الجلسة الحادية عشر

التحليل المعنوي

Semantic Analysis -2-

الهدف من الجلسة

- التعرف على مفهوم التحليل المعنوي وبناء جداول الرموز.
- التعرف على كيفية ضبط وكشف والتخلص من الأخطاء المعنوية ضمن تسلسل الدخل.

مستلزمات الجلسة

- حاسب بمواصفات دنيا RAM: 1 GB, CPU: 1.6 GHz, Windows 7 OS 32 bit
- Turbo c++
- LEX & BISON tools
- تذكرة باللوائح المترابطة التي تم التعرف عليها في مادة بنى المعطيات.

خطوات العمل

- كيفية كشف الأخطاء المعنوية.
- إنشاء جدول الرموز Symbol Table.
- أولاً تعليمات تابع put_sym لوضع الرموز ضمن جدول الرموز.
- ثانياً تعليمات تابع get_sym للحصول على الرموز من جدول الرموز.

الخلاصة والنتائج:

يفترض عند نهاية الجلسة:

- تمكن الطالب من فهم مبدأ التحليل المعنوي وكيفية تعديل ملفات الماسح والمعرب للتعامل مع ذلك.
- تمكن الطالب من فهم كيفية إنشاء جدول الرموز.

1.1 مرحلة التحليل المعنوي:

1.1.1 ملف توصيف الماسح scanner:

نستخدم نفس ملف الماسح Scanner الخاص بالمحاضرة السابقة كما هو:

```
%{
#include<string.h>
#include<stdlib.h>
#include "sem.h"
int line=1;
%}
alpha [a-zA-Z]
digit [0-9]+
rdigit {digit}\.{digit}
blanc [ \t]+
ident {alpha}({alpha}|{digit})*
a [aA]
b [bB]
c [cC]
d [dD]
e [eE]
f [fF]
g [gG]
h [hH]
i [iI]
j [jJ]
k [kK]
l [lL]
m [mM]
n [nN]
o [oO]
p [pP]
q [qQ]
```

```
r [rR]
s [sS]
t [tT]
u [uU]
v [vV]
w [wW]
x [xX]
y [yY]
z [zZ]
%%
{blanc}
\n { line++; }
{i}{n}{t} return(DEC_INT);
{r}{e}{a}{l} return(DEC_REAL);
{c}{h}{a}{r} return(DEC_CHAR);
{c}{h}{a}{i}{n} return(DEC_CHAIN);
"=" return EQ;
"\"."\"" return(CHAR);
"\"(.)+"\"" return(CHAIN);
";" return(SEMICOLON);
{digit} {
yylval.Tint= atoi(yytext);
return(NUM);
}
{rdigit} {
yylval.Treal = atof(yytext);
return(REAL);
}
{ident} {
strcpy(yylval.Tstr,yytext);
return(ID);
}
```

%%

شرح بسيط:

في الكود:

```
{digit} {
yylval.Tint= atoi(yytext);
return(NUM);}
```

هنا عند التعرف على رقم digit يتم إعادة الـ token المسمى NUM كما يتم إعادة قيمة هذا الرقم في الحقل Tint. في السجل yylval. التابع atoi يحول السلسلة الموجودة في المصفوفة yytext إلى رقم ويقوم بوضعها في الحقل Tint.

بينما في مقطع البرنامج التالي:

```
{rdigit} {
yylval.Treal = atof(yytext);
return(REAL);}
```

عند التعرف على رقم حقيقي Real يتم إعادة الـ token المسمى REAL كما يتم إعادة قيمة هذا الرقم في الحقل Treal. في السجل yylval.

التابع atof يحول السلسلة الموجودة في المصفوفة yytext إلى رقم حقيقي ويقوم بوضعها في الحقل Treal.

يقود مقطع البرنامج التالي:

```
{ident} {
strcpy(yylval.Tstr,yytext);
return(ID);
}
```

عند التعرف على رقم معرف ident إلى إعادة الـ token المسمى ID كما يتم إعادة السلسلة المحرفية التي تمثل هذا المعرف في الحقل Tstr. في السجل yylval

1.1.2 ملف توصيف الـ Parser:

هنا تكمن التعديلات الرئيسية والجوهرية والتي ستمكن المعرب من كشف الأخطاء المعنوية بالاعتماد على جدول الرموز الذي تم إنشاؤه.

والنقطة الأهم الآن هي تضمين ملف جدول الرموز وهو الملف SYMB_TAB.h:

```
%{
#include <stdio.h>
```

```
#include <stdlib.h>
#include<conio.h>
#include<math.h>
#include "lex.yy.c"
#include "SYMB_TAB.h"
int errors=0;
```

والآن يتم تعريف تابع setup_sym يقوم بنداء التابع get_sym ليتحقق فيما إذا كان الرمز sym_name موجوداً ضمن جدول الرموز:
فإن كانت القيمة المعادة منه NULL فهذا يعني أن الرمز غير موجود عندها تتم إضافته باستخدام التابع put-sym، أما إن لم تكن القيمة المعادة NULL فهذا يعني أن الرمز موجود لذا يطبع التابع رسالة خطأ تفيد بأن الرمز معرف وموجود مسبقاً ولا يجوز التصريح عنه من جديد.

```
void setup_sym(char* sym_name)
{
sym_node *sym;
sym=get_sym(sym_name);
if(sym==NULL) put_sym(sym_name);
else
{
errors++;
printf("Error %d :: %s Identifier is defined previously: line%d.\n",errors,sym_name,line);
}
}
```

النقطة التالية في الرنامج هي التصريح عن تابع sym_check والذي يقوم بالتحقق من أن الرمز sym_name الممرر كوسيط له معرف أو لا ضمن جدول الرموز:

يتم ذلك من خلال مناداة التابع get_sym باستخدام الرمز sym_name كوسيط لهذا التابع فإن كانت القيمة المعادة من التابع هي NULL فهذا يعني أن الرمز غير معرف وستتم طباعة رسالة تفيد بأن الرمز غير معروف مسبقاً، ولا يجوز إسناد قيمة لمتغير غير معرف.

```
int sym_check(char* sym_name)
{
if(get_sym(sym_name)==NULL)
{
errors++;
printf("Error %d :: %s Identifier is not known: line %d.\n",errors,sym_name,line);
```

```
return 0;
}
return 1;
}
%}
```

والآن نكتب باقي ملف وصف المعرب وهي نفس تعليمات المحاضرة السابقة:

```
%union{
char Tstr[10];
int Tint;
float Treal;
}
%token DEC_INT DEC_CHAR DEC_REAL DEC_CHAIN EQ SEMICOLON
%token <Tint> NUM
%token <Treal> REAL
%token <Tstr> ID CHAR CHAIN
%right '='
%start st
%%
st:declaration expression;
declaration:
|decl_type ident SEMICOLON declaration;
decl_type:DEC_INT
|DEC_REAL
|DEC_CHAR
|DEC_CHAIN;
```

تستخدم <>لتعريف نوع الـ Token لأن لكل منها نوعاً مختلفاً ضمن السجل union.

والآن هنا وعند كل ظهور للرمز ID ضمن مرحلة التصريح عن المتغيرات، يتم تسجيل الرمز ضمن جدول الرموز فقط في حال كونه لم يكن معرفاً مسبقاً كما في التعليمات التالية الموضوعة بين قوسين والتي تمثل استدعاء للتابع setup_sym.

```
ident:ID {setup_sym($1);}
|ID ident {setup_sym($1);};
```

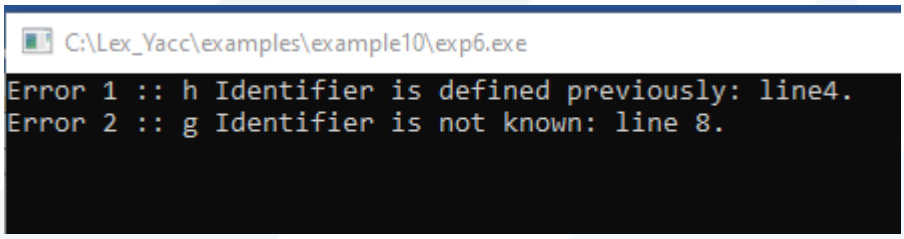
أما عند ظهور عبارة تتضمن token من النوع ID يتم التحقق فيما إذا كان الرمز معرفاً مسبقاً أو لا ويتم ذلك من خلال استدعاء التابع sym_check كمايلي:

```
expression:
|ID EQ ID expression SEMICOLON {
sym_check($1);
sym_check($3);}
|ID EQ CHAIN SEMICOLON expression {sym_check($1);}
|ID EQ CHAR SEMICOLON expression {sym_check($1);}
|ID EQ NUM SEMICOLON expression {sym_check($1);}
|ID EQ REAL SEMICOLON expression {sym_check($1);};
%%
int yyerror (char *s)
{
errors++;
printf("Error:: %d%s at line:: %d in statement:: %s\n" ,errors,s,line ,yytext);
}
int yywrap(){
return 1;
}
main()
{
clrscr();
if((yyin=fopen("input.txt","r"))==NULL)
{
printf("input.txt not found !\n");
getchar();
}
yyparse();
getchar();
}
```

والآن نختبر العبارة التالية بعد كتابتها في ملف input.txt:

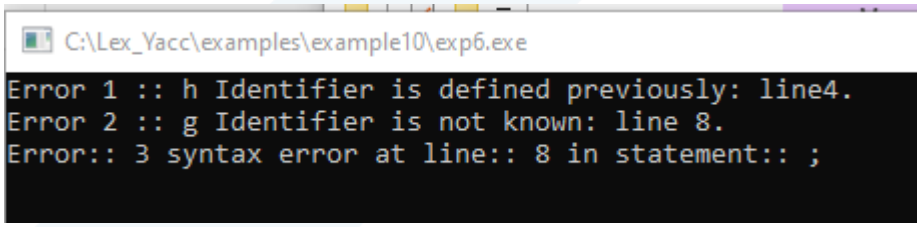
```
real x;
int y;
chain h;
```

```
int h;
x=7.9;
h="computer engineer";
h="aa";
g=2;
```



مثال آخر:

```
real x;
int y;
chain h;
int h;
x=7.9;
h="computer engineer";
h="aa";
g=2;;
```



إلى هذه النقطة تمكنا من كشف نوعين من الأخطاء فقط هي الأخطاء الناتجة عن تكرار التصريح والأخطاء الناتجة عن إسناد قيم لمتغيرات غير مصرح عنها سابقاً.

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