات مختلفة للبرنامج القياسي لتحقق المتطلبات المختلفة.

إن الإصدار الأول يدعى "regular benchmark" أو الإصدار القياسي النظامي، والذي يحتوي على البرنامج القياسي متوضعاً داخل برنامج تصميم اللغة بشكل مشابه إلى حد كبير للغة " Pascal"، والغرض منه هو وضع البرنامج القياسي بشكل خوارزمية، ويدعى أيضاً إصدار الرمز فقط "code-only version ".

أما الإصدار الثاني للبرنامج القياسي، فيحتوي بالإضافة إلى الشيفرة أو الرمز على عدة تعليقات "comment" تدعم التعليمات المخصصة لتطبيق البرنامج القياسي في لغات مختلفة.

إن الإصدار الثالث خالٍ من وظائف الدخل/الخرج، وقد تم تطويره لحذف معظم المترجمات التي تولد اختلافات في زمن التشغيل، وهذا كان أساسياً لمقارنة الاختلافات في أحجام البرنامج الهدف الذي تم توليده.

وأخيراً الإصدار الرابع وهو "error-seeded version " والذي يحتوي على البرنامج القياسي الأصلي مع خمسة أخطاء، والتي تم تصميمها لاختبار قابلية البرنامج لاكتشافها وإعطاء التقرير عنها في زمن الترجمة.

خوارزمية العمل:

تبين الخوارزمية التالية آلية عمل البرنامج القياسي:

```
Initialize variables;
Write ('begin execution');
While Timing Control < Time Limit DO
  BEGIN
        For Index := 1 TO Weight Factor1 DO
                BEGIN
                    [Calculation set 1]
```

```
        END;
FOR Index :=1TO Weight Factor2 DO

        BEGIN
            [Calculation set 2]

        END;
          .
          .
For Index := 1 TO Weight Factor i DO
        BEGIN
            [Calculation set (i)]
        END;

END;
WRITE ('END Execution');
```

يقوم البرنامج القياسي بتنفيذ عدد من التعليمات ضمن لغة البرمجة الهدف عدداً من المرات، وبالتالي حساب الزمن اللازم لتنفيذ هذه التعليمات.

فإذا كان الزمن أكبر من زمن التحكم المحدد من قبل المبرمج، فإن البرنامج القياسي يعطي رسالة خطأ، وبالتالي يتم حذف هذه اللغة من قائمة الترشيح.

خطوات التطبيق:

تبين لنا أنه من أجل تصميم برنامج قياسي للمقارنة بين عدة لغات علينا اتباع الخطوات التالية:

1. تحديد اللغات التي يجب المقارنة فيما بينها من حيث زمن التنفيذ.
2. تحديد التعليمات الهامة والتي يجب مقارنة زمن تنفيذ هذه التعليمات بين اللغات مثلاً التعليمات الحسابية أو تعليمات الإدخال والإخراج.....الخ .
3. تحديد زمن معياري أو زمن العتبة لكل تعليمة، والذي يجب أن لا تتجاوزه التعليمة أثناء تنفيذها مثلاً: التعليمات الحسابية نعطيها عامل كفاءة لزمن التنفيذ 0.33 من الزمن الكلي وتعليمات الإدخال والإخراج نعطيها 0.15 ..... الخ .

وبالتالي تكون آلية عمل البرنامج كما يلي:

- الحلقة الخارجية "while" من أجل تنفيذ مجموعة التعليمات المحددة عدد من المرات بحيث لا يتجاوز زمن تنفيذها الزمن الكلي المحدد بعناية من قبل المبرمج.
- الحلقات الداخلية "for" من أجل تنفيذ تعليمة معينة عدد من المرات تحدد من قبل المبرمج أيضاً.

مثال: إذا أردنا التأكد من أن تطبيق معين ينجز حسابات الفاصلة العائمة خلال "30" بالمئة من الزمن، يكون ذلك عن طريق البرنامج القياسي الذي ينفذ حلقة "While" خلال "30" بالمئة من الزمن عند إنجازه لحسابات الفاصلة العائمة (المنفذة ضمن حلقة "for").

- يجب أن تعطي مجموعة الحسابات الناتجة عن البرنامج القياسي حسابات فعلية.
- يحتوي البرنامج القياسي على عدة حلقات يتم التحكم بها عن طريق عوامل الوزن"weight factors"، حيث إن مجموع كل هذه الأوزان هو "1000".

- بتغيير الحلقة التي يتم التحكم بها عن طريق عوامل الوزن، فإن البرنامج القياسي سوف يتغير ليعطي أزمنة تنفيذ مختلفة (اعتماداً على التطبيق المقترح).

بالإضافة إلى قياسات عدد تعليمات البرنامج والزمن (زمن التنفيذ – زمن الترجمة)، فإن البرنامج القياسي يسمح بتقييم موضوعي لمميزات اللغة الهامة مثل بنى التحكم "control structures"، وتراكيب المعطيات، وقابلية برمجة النظام، والتوثيق، وترابط الوحدة "module linkage"، وبيئة البرمجة الداعمة، وقدرة معالجة الخطأ.

إن الإصدار "error-seeded version " مهم جداً في دراستنا هذه لسببين:

- الأول: لأنه يشير بشكل واضح إلى لغات البرمجة الضعيفة.
- الثاني: لأنه يقوم بتشخيص خطأ المترجمات، بالإضافة إلى قدرتها على تصحيح الخطأ.

عموماً، إن الإصدارات المختلفة للبرامج القياسية تسمح بتقييمات موضوعية وكمية للغة البرمجة وخصائص المترجم.

للتأكيد على أنه فعلاً يتم التنفيذ لكامل شيفرة البرنامج القياسي عند تشفير الإصدارات المختلفة للبرامج القياسية فإننا نحتاج إلى برامج تتبع الأخطاء "debuggers" لتتبع تنفيذ هذا البرنامج.

الدراسة التطبيقية:

قمنا بتطبيق هذا المعيار على بعض لغات البرمجة نورد بعضاً منها، وذلك من خلال معالج بسرعة 1600MHZ وذاكرة كاش 512KBYTE وذاكرة رئيسة 512MBYTE وبنظام تشغيل WINXP PACK2 :

1- لغة C++ : قمنا باستخدام المترجم Dev-C++ لتنفيذ الاختبارات التطبيقية وهو من انتاج:

```
Copyright (C) Bloodshed Software
http://www. Bloodshed.net
NU GENERAL PUBLIC LICENSE
Version 4.9.9.2, June 1991
675 Mass Ave, Cambridge, MA 02139, USA
```

يمكن الاعتماد على التوابع الخاصة بهذه اللغة لدراسة زمن تنفيذ تعليمات برنامج ما.

فالبرنامج التالي يعتبر برنامج اختبار بسيط للتوابع المستخدمة من أجل هذا الغرض:

```
#include <time.h>
#include <stdio.h>
#include <dos.h>
int main(void)
{
clock_t start,end;
start = clock();
delay(2000);
end = clock();
printf ("the time was%f\n",(end-start)/CLK_TCK);
return 0;
}
```

ومن أجل برنامج أكثر تعقيداً من حيث تداخل الحلقات وبنى المعطيات، يمكن الاستفادة من التوابع السابقة في حساب زمن التنفيذ، كما هو الحال في البرنامج التالي:

```
#include <time.h>
#include <stdio.h>
```

```
#include <dos.h>
#include <math.h>
#include <iostream>
using namespace std;
int main(void)
{
clock_t start,end;
start = clock();
end=0;
int t=0;
while ( t<=7)
{
  for (int i=0;i<=5;i++)
  for (int j=0;j<=5;j++)
  for (int k=0;k<=5;k++)
       {
       int x=0,v=0,s=0;
       v=v+k;
       s=s+v*j;
       x=x+s;
       cout<<v<<"\n"<<s<<"\n"<<x;
       }
       t=t+1;
}
end=clock();
cout<<"\n"<<"the time was  "<<(end-start)/CLK_TCK;
cout << "\n"<< "end="<< end;
while (getchar() !='q');
return 0;
}
```

يقوم البرنامج بعمليات حسابية رياضية وعمليات طباعة وفقاً لتتابع الحلقات المتداخلة، ومن خلال التابع clock() يطبع البرنامج زمن التنفيذ من مرتبة الثانية وهنا (CLK_TCK = 4.615385 Sec).

البرنامج نفسه ممكن أن يكتب بلغة الباسكال أو الدلفي كما يلي:

```
program test;
uses dos;
var hr,min,sec,hundredths:word;
   eend_sec,eend_hun,eend_min:word;
   i,j,k,v,s,x,t:integer;
begin
gettime (hr,min,sec,hundredths);
while (t<=7) do
begin
    for i:=1 to 5 do
    for j:=1 to 5 do
    for k:=1 to 5 do
    begin
        v:=v+k;
```

```
            s:=s+v*j;
            x:=x+s;
            writeln(v);
            writeln(s);
            writeln(x);
        end;
        t:=t+1;
    end;
    writeln(hr,':',min,':',sec,':',hundredths);
    gettime(hr,eend_min,eend_sec,eend_hun);
    write(hr,':',eend_min,':',eend_sec,':',eend_hun);
    end.
```

يمكننا من خلال الإجرائية GETTIME أو GetTickCount حساب زمن التنفيذ، وقد بلغت قيمته: (11sec, 15%).

نلاحظ أن هناك فرق واضح في زمن التنفيذ بين اللغتين، ولكن بشكل عام فإن أساس المعيار هو التنفيذ بزمن أقل من زمن محدد مسبقاً، لذلك سيكون الزمن الأكبر المقبول للتنفيذ محدد مسبقاً هنا في التطبيق لم نحدد زمن للمقارنة قبل البدء بالاختبار لذلك لا تظهر رسائل خطأ.

الاستنتاجات والتوصيات:

1- إن اختيار اللغة بشكل أساسي ذو أهمية ويعتبر جزءاً من تطوير الأنظمة كما هو الحال في اختيار التجهيزات الصلبة "hardware" .

2- من أجل مهمات برمجة أو بيئات قياسية، فإن اختيار اللغة يجب أن يعتمد على تحليل دقيق يستندِ إلى طرق كمية وتحليلية.

3- الطرق التي قمنا بتحليلها في هذ البحث تمثل منهجية عامة لاختيار اللغة والمعالج، وقد بينا من خلال الدراسة التطبيقية تأثير الإختيارات وسرعة الإنجاز.

4- تبين هذه المنهجية الدمج بين مفهومين اثنين (طريقة دلفي " Delphi method"، والبرنامج القياسي "benchmark program") وذلك لقياس اختيار اللغة (أي لتحويل خيارات اللغة إلى كميات مقاسه).

5- يمكن أن يتم تطوير عملية قياس الأداء لاختيار لغة البرمجة الصناعية، وذلك في التطبيقات الخاصة بالمتحكمات المصغرة.

المراجع:

[1]. STIVEN MUCHNICK, - *Advanced Compiler Design implementation.* Academic press, USA. 1997

[2]. ANDREW W, - *Appel Modern Compiler in java implementation.* Cambridge University press, UK. 2000

[3]. FRIEDMAN DAVID P., WAND Mitchell, HAYNES T.CHRISTOFER, -E*ssential of programming language Friedman.* MITpress and MLgrawHill. 416 pages, 2001. 2nd edition

[4].*Theories of Programming Languages*- Page 225 by John C. Reynolds - 512 pages, 1998

[5]. http://www.awaretek.com/atesterea.html(A Program to Help You Choose a Programming Language), 2008.10.16.

[6]. http://www.scriptol.org/csharp.html 2008.10.16.

[7]. http://www.scriptol.org/pascal.html 2008.10.16.

[8]. http://www.digitalmars.com/, 2008.10.16

[9]. https://lists.inf.ethz.ch/pipermail/oberon/2002/002154.html, 2008.10.16