مجلة جامعة تشرين للدراسات و البحوث العلمية _ سلسلة العلوم الهندسية المجلد (28) العدد (2) 2006
Tishreen University Journal for Studies and Scientific Research- Engineering Science Series Vol. (28) No (2) 2006

المترجم المبرمج

الدكتور جبر حنا*

(قبل للنشر في 2006/9/14)

□ الملخّص □

عندما يطور المبرمج برنامجا فإنه كثيراً ما يستخدم نماذج لخوارزميات مطورة مسبقا. هذا يفسر لماذا الأشخاص ذوي الخبرة الأكثر يبرمجون بشكل أسرع. كما يكتسب المبرمج خبرة أكثر، فإنه يقوم بتطوير تعريف معمارية البرنامج بشكل أسرع. ولكنه يقضي نفس المدة الزمنية في عمل كود لخوارزميات مألوفة عند عمل كود لخوارزميات جديدة. يوجد طريقة واحدة تمكننا من تلافي هذا التكرار وضياع المصادر هو استخدام دوال فرعية subroutines وتعريفات ميكروية macro definitions لتخطيط الخوارزمية. ومع ذلك فإن تخطيط الخوارزميات هو أمر قاسي جداً. ويعمل فقط مع الخوارزميات التي تكون متساوية تقريباً لولا المتغيرات. والتخطيط الأفضل للخوارزميات ممكن عن طريق برمجة المترجم لذلك لا بد من برمجة التطبيق نفسه. فيبرمج المترجم للسماح لنفس المحاولة بأن تستخدم في حالات مستقبلية مشابهه.

الكلمات المفتاحية: برمجة المترجم، الفهارس، التعابير القواعدية، الدوال الفرعية، جدول الإعراب، محلل المفردات، محلل القواعد، محلل المعاني، توليد الشفرة الهدف.

*أستاذ مساعد في قسم الحاسبات والتحكم الآلي بكلية الهندسة الميكانيكية والكهربائية-جامعة تشرين – اللاذقية – سوريا.

مجلة جامعة تشرين للدراسات و البحوث العلمية _ سلسلة العلوم الهندسية المجلد (28) العدد (2) 2006
Tishreen University Journal for Studies and Scientific Research- Engineering Science Series Vol. (28) No (2) 2006

Programming Compiler

Dr. Jabr Hanna*

(Accepted 14/9/2006)

□ ABSTRACT □

When a programmer develops a program, he frequently uses previously developed patterns of algorithms. These explain why people with more experience program faster. As the programmer gains more experience, he develops the program architecture definition faster. But he spends the same amount of time coding familiar algorithms as he would coding a new algorithm. One way to avoid this repetition and waste of resources is to use subroutines and macro definitions to map the algorithms. However, mapping algorithms this way is too rigid, and works only with algorithms that are almost equal except for variables. A much better mapping of algorithms is possible by preprogramming the compile, so instead of programming the application itself, we program the compiler to allow the same effort to be used in future similar situations.

Key words: Compiler programming, Indexes, Syntactic expressions, Subroutines, Parsing table, Lexical analyzer, Syntax analyzer, Semantic analyzer, Target code generation.

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

مقدمة:

أظهر المبرمجون تطويراً شاملاً للمترجمات، ويتطلع علماء الحاسوب في السنوات الأخيرة للغة أفضل تسمح بتمثيل أسرع وأسهل للخوارزميات، هذه الخوارزميات يجب أن تعيد بناء ملخص برمجي من خلال النظرة على هذا التمثيل في لغة البرمجة.

ومع تطور مفهوم البرمجة تطورت أيضاً مناهج البحث المعقدة والمفيدة، و يضم هذا أيضاً برمجة النظم والبرمجة المعيارية والبرمجة والتصميم بطريقة top-down ولغات البرمجة عالية المستوى ومولدات المترجم compiler generators و مولدات التقارير report generators ومولدات استرجاع البيانات.

في مقابل هذه التحسينات ما يزال نشاط البرمجة في مرحلة بدائية، عندما نأخذ بالحسبان سرعة وتكلفة حواسيب هذه الأيام. يجب أن لا يعمل المبرمجون في المستوى الحالي من التفصيل في زيادة الفهارس indexes وتحريك الحقول وتعديل القوائم وفحص نهاية الملف بعد كل عملية قراءة أو النتيجة بعد كل عملية إدخال أو إخراج. نفترض أن الحاسوب يجب أن يمتلك القابلية لتطوير أداءه مع الخبرة، أكثر من ذلك فإن عملية برمجتنا له يجب أن تصبح أبسط. لإنجاز هذا يجب أن نبرمج المترجم بالطريقة التي نتصرف بها نحن عندما نلاقي مهمة مشابهه لمهمة أخرى قمنا بإنجازها. الحاسوب يعرف مسبقا الكيفية التي سيعمل بها بعد أن تقوم بتعريف خصائص مثل هذه المشكلة.

ما أنويه هنا ليس لغة لكنه بالأحرى مفهوم المترجم الذي سيمكن المبرمجين من إضافة تعليمات جديدة وتعليمات ميكروية للغة ما حسب حاجاتهم الفردية. هذا ربما يتضمن توليد تقرير report generation والتحقق من الحقول والبحث وعمليات نظم البيانات أو دمج ومطابقة الملفات.

هدف البحث:

عندما نواجه مهمة ما. عادة نلاحظ بعض مقاييس التشابه بين المهمة الجديدة ومهمة أخرى أنجزت سابقا. ربما يكون سهلا أن نكيف حل المهمة المعروف للمشكلة الجديدة الذي يبرز مفهوم عام لحل المشاكل المشابهة.

درست الطرق الملخصة حتى الآن اظهار مسؤول للتشغيل ومع ذلك لم يتم عمل الكثير لتلخيص أداة خاصة في الروتين اليومي للمبرمجين. المترجم المبرمج استطاع أن يكون خطوة اولى في تطوير أداة برمجية جديدة تعتمد على المخطط. قد يبرمج المترجم ثم يستخدم عند الحاجة من قبل المبرمجين عندما يحتاجون لتطوير مهام مشابهة. هذا سيجعل عمل المبرمج أسهل و أضمن واقتصادي بشكل أكبر عن طريق إلغاء التكرار.

حتى الآن الأداة الوحيدة لتطبيق المخططات للخوارزميات المهمة هي الدوال الفرعية subroutines (البرامج الفرعية subprograms او الاجراءات procedures). هذه الأدوات بدائية وغير مرنة لتطبيق مخطط- تلخيص ما. مثلا هي غير عملية لتطبيق خوارزمية البحث الثنائي باستخدام الدالة الفرعية لإيجاد عنصر ما في قائمة متصلة، أو عنصر في مصفوفة أحادية البعد أو نقطة تقاطع intersection لاقترانين.

وفيما يلي بعض محددات استخدام الدوال الفرعية لتطبيق مخطط لملخص ما: [1]

1- تعمل الدوال الفرعية عادة مع متغيرات محددة الاشارة لذلك يتم عمل الدوال الفرعية لحل المشكلة بوساطة الخوارزمية المعطاة لمجموعة من المتغيرات الحقيقية.

2- توثيق استدعاء الدالة الفرعية يحدد اسم الدالة الفرعية. ولا يوجد أي شئ عن العملية، أو العلاقة بين الثوابت arguments، على الجانب الآخر فإن التعبير القواعدي syntactic expression للتعليمة المبرمجة في المترجم موثق تماماً.

3- التخطيط للخوارزميات باستخدام الدوال الفرعية مرتبط مع استبدال المتغيرات في الإجراء الثابت. التخطيط الأفضل للخوارزميات هو الذي يولد فعليا بشكل آلي الإجراء الضروري لكل تعليمة. هذا يعطي للمخطط طيف واسع من التطبيقات، مثلاً إجراء فرعي لإيجاد القيمة العظمى ما بين A,B,C حيث D=MAX (A,B,C) سيعمل لإيجاد القيمة العظمى للمتغيرات الثلاث A,B,C. الإجراء الفرعي يجب أن يبرمج بشكل خاص ليعطي كل الاحتمالات. في الجانب الآخر نحن سنقوم ببرمجة المترجم بحيث يأخذ تعليمة مثل:

MAX of variable-name-1[variable-name-2 …]
DEPOSIT IN [variable-name-3]

هذا هو مخطط كل الخوارزميات التي تجد القيمة العظمى من بين مجموعة متغيرات.

4- من فكرة ملخص خوارزمية ما يمكن أن نكتب عدة دوال او إجراءات فرعية لحل المشاكل نفسها وكل إجراء فرعي يمكن استخدامه لمجموعات مختلفة من المتغيرات. بينما يمكن استخدام الدوال الفرعية لحل مشاكل محددة بمدخلات مختلفة. إن المترجم المبرمج يسمح باستخدام فكرة لملخص خوارزمية ما لحل عدة مشاكل مختلفة (انظر الشكل 1) [2]

ولمعرفة الكيفية التي يعمل بها المترجم المبرمج. افترض أنه في شركة ما يتعامل المبرمجون بشكل متكرر مع screen formatting مع بيانات مدخلة وفي كل وقت يحتاجون إلى أخذ بيانات من الطرفيات. يحتاجون لبرمجة خوارزمية مشابهه لهذه الخوارزمية: [3]

```
Flag: = FALSE
Do While (Flag = FALSE )
|  PUT cursor in the desired position
|  DISPLAY prompt
|  ACCEPT data
|  IF valid data
|    |  Flag: = TRUE
ELSE
|     | WRITE error message
|     | RING the bell
|  ENDIF
ENDDO
```

بعض المبرمجين الذين يتعاملون مع بيانات مكتسبة لديهم برامج ضخمة تحتوي على المخطط نفسه مكرر لـ30 الى 40 مرة. (إذا رغب المبرمج فإنه يكتب برنامج بحيث إنه عندما تكون الشاشة ممتلئة فإنه يسأل الشخص الذي يقوم بعملية الطباعة إذا كان يرغب بتعديل أي من البيانات المكتوبة قبل أن يتم إدخالها في ملف الحاسوب، ثم يعود ويقوم بقبول طلب الشخص الذي يقوم بالطباعة وهكذا).

لتلافي مثل هذا الإزعاج. فإن الشركة تقوم بتطوير نظام لإدارة الشاشة. ومع ذلك فالحل الأرخص والأسهل للاستخدام والقابل للنقل هو برمجة مترجم يقبل التعليمة الآتية: [3]

```
ACCEPTSCREEN
X  DOMAIN( value1, value2  TO  value3, value4 …)
PROMPT  " 1.type  x:",
Y  VALIDPROCEDURE
BEGIN
----
   validfield: = TRUE
----
    validfield: = FALSE
----
   END
   PROMPT "2.type y:",
   w DOMAIN ……,
   v  PROMPT "4.Type v",
   z  VALIDPROCEDURE
        BEGIN
            ----
   END
   PROMPT "5.Type z";

SYNTAX

<Instruction>: = <accept screen>;
                 | BEGIN  <instruction list> END
                | . ....
<Accept screen>: =ACCEPTSCREEN  <screen list>
<Screen list.> : =  <Variable> DOMAIN
                        (< Domain list >)
            PROMPT < Literal >
                        |< Variable > VALIDPROCEDURE
              <Instruction> PROMPT <Literal>
                        |<Variable> PROMPT <Literal>
<Domain list>   : =  < Domain elem. >
            |<Domain list>,<Domainelem.>
<Domain elem.>: =    <Expression >
          |<Expression>TO<Expression>
```

ميزات أخرى مثل صحة البيانات والمحاذاة لليمين أو اليسار. .. الخ سيتم إضافتها لهذه التعليمة.

وفيما يتعلق بالمترجم. وحتى يقوم المبرمج بتطبيق مثل هذه التعليمة فإنه يجب أن يتبع الإجراء التالي: [4]

Parsing Table

1- يسير بناء على سياق التعليمة حتى لا يبقى هناك أي تضارب مع سياق اللغة.

2- إعادة توليد جداول الإعرابParsing Table وجداول المسحScanning tables مع البرنامج المخصص.

3- كتابة Module لكل نتاج من نتاجات سياق التعليمة لتوليد الكود الملائم.

4- فحص وعمل Debug لكل module.

5- ربط instruction modules مع module المترجم.

6- فحص المترجم ككل متكامل.

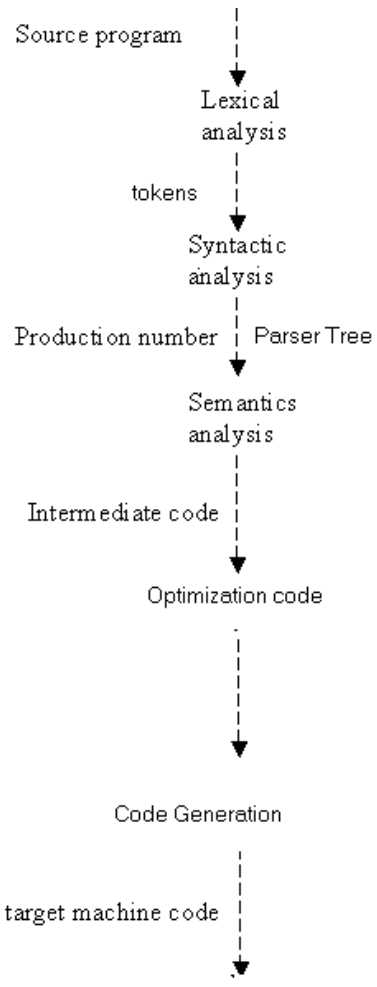
7- تنصيب صورة المترجم.

نستطيع أن نفرض أن محلل المفردات Syntax Analyzer لكل اللغات هو واحد فيما عدا ما يخص الكلمات المفتاحية الخاصة وتمييز الإشارات. في الحقيقة معظم مولدات الماسحات scanner generators تنتج دوال تحليل مفردات ثابته تختلف فقط في قائمة تمييز الكلمات المفتاحية. وهذا مدعوم من قبل المستخدم.

في الجانب الآخر تم إثبات أن محللات المفردات الأوتوماتية يمكن أن تبنى لتراكيب المفردات المحددة بوساطة التعابير المنتظمة Regular Expressions.

و تم تطوير عدد من مولدات الإعراب لتحليل القواعد للغات البرمجة في تدوين نحو السياق الحر Context Free Grammar، لمثل مولدات الإعراب هذه ومن قواعد التصميم الملائمة يمكن إنشاء معرب آلي كفؤ.

وتطوير مولدات كود جيد هو أمر صعب، دون التعامل مع تفاصيل الآلة الخاصة، والتركيب التقليدي للمترجمات هو التالي: [5]



تم تطويره للمرحلة التي تستطيع بها بناء مترجم بوساطة التعريف الدقيق لتنظيم المفردة. وقواعد اللغة وبوساطة كتابة نماذج إنتاج الكود (بالاعتماد على التركيب العام للبيانات).

من الجملة أعلاه سنقوم ببناء مترجم مبرمج ومحلل مفردات Lexical Analyzer بالاعتماد على الجدول المشتق Table driven أو الماسح scanner ومحلل المفردات. محلل القواعد Syntax Analyzer المشتق من الجدول او المعرب وإمكانية كتابة النماذج modules لتوليد الكود لكل نتاج.

لبرمجة المترجم يجب أن يحدد المبرمج أولا التشابكات في قواعد التعليمة (الشكل 2) [2]. وعندما تكون قواعد التعليمة صحيحة وموجودة في اللغة. يقوم مبرمج المترجم بالعمل مع الرموز الجديدة لدمجهم في بناء المفردات في اللغة. لعمل هذا فان عليه العودة إلى الخطوة السابقة (الشكل 3) [2].

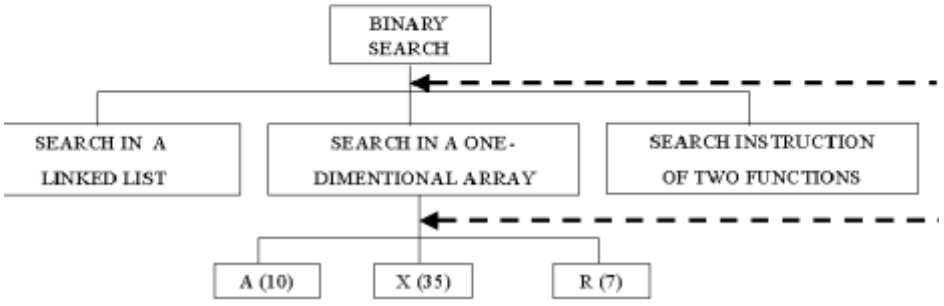
لتحليل المعاني Semantic Analyses فان المترجم يجب أن يبرمج النماذج modules التي ستولد الكود للتعليمة الجديدة (مرة لكل نتاج). ربما عليه الرجوع إلى الخلف للخطوة الأولى لإعادة تصميم التعريف القواعدي للتعليمة لملائمة النتاجات لإنتاج الكود بالتسلسل المطلوب.

إعادة تغطية الأخطاء القواعدية recovery of syntax errors تم إنجازها بواسطة إزالة التعليمات التي تحدث بها الأخطاء القواعدية.

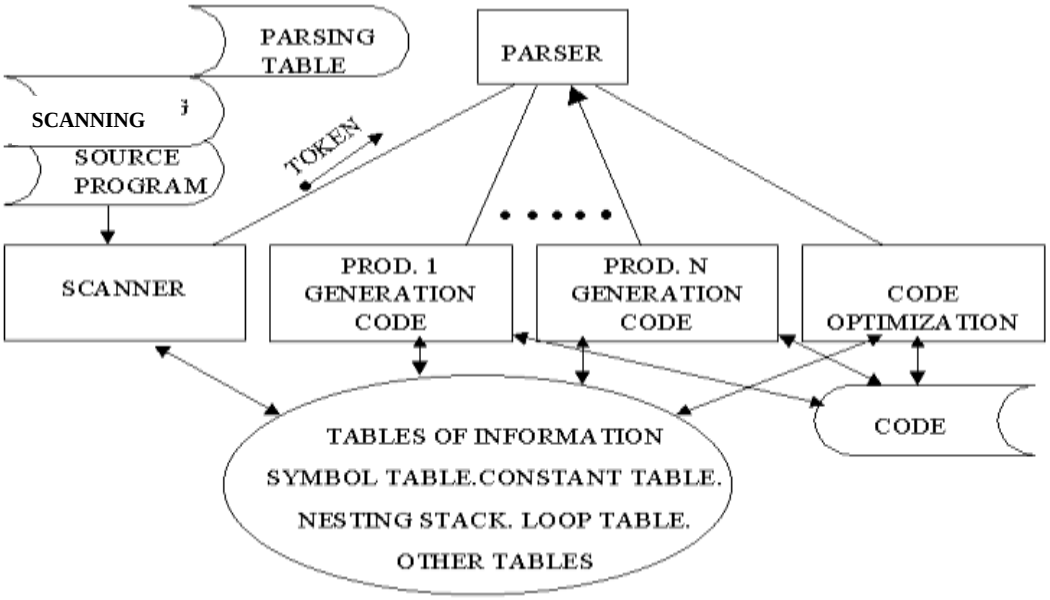
النتائج:

بما أن التطبيقات لها خصائص مختلفة والمبرمجين لديهم أفضليات مختلفة فيما يتعلق بمميزات اللغة. فالمترجم المبرمج سيجعل بالإمكان إنتاج لغة جديدة خاصة بالسماح للمبرمجين بالتشارك بوظائف جديدة أو وظائف متخصصة ضمن لغة موجودة، وهذا سيجعل البرامج أقصر وأقل عرضة للأخطاء، أي يمكننا تطوير اللغة مع المترجم المبرمج.

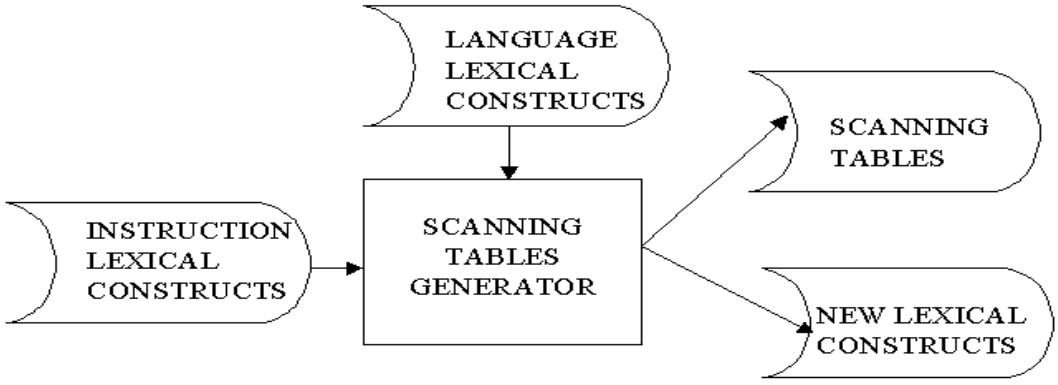
الأشكال المرفقة بالمقالة:

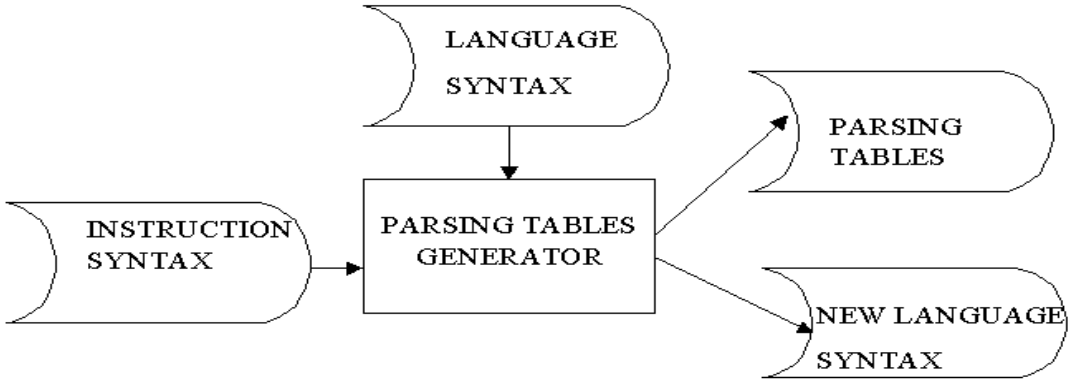


الشكل (1): مثال على تركيب حل المشكلة باستخدام المترجم المبرمج، المستوى العلوي يمثل تلخيص للخوارزمية، والمستوى الثاني يمثل المشكلة الموصوفة والمستوى الثالث يمثل مجموعة المتغيرات.



الشكل (2): تركيب المترجم بعد حل مشكلة التشابك





الشكل (3): إعراب التعليمة يحتوي على قواعد اللغة، عندما يتم حل التشابكات. المترجم القابل للبرمجة يعمل مع رموز جديدة ليكاملهم مع تراكيب المفردة.

المراجع:

1.WATT,D.,BROWN,D. *Compilers and Interpreters*, Prentice Hall,2000.

2.LOUDEN,K.C. *Compiler Construction: Principles and practice*, brooks cole, 1997.

3.DERSHOWIZ,N. the *evolution of program: program abstraction and instantiation*, Fifth Int'l Conf. Softwre Eng, 1981.

4.MARK,R.*Writing Compilers and Interpreters*, 1996

5.AHO,A.V.,Ullman,J.D. *Compiler's: Principles, Techniques, And Tools*, Prentice Hall, 2003.