الجلسة الثالثة

بناء مترجم يتضمن عمليات scanning و Parsing -1-

الهدف من الجلسة

- التعريف بكيفية كتابة ملف Parser Description File
- كتابة ملف Scanner Description لعملية مسح كود مصدري والتحقق من مطابقته للقوالب.

مستلزمات الجلسة

- حاسب بمواصفات دنيا RAM: 1 GB, CPU: 1.6 GHz, Windows 10 OS 64 bit
- Turbo c++/ DevC++
- LEX & BISON tools

الخلاصة والنتائج:

يفترض عند نهاية الجلسة:

تمكن الطالب من فهم بنية ملف وصف المعرب وبناء مترجم يتضمن مرحلتي scanning & Parsing.

1-بنية ملف وصف المعرب Parser

1) يتضمن الجزء الأول من ملف وصف المعرب التعريفات وتوضع ضمن قوسين %{ }% للدلالة على أنها تعليمات بلغة الـ C.

```
%{
.....
%}
```

يتضمن الجزء الأول كذلك التصريح عن جميع المفردات tokens التي سيتعامل معها المعرب ضمن النحو الخاص باللغة، ويتم التصريح عنها بعد الكلمة المفتاحية %token.

%token NUMBER PLUS MINUS MULT DIVS POWER PARENTHESISL PARENTHESISR

ملاحظة هامة:
حتى يتمكن الماسح scanner من مسح تسلسل الدخل وتحويله إلى مفردات tokens فهو بحاجة إلى معرفة ماهية هذه المفردات، وحتى يتمكن من ذلك فإن المعرب وبعد تنفيذ ملف الوصف الخاص به ينتج ملف رأسي h. يتضمن التصريحات الخاصة بالمفردات التي يتعامل معها ولذا نحن بحاجة لأن نقوم بتضمين هذا الملف ضمن ملف وصف الماسح scanner وذلك في قسم التضمين الخاص به.

يتضمن القسم الثاني إضافة لذلك التصريح عن الأولويات الخاصة بالعمليات. هناك أولوية أثناء إجراء عمليات الحساب فالقسمة أقوى من الضرب والذي هو أقوى من الجمع والطرح.

2) يحتوي القسم الثاني **قواعد النحو Grammar Rules**، وكيفية استجابة المعرب لها.

باقي الملف يحتوي على بعض التوابع الضرورية لعمل المترجم مثل التابع الرئيسي **main** والذي ستبدأ منه عملية الترجمة عندما يتم ترجمة الملف باستخدام مترجم GCC.

2- بعض التوابع الهامة في ملفي وصف الماسح والمعرب:

يستخدم المعرب parser التوابع التالية:

()yywrap: يحدد من خلاله متى تنتهي عملية المسح أي متى تنتهي سلسلة الدخل ويتوقف عمل الماسح ليبدأ بعدها عمل المعرب أي تبدأ مرحلة التحليل القواعدي. هذا التابع يعرف ضمن جسم ملف وصف المعرب بعد نهاية قسم القواعد. يستدعى هذا التابع من قبل الماسح ويعيد قيمة 1 ليدل على انتهاء سلسلة الدخل. يتم وضع هذا التابع بعد قسم القواعد ويمكن أن يكتب كمايلي:

```
int yywrap()
```

```
{
return 1;
}
```

yyerror(char *s): تابع الخطأ وينفذه المعرب في حال حدوث أي خطأ في إعراب السلسلة. أما عن البارامتر الذي يخذه كوسيط (char *s) فهو يمثل رسالة الخطأ التي سيتم طباعتها عند استدعائه من قبل التابع الرئيسي.

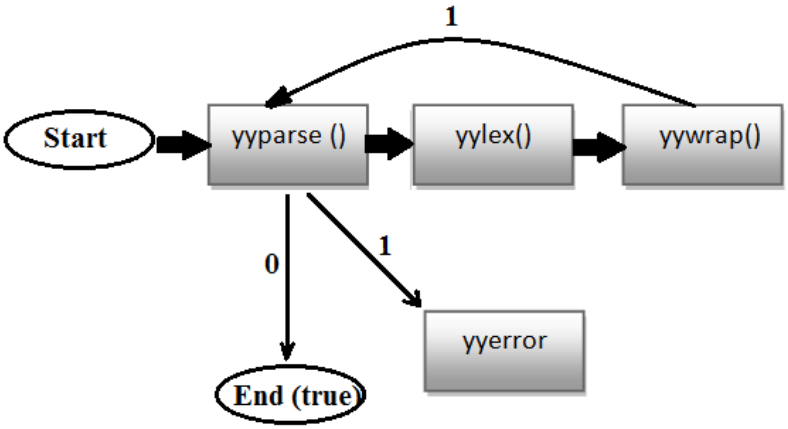
```
int yyerror (char *s)
{
printf("%s\n",s);
}
```

التابع الرئيسي main: هو التابع الذي ستبدأ منه عملية التنفيذ عند ترجمة ملف المعرب باستخدام لغة الـ C، ويجب أن يتم ضمنه مناداة التابع yyparse().

تابع الإعراب yyparse(): وهو التابع الذي تتم مناداته ضمن التابع الرئيسي للبدء بعملية الإعراب حيث يتم عند مناداته استدعاء التابع yylex() الذي يبدأ بمسح سلسلة الدخل لتحديد المفردات ثم يعيدها للمعرب، الذي ينادي باستمرار تابع yywrap() والذي سيعيد له قيمة 1 عندما ينتهي عمل الماسح بانتهاء سلسلة الدخل، وهنا يبدأ دور المعرب الذي يتأكد من البناء القواعدي لسلسلة الدخل التي قدمها له الماسح مع قيم كل منها عبر المتغير yylval. في حال كان السلسلة اللفظية صحيحة قواعديا فإن التابع yyparse() يعيد قيمة 0 للتابع الرئيسي ليدل على أن الترجمة تمت بدون أخطاء.

في حال لم تكن السلسلة اللفظية صحيحة قواعدياً وفق قواعد المعرب، فإن التابع yyparse() يعيد قيمة 1 للتابع الرئيس والذي يقوم بدوره باستدعاء تابع الخطأ yyerror() لينفذ تعليماته (مثلاً طباعة رسالة خطأ على الشاشة).

يوضح الشكل (1) مخطط الحالة لسير عملية الترجمة من منظور محللي المفردات والقواعد.



الشكل (1) مخطط الحالة لسير عملية الترجمة من منظور الماسح والمعرب

الجانب العملي:

تنفيذ مترجم يقبل عبارات العمليات الحسابية والمنطقية مثل: 20*50/70^2 ويتعرف عليها ثم يقوم بحساب النتيجة وطباعتها على الخرج.

بداية نكتب ملف Scanner Description File:

```
%{
#include<stdlib.h>
#include "y.tab.h"
%}
blanc           [ \t]+
number          [0-9]
entire          {number}+
%%
{blanc}
{entire}        {
                yylval=atoi(yytext);
                return(NUMBER);
                }
"+"             return(PLUS);
"*"             return(MULT);
"-"             return(MINUS);
"/"             return(DIVS);
"^"             return(POWER);
"("             return(PARENTHESISL);
")"             return(PARENTHESISR);
"="             return(EQUAL);
 \n             {}
%%
```

تضمين ملف تعريفات التوكين Tokens ضمن ملف وصف scanner

إذا صادف الماسح مثلاً فراغ أو أكثر ضمن رموز الدخل فإنه لن يقوم بأي شيء (أي سيتجاهله) وفقاً للقاعدة الأولى.

القاعدة الثانية {entire}: لدى مصادفة الماسح لعدد ضمن رموز الدخل سيعيد للمعرب parser المفردة token المسماة NUMBER، وسيعيد له كذلك قيمة هذه المفردة عبر المتغير yylval. بما أن الماسح يمسح رموز الدخل ويضعها في المصفوفة yytext فإن قيمها ستكون من النوع string لذا فهو بحاجة لتحويلها إلى النمط الصحيح integer وللقيام بذلك يستخدم الدالة atoi الموجودة في المكتبة stdlib.h.

إذا صادف الماسح الرمز "/" ضمن سلسلة الدخل سيعيد مفردة token اسمها DIVS، وإذا صادف رمز "(" فسيعيد مفردة اسمها PARENTHESISR، وفي حالة صادف سطراً جديداً فلن يقوم بعمل شيء أي سيتجاهله..... وهكذا بالنسبة لبقية القواعد.

ثانياً: ملف Parser Description Files

```
%{
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "lex.yy.c"

%}
%token NUMBER PLUS MINUS MULT DIVS POWER PARENTHESISL PARENTHESISR EQUAL
%left PLUS MINUS
%left MULT DIVS
%left NEG
%right POWER
%start Input
```

Stdlib for ATOI, ATOF
Stdio for printf
Math for pow
Conio for getch

تضمين ملف الماسح Scanner ضمن ملف وصف المعرب لأن Parser هو من يقود عملية الترجمة ويستدعي الماسح

تعريفات المفردات Tokens Identifications سيتم بناء ملف h. بالاعتماد على هذه التعريفات

تحديد أولويات العمليات. أثناء إجراء عمليات الحساب فالقسمة أقوى من الضرب والذي هو أقوى من الجمع والطرح ولذلك تم إضافة سطور الأولويات التالية، مع ملاحظة أن %left وَ %right هي كلمات محجوزة تعبر عن الأولويات واتجاه حسابها

```
%%
Input:
|Input Line
;
Line :EQUAL
|Exp EQUAL {printf("The result is=%d\n",$1);
}
;
Exp :NUMBER {$$=$1;printf("Number %d\n",$$);}
|Exp PLUS Exp {$$=$1+$3;printf(" PLUS Op.\n");}
|Exp MINUS Exp {$$=$1-$3;printf(" MINUS Op.\n");}
|Exp MULT Exp {$$=$1*$3;printf(" MULT Op.\n");}
|Exp DIVS Exp {$$=$1/$3;printf(" DIV Op.\n");}
|MINUS Exp  {$$=-$2;printf(" Negative Op.\n");}
|Exp POWER Exp {$$=pow($1,$3);printf(" Power Op.\n");}
|PARENTHESISL Exp PARENTHESISR {$$=$2;};
%%
int yyerror (char *s)
{
printf("%s\n",s);
}
int yywrap(){
return 1;
}
main()
{
yyparse();
getch();
}
```

قواعد النحو Grammar rules

كل ما هو ضمن قوسين Semantic Actions يمكن من خلالها تنفيذ أي عملية مباشرة بلغة C++ ونستخدم فيها المؤشرات للوصول لقيم المتحولات لنتمكن من التعامل معها

استدعاء المعرب Parser

لمنع إغلاق شاشة DOS عند حدوث خطأ حتى يتثنى لنا معرفة الخطأ

معلومات مساعدة لفهم الأكواد:

جدول (1) الأولويات في المعرب	
Priority الأولوية	**Instruction التعليمة**
الجمع والطرح لهما الأولوية الأقل وتجري العمليات من اليسار	%left PLUS MINUS
القسمة والضرب لهما أولوية أعلى، ونبدأ أثناء الحساب من اليسار	%left MULT DIVS
النفي أقوى مما سبقه (أقوى من الضرب والقسمة والجمع والطرح)، الحساب من اليسار	%left NEG
الرفع لقوة أقوى من كل ما سبق، لكن نبدأ بالحساب من اليمين	%right POWER

بالنسبة للقواعد والـ Semantic Actions

القاعدة الأولى هي ;| input : input line وهي نقطة البداية حيث تعطي input إما فراغاً أو input line .

القاعدة الثانية line إما تعطينا EQUAL (وهي الحرف "=" على مستوى وصف لغة LEX) أو Exp EQUAL وهنا نكتب نتيجة العملية الحسابية باستعمال printf("=%d\n",$1) وهذه التعليمة تعني طباعة قيمة العدد المعنون بـ $1 وهويمثل قيمة EXP ولكن ماذا يعني الرمز $1؟

أثناء التحقق يمكننا أن نأخذ وأن نعطي قيم الرموز و الـ Tokens التي نجدها في طريقنا وهنا سنكتب على الشاشة قيمة Exp وهي معرفة بـ $1, أما قيمة EQUAL إذا أردنا استعمالها فهي $2 وهكذا.

في كل قاعدة هناك نصف أيمن ونصف أيسر ولإسناد قيم أو قراءة قيم الرموز المكونة للقاعدة فإننا نستعمل $x:

EXP –> EXP PLUS EXP

$$ $1 $2 $3

أي يتم ترميز EXP النتيجة بـ $$ أما EXP على الطرف الأيمن للقاعدة ترمز بـ $1 أما الـ EXP الثالثة ترمز $3 أما الـ TOKEN التي تتوسط القاعدة والممثلة بـ PLUS فإنها ترمز بـ $2.

وطبعاً استخدام الرموز السابقة تابع للتحليل المعنوي, فالمحلل القواعدي لايهتم بالقيم.

ننتقل إلى القاعدة الثالثة الخاصة بـ Exp وهي تعطينا واحداً من ثمانية خيارات:

- في حالة Exp–> NUMBER فالأمر بسيط نعطي قيمة الرقم إلى Exp باستعمال ;$$=$1 .

- في حالة Exp PLUS Exp <-Exp بما أن Exp التي على يسار القاعدة تأخذ قيمة مجموع Exp التي على يمين القاعدة ويكون لدينا $$=$1+$3 ونأخذ الشيء نفسه للقواعد المماثلة.

- في حالة Exp MINUS Exp <-Exp نقصد هنا حالة النفي مثل 112-وليست عملية الطرح وهي لها أولوية قصوى فإذا وجدنا 5/6- فإننا نحسب 5- أولا ولهذا نعطيها أولوية NEG باستعمال التعليمة %prec NEG وهي كلمة مفتاحية تستخدم لتحديد الأولويات Preceding، لكن يمكن حذف هذا الأمر كوننا قمنا سابقاً بإعطاء NEG أولوية أعلى من الطرح لذلك تعد التعليمة هنا زائدة أما في حالة لم نذكر سابقاً أولوية النفي NEG فإن التعليمة %prec ضرورية، أما باقي السطر فهو يمثل هو إعطاء القيمة Exp- إلى Exp التي على اليسار باستعمال $$ $2 =.

- بالنسبة للحالة Exp POWER Exp <-Exp فهي الرفع إلى قوة 3^2 = 9 وستحسب باستخدام الدالة pow الموجودة في المكتبة math.h كما يلي: $$ = pow($1,$3);.

انتهت الجلسة – د. علي ميا ، م. رشا شباني