الجلسة الثانية عشر

التحليل المعنوي

Semantic Analysis -3-

الهدف من الجلسة

- معرفة كيفية سير عملية التحليل المعنوي وكشف الأخطاء المعنوية ضمن تسلسل الدخل.
- الوصول بالطالب إلى مرحلة توليد الشيفرة الوسيطية.

مستلزمات الجلسة

- حاسب بمواصفات دنيا RAM: 1 GB, CPU: 1.6 GHz, Windows 7 OS 32 bit
- Turbo c++
- LEX & BISON tools

خطوات العمل

- ملف توصيف الماسح scanner لعملية التحليل المعنوي.
- ملف توصيف الـ Parser لعملية التحليل المعنوي.
- كشف أخطاء المعنوية الناجمة عن عمليات الإسناد الخاطئة لمتغيرات من أنواع مختلفة.

الخلاصة والنتائج:

يفترض عند نهاية الجلسة:

- تمكن الطالب من تطبيق مثال عملي متكامل عن بناء المترجمات بدءاً من مرحلة التحليل اللفظي وانتهاء بمرحلة تحليل المعاني.
- تمكن الطالب من تطبيق مثال يوضح له كيفية كشف الأخطاء المعنوية والتعامل معها.

1.1 تتمة مرحلة التحليل المعنوي:

1.1.1 كشف أخطاء المعاني الناجمة عن عمليات الإسناد الخاطئة لمتغيرات من أنواع مختلفة:

أولاً: لابد من تغيير بنية ملف جدول الرموز لأن نوع المتغيرات أصبح مهماً لنا الآن، لذلك سنقوم بتعديل التركيبة sym_node لتتضمن حقلاً جديداً هو type، نعدل ملف Symbol_table كمايلي:

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define _int 1
#define _float 2
#define _str 3
#define _chr 4
typedef struct sym_node
{
char name[56];
int type;
struct sym_node *next;
}sym_node;
```

تم تعريف أربعة ثوابت جديدة هي _int،_float،_str، و _chr تمثل النوع الصحيح والحقيقي وسلاسل الحروف والمحارف، كما تم إضافة حقل جديد للتركيبة sym_node هو type سيمثل نوع المتغير ولاحقاً ستظهر أهمية هذا الحقل.

ثانياً: سنغير الدالة (الإجرائية) التي تقوم بتثبيت الرمز في جدول الرموز كمايلي:

```
sym_node *put_sym(char *sym_name, int sym_type)
{
sym_node *ptr;
ptr=(sym_node*)malloc(sizeof(sym_node));
strcpy(ptr->name,sym_name);
ptr->type=sym_type;
ptr->next=(sym_node*)sym_table;
sym_table=ptr;
return ptr;
}
```

هنا تم فقط إضافة بارامتر وسيط للتابع put_sym هو نوع المتغير sym_type وتم إضافة تعليمة واحدة فقط لجسم التابع تقوم بنسخ نوع المتغير إلى الحقل type من المؤشر ptr الذي يتم إنشاؤه.

هذا يعني أن أي متغير جديد سيضاف إلى جدول الرموز، سيحفظ معه اسمه وكذلك نوعه type.

نحتاج إضافة للدالتين الموجودتين أصلاً في ملف Symbol_table نحتاج إلى دالة أخرى تقوم لدى استدعائها بإعادة نوع المتغير (الرمز) الذي نريد معرفة نوعه، هذه الدالة هي get_sym_type وتعرف كمايلي:

```
int get_sym_type(char *sym_name)
{
sym_node *ptr;
for(ptr=sym_table;ptr!=NULL;ptr=(sym_node*)ptr->next)
if(!strcmp(ptr->name,sym_name)) return ptr->type;
return 0;
}
```

يقوم هذا التابع (الإجرائية) بأخذ كوسيط السلسلة sym_name وهي تمثل اسم الرمز المطلوب إعادة نوعه، وفي جسم التابع يتم تعريف مؤشر ptr من النوع sym_node ويتم بعدها البحث ضمن جدول الرموز عن الرمز ذي الاسم sym_name وعند العثور عليه يتم إعادة نوع هذا الرمز ptr->type وفي حال عدم العثور عليه يتم إعادة الـ 0.

نجري هذه التغييرات على ملف Symol_table نحفظها ونغلق الملف.

ثالثاً: نعدل الآن ملف وصف المعرب:

نحتاج أولاً لمتغير لنحفظ ضمنه نوع المتغير سنسميه مثلاً current_type وهو من النوع الصحيح، وسنصرح عنه ضمن قسم التصريح في ملف المعرب كمايلي:

```
بعد التضمينات//
int errors=0;
int current_type;
```

والآن نغير في القواعد كمايلي:

```
decl_type:DEC_INT {current_type = _int;}
|DEC_REAL {current_type = _float;}
|DEC_CHAR {current_type = _chr;}
```

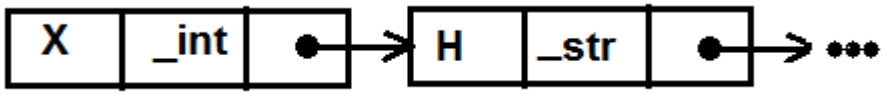
|DEC_CHAIN {current_type = _str;};

ident:ID {setup_sym($1,current_type);}

|ID ident {setup_sym($1,current_type);};

هنا يتم تغيير قيمة المتغير current_type حسب نوع التصريح، فإن كان التصريح عن متغير من النوع الصحيح أي DEC_INT عندها تصبح قيمة المتغير current_type هي int_، وإذا كان التصريح عن متغير من النوع الحقيقي أي DEC_REAL عندها تصبح قيمة المتغير current_type هي float_ وهكذا...

أما في السطرين الأخيرين فيتم إضافة الرمز إلى جدول الرموز عن طريق التابع setup_sym والذي سنمرر له اسم المتغير $1 ونوع المتغير الموجود في current_type، وبهذا فإن كل رمز ضمن جدول الرموز سيحتوي على اسم ونوع ومؤشر إلى الرمز التالي كما يوضح الشكل (9-1)



الشكل (9-1) جزء من جدول الرموز (اللائحة المترابطة)

وبذلك نضمن أن الرمز سيخزن ضمن جدول الرموز مع نوع البيانات الخاص به.

أهمية هذه الخطوة تكمن في كشف أخطاء عمليات الإسناد، والآن كل ما علينا فعله هو التحقق من أن عمليات الإسناد لن تكون خاطئة. بالعودة للقواعد لنأخذ القاعدة التالية:

```
|ID EQ ID expression SEMICOLON {
sym_check($1);
sym_check($3);
}
```

بهذا الشكل يتم فقط التحقق من وجود الرمز ضمن جدول الرموز ليتم إدخاله للجدول أو لا، لكن السؤال المهم كيف بمكن جعل هذه القاعدة تتأكد من صحة عملية الإسناد؟

نعدل القواعد لتصبح بالشكل التالي:

```
expressions:
  | expressions expression
  ;
expression:ID EQ ID SEMICOLON {
sym_check($1);
```

```
sym_check($3);
if(get_sym_type($1)!=get_sym_type($3))
{
errors++;
printf("Error %d :: %s and %s have different types :line %d.\n",errors,$1,$3,line);
}
}
|ID EQ CHAIN SEMICOLON{
sym_check($1);
if(get_sym_type($1)!=3)
{
errors++;
printf("Error %d :: %s is not declared as string type :line %d.\n",errors,$1,line);
}
}
|ID EQ CHAR SEMICOLON{
sym_check($1);
if(get_sym_type($1)!=4)
{
errors++;
printf("Error %d :: %s is not declared aschar type:line %d.\n",errors,$1,line);
}
}
|ID EQ NUM SEMICOLON{
sym_check($1);
if(get_sym_type($1)!=1)
{errors++;
printf("Error %d :: %s is not declared as integer type:line %d.\n",errors,$1,line);
}
}
|ID EQ REAL SEMICOLON{
sym_check($1);
```

```
if(get_sym_type($1)!=2)
{
errors++;
printf("Error %d :: %s is not declared as float type:line %d.\n",errors,$1,line);
}
};
```

التعديل الذي قمنا به في قاعدة expression هو:

استدعاء التابع get_sym_type لمعرفة نوع المتغيرات ومقارنتها لمعرفة إذا كانت عملية الإسناد صحيحة، وفي حال كانت عملية الإسناد خاطئة يتم طباعة رسالة خطأ.

مثلاً:

القاعدة:

```
exp:ID EQ ID SEMICOLON {
sym_check($1);
sym_check($3);
if(get_sym_type($1)!=get_sym_type($3))
{
errors++;
printf("Error %d :: %s and %s have different types :line %d.\n",errors,$1,$3,line);
}
}
```

يتم بداية استدعاء الدالة (التابع) sym_check للتأكد من وجود الرمز مسبقاً ضمن جدول الرموز، فإن كان موجوداً فهذا يعني أن عملية الإسناد تتم على متغيرات معرفة ولا يوجد خطأ (هذا من الفقرات السابقة).

أما الجديد، فهو بعد عملية التحقق يتم استدعاء التابع get_sym_type لكلا طرفي عملية الإسناد.

تقوم الأسطر الجديدة المضافة بالتحقق من تطابق نوع المتغيرين $1 و $3 حيث إنه لا يجوز أن يكون نوع المتغيرين غير متطابقين وإلا فإن عملية الإسناد خاطئة. يتم التأكد من ذلك باستدعاء التابع get_sym_type مرة باستخدام المتغير الأول get_sym_type($1) ومرة باستخدام المتغير الثاني get_sym_type($3)، فإن تطابقت نتيجتا الاستدعاءين هذا يعني أن المتغيرين يمتلكان النوع نفسه وعملية الإسناد صحيحة وإلا سيتم طباعة رسالة خطأ تفيد بأن عملية الإسناد خاطئة لأن المتغيرين من نوعين مختلفين.

لنأخذ قاعدة أخرى:

```
|ID EQ NUM SEMICOLON{
sym_check($1);
if(get_sym_type($1)!=1)
```

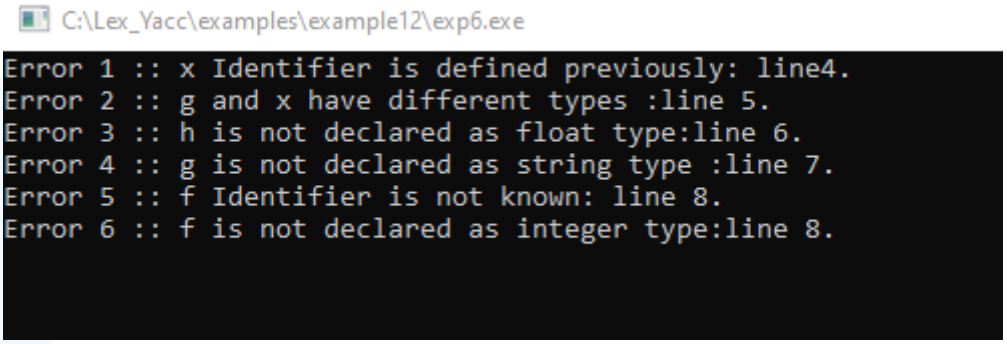
```
{
errors++;
printf("Error %d :: %s is not declared as integer type:line %d.\n",errors,$1,line);
}
}
```

هنا تم إضافة أسطر تقوم باستدعاء التابع get_sym_type للحصول على نوع المتغير $1 ليتم التأكد من أنه من النوع الصحيح int_ (ذو الرقم 1)، فإذا لم يكن المتغير من النوع الصحيح، فهذا يعني أن عملية الإسناد خاطئة وسيتم طباعة رسالة خطأ.

نغلق ملف المعرب بعد حفظ التغييرات السابقة ونعيد مراحل بناء المترجم من جديد ونختبره على العبارة التالية:

```
real x;
real x;
int g;
chain h;
chain x;
g=x;
h=0.05;
g="ddd";
f=5;
```

والنتيجة:



التقط المترجم كل الأخطاء المعنوية التي حدثت في تسلسل الدخل السابق رغم أن ليس أي منها هو خطأ قواعدي أو معجمي وستمر هذه العبارات على مرحلتي التحليل المعجمي والقواعدي دون أي مشكلة.

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