الجلسة العاشرة

التحليل المعنوي

Semantic Analysis

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 كيفية كشف الأخطاء المعنوية:

قد تكون العبارة التي ندخلها للمترجم صحيحة لفظياً وقواعدياً لكنها ليست ذات معنى، وهنا يأتي دور المحلل المعنوي، والذي يعتمد على جدول الرموز Symbol Table لكشف مثل هكذا أخطاء.

بفرض استخدامنا نفس المترجم الذي قمنا ببنائه المحاضرة السابقة ولنقم الآن باختباره على العبارة التالية:

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

تحتوي العبارة السابقة العديد من الأخطاء المعنوية:

1- مثلا في السطر الرابع هناك تكرار لتصريح عن المتغير h الذي صرح عنه في السطر الثالث على أنه من النوع الحرفي char.

2- هناك أيضاً خطأ معنوي في السطر السابع وهو إسناد سلسلة لمتغير من النوع الصحيح.

3- كذلك هناك خطأ معنوي في السطر الثامن وهو استعمال متغير لم يتم التصريح عنه سابقاً وهو المتغير y.

لكن المترجم السابق لن يتعرف على هذه الأخطاء كونها ليست قواعدية وإنما معنوية ولجعله قادراً على ذلك نحن بحاجة لبناء جدول الرموز الذي يحتوي الرموز التي تم العثور عليها في تسلسل الدخل وقيم ونوع هذه الرموز.

سنتعلم أولاً كيفية التخلص من أخطاء التصريحات المتكررة والأخطاء الناجمة عن استخدام متغيرات لم يتم التصريح عنها.

1.2 إنشاء جدول الرموز Symbol Table:

من أجل إنجاز مرحلة التحليل المعنوي، علينا كتابة جدول الرموز والذي نخزن فيه المتغيرات بكافة أنواعها، حيث لن تهمنا قيمها وإنما يهمنا نوع هذه المتغيرات واسمها. ننشئ ملفاً جديداً باسم SYMB_TAB.H يمثل جدول الرموز داخل المجلد الذي نعمل به، ونفتحه ونبدأ بكتابة الكود اللازم:

يتضمن هذا الكود ثلاثة أجزاء: الأول إنشاء لائحة مترابطة تمثل جدول الرموز، الثاني إجرائية (تابع) إضافة رمز جديد إلى اللائحة، والثالث إجرائية التحقق من وجود رمز في اللائحة.

إذاً يتألف البرنامج من المراحل التالية:

1- تعريف تركيبة struct مكونة من سلسلة محرفية بحجم أعظمي 50 محرف اسمها name تمثل هذه السلسلة اسم المتغير ، كما تتضمن التركيبة مؤشر لتركيبة من نفس النوع next، ثم تم إنشاء مؤشر sym_table من نوع التركيبة أيضاً وجعلناه يشير إلى الـ NULL، مبدئياً وحقيقة هو القائمة التي تمثل جدول المتغيرات كما في الكود التالي:

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
typedef struct sym_node
{
char name[50];
struct sym_node *next;
}sym_node;
sym_node *sym_table=NULL;
```

2- التصريح عن تابع اسمه put_sym يعيد قيمة من النوع sym_node ويأخذ كوسيط البارامتر sym_name ويقوم هذا التابع بتسجيل الرمز sym_name ضمن جدول الرموز ليصبح معلوماً.ويعيد مؤشر إلى جدول الرموز وهو ptr.

أي أن وظيفة هذا التابع تسجيل الرمز ضمن جدول الرموز كما في الكود التالي:

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

3- التصريح عن تابع اسمه get_sym يعيد قيمة من النوع sym_node ويأخذ كوسيط البارامتر sym_name ويقوم هذا التابع بالحصول على الرمز ذي الاسم sym_name من جدول الرموز، حيث يقوم بالبحث ضمن جدول الرموز الممثل بالمؤشر sym_table حتى الوصول إلى نهايته (أي إلى NULL) وفي حال وجوده يعيد مؤشر لمكان وجوده ptr وإلا فإنه يعيد NULL ليدل على أن الرمز المطلوب غير موجود في جدول الرموز.

وهنا يتم التأكد من أن الرمز موجود فعلاً أو غير موجود ضمن جدول الرموز.

كما في الكود التالي:

```
sym_node *get_sym(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;
return NULL;
}
```

فيمايلي شرح التعليمات بالتفصيل:

1.2.1 أولاً تعليمات تابع put_sym:

```
ptr=(sym_node*)malloc(sizeof(sym_node));
```

تقوم هذه العبارة بتخصيص بلوك من البايتات بحجم sym_node التي تم إنشاؤها سابقاً. كما تعيد هذه التعليمة مؤشراً إلى بداية هذا البلوك. وكنتيجة سنحصل على مؤشر ptr يشير إلى بداية بلوك من البايتات حجم هذا البلوك هو بحجم sym_node. أما بالنسبة للعملية (sym_node*) فهي تمثل عملية قصر نوع المؤشر الناتج إلى نوع التركيبة sym_node* لضمان أن المؤشر ptr يطابق مؤشر جدول الرموز sym_table.

```
strcpy(ptr->name,sym_name);
```

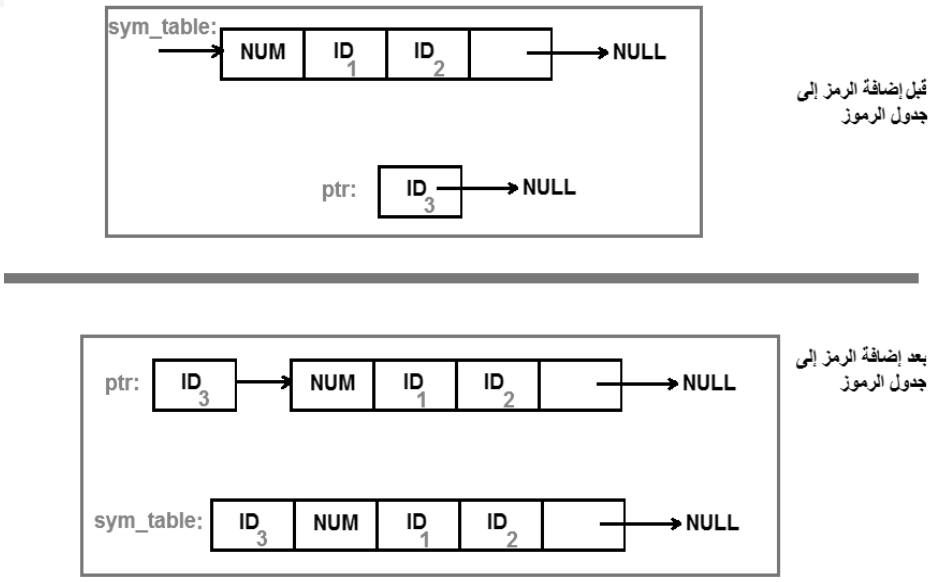
تقوم بنسخ اسم الوسيط sym_name إلى اسم المؤشر الجديد ptr.

```
ptr->next=(sym_node*)sym_table;

sym_table=ptr;

return ptr;
```

في هذه التعليمات يتم جعل next الخاصة بالمؤشر ptr تشير إلى جدول الرموز نفسه ثم يتم جعل المؤشر ptr يشير إلى جدول الرموز، وأخيراً يعاد المؤشر ptr ليمثل القيمة المعادة من قبل التابع put_sym.

للتوضيح أكثر لاحظ الشكل التالي:

الشكل (8-1) يوضح عملية إضافة رمز جديد إلى جدول الرموز.



الشكل (8-1) كيفية إضافة رمز إلى جدول الرموز

1.2.2 ثانياً تعليمات تابع get_sym:

يعيد التابع strcmp قيمة الصفر في حال كانت السلسلتان التي يقارنهما متطابقتان وَ التعليمة:

```
if(!strcmp(ptr->name,sym_name))return ptr
```

تعني أنه في حال كون ناتج strcmp هو الصفر أي السلسلتان متطابقتان فإن strcmp! هو 1 وبالتالي هنا ستتم إعادة المؤشر لمكان وجود الرمز المطلوب بسبب حصول تطابق بين اسم المؤشر (الرمز) المطلوب وبين اسم أحد الرموز في جدول الرموز، وإلا سيعيد NULL.

بعد أن قمنا ببناء جدول الرموز نكتب ملف وصف الـ Scanner والـ Parser وهنا ستعدل الملفين الموجودين في الجلسة التاسعة بما يناسب التحليل المعنوي (محتوى الجلسة العاشرة القادمة).

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