الجلسة السادسة

إضافة تعقب مكان الخطأ وعدد الأخطاء لمترجم لعبارات الشرط

Add location and errors count to the Compiler of If-then-else

الهدف من الجلسة

- التعرف على كيفية تحديد مكان الخطأ في تسلسل الدخل.
- التعرف على كيفية حساب عدد الأخطاء في تسلسل الدخل.

مستلزمات الجلسة

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

خطوات العمل

- بناء مترجم خاص بالشرط IF THEN ELSE
- بناء ملف وصف الماسح لعبارات الشرط
- كتابة ملف وصف المعرب لعبارات الشرط
- قواعد اللغة لعبارات الشرط
- كود ملف وصف المعرب لعبارات الشرط

الخلاصة والنتائج:

يفترض عند نهاية الجلسة:

- تمكن الطالب من تعديل بنية مترجم المحاضرة السابقة بإضافة التعليمات اللازمة لتحديد مكان الخطأ في تسلسل الدخل وسبب الخطأ وعدد الأخطاء.
- تطبيق مثال عملي على الأجهزة.

في هذه الجلسة سنتعلم كيفية الاستجابة لخطأ الدخل بتحديد مكان الخطأ في العبارة المصدرية وتحديد سبب الخطأ إضافة لحساب عدد الأخطاء.

التعديلات التي سنقوم بها هي تعديلات في بنية ملف وصف الماسح وملف وصف المعرب:

1.1 التعديلات في بنية ملف وصف الماسح Scanner Description File Modifications:

يتم بداية إضافة متحول line من النوع الصحيح Integer إلى قسم التصريح في ملف وصف الماسح ويتم تهيئته بالقيمة الابتدائية 1 كونه سيحتفظ بعدد الأسطر وتلقائياً نحن في البداية سنقرأ السطر الأول.

بعد ذلك في قسم الاستجابة للقوالب نفصل استجابة \n عن قالب الفراغات بحيث نجعل الماسح يقوم عند مصادفة \n (كلما صادف سطر جديد) بزيادة عداد الأسطر بمقدار 1 أي line++.

ويصبح شكل ملف وصف الماسح:

```
%{
#include <stdlib.h>
#include"y.tab.h"
int line=1;
%}
blank [ \t]+
alpha [A-Za-z]
digit [0-9]
%%
{blank}
\n {line++;}
if    return IF;
then    return THEN;
else    return ELSE;
{digit}+   {
yylval=atoi(yytext);
 return NUM;}
{alpha}({alpha}|{digit})* return ID;
"("    return LPAR;
")"    return RPAR;
```

```
";"     return SEMI;
"<"     return LT;
">"     return GT;
"<="    return LE;
">="    return GE;
"=="    return EQ;
"!="    return NE;
"||"    return OR;
"&&"    return AND;
"+"     return PLUS;
"*"     return MULT;
"-"     return MINUS;
"/"     return DIVS;
"="     return EQUAL ;

%%
```

1.2 التعديلات في بنية ملف وصف المعرب Parser Description File Modifications:

في ملف وصف المعرب نقوم في قسم التصريح بإضافة متحول errors ونهيئه بالقيمة الابتدائية 0 وهو سيحمل عدد الأخطاء.

وفي قسم التوابع وتحديداً تابع الخطأ نقوم بزيادة المتحول errors ونقوم بتعديل تعليمة الطباعة لتطبع رقم السطر ورقم الخطأ والعبارة التي سببت الخطأ وفق الآتي:

```
printf("Error:: %d %s at line:: %d in statement:: %s\n" ,errors ,s,line ,yytext);
```

حيث يحتفظ المتحول line بعدد الأسطر (وهو قادم من ملف الماسح) أما المتحول errors فيحتفظ بعدد الأخطاء، أما مصفوفة المسح yytext فتحتفظ بالرمز الذي توقف عنده الماسح وهو الرمز الذي سبب الخطأ.
المشكلة أن YACC مجهز ليكشف خطأ واحد فقط حيث يقوم بكشف الخطأ ثم الخروج من البرنامج دون المتابعة وبالتالي لن نستطيع كشف سوى خطأ واحد كل مرة وبعد تصحيح كل خطأ يمكن كشف الخطأ التالي.
لحل هذه المشكلة نقوم بإضافة حالة error لكل قاعدة يمكن أن يحصل فيها الخطأ فمثلاً القاعدة:

```
ST    : IF LPAR E2 RPAR THEN ST1 SEMI ELSE ST1 SEMI
      | IF LPAR E2 RPAR THEN ST1 SEMI
```

تصبح:

```
ST    : IF LPAR E2 RPAR THEN ST1 SEMI ELSE ST1 SEMI
      | IF LPAR E2 RPAR THEN ST1 SEMI
      | error
```

إضافة لذلك في تابع الخطأ نستخدم التعليمة return 0 لإجبار المترجم على إعادة 0 بدلاً من 1 لكي لا تتوقف عملية التنفيذ.

تبقى مشكلة واحدة هي أن المترجم في هذه الحالة سيستمر في بناء شجرة الإعراب ويعطي دوماً حالة نجاح وبالتالي يطبع البرنامج لنا عبارة input accepted مهما كان عدد الأخطاء. لحل هذه المشكلة نضيف التعليمة التالية في القاعدة الأولى (جذر الإعراب):

```
S     : ST {if (errors==0) printf("Input accepted.\n");};
```

يصبح كود ملف وصف المعرب الجديد:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "lex.yy.c"
int errors=0;
%}
%token ID NUM IF THEN LE GE EQ NE OR AND ELSE EQUAL LT GT PLUS MINUS MULT
DIVS LPAR RPAR SEMI
%right EQUAL
%left AND OR
%left LT GT LE GE EQ NE
%left PLUS MINUS
%left MULT DIVS
%%
S     : ST {if (errors==0) printf("Input accepted.\n");};
ST    : IF LPAR E2 RPAR THEN ST1 SEMI ELSE ST1 SEMI
      | IF LPAR E2 RPAR THEN ST1 SEMI
      | error
      ;
```

```
ST1   : ST
      | E
      ;
E     : ID EQUAL E
      | E PLUS E
      | E MINUS E
      | MINUS E
      | E MULT E
      | E DIVS E
      | E LT E
      | E GT E
      | E LE E
      | E GE E
      | E EQ E
      | E NE E
      | E OR E
      | E AND E
      | ID
      | NUM
      | error
      ;
E2    : E LT E
      | E GT E
      | E LE E
      | E GE E
      | E EQ E
      | E NE E
      | E OR E
      | E AND E
      | ID
      | NUM
      | error
```

```
    ;
%%
int yywrap()
{
  return 1;
}
int yyerror (char *s)
{
  errors++;
  printf("Error:: %d %s at line:: %d in statement:: %s\n" ,errors ,s, line ,yytext);
  return 0;
}
int main()
{
  if((yyin=fopen("input.txt","r"))==NULL)
  {
    printf("input.txt not found !\n");
    return 1;
  }
  yyparse();
  getchar();
}
```

اكتب العبارة المصدرية التالية في الملف input.txt

```
if(a<5) )
then b b=1;
else b=*2;
```

نفذ المترجم:

```
C:\Lex_Yacc\examples\example2\exp3.exe
Error:: 1 syntax error at line:: 1 in statement:: )
Error:: 2 syntax error at line:: 2 in statement:: b
Error:: 3 syntax error at line:: 3 in statement:: *
```

لاحظ أن المترجم كشف الأخطاء الثلاثة بشكل صحيح.

والآن جرب تنفيذ حالات أخرى صحيحة وحالات خاطئة وراقب استجابة المترجم لكل منها.

س: لماذا يعطي المترجم خطأً عندما نكتب الكلمات IF THEN ELSE بأحرف كبيرة:
لأننا لم نعرف برنامج LEX على الأحرف الكبيرة. ما التعديل الذي يجب إضافته في بنية ملف LEX حتى يعامل الأحرف الكبيرة والصغيرة عند قراءة سلسلة الدخل معاملة واحدة.

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