编译原理实验 2

课程：编译原理

211220169 祝明浩 211220182 刘钊瑜

专业：计算机科学与技术

一．功能说明

完成了实验 2 必做部分和选做 2.2 内容，实现了嵌套作用域的相关内容

新增代码结构：

Symbol_table.h Symbol_table.c 符号表定义和相关操作实现

Semantic.h Semantic.c 自顶向下分析语法树，找到相关的语义错误并报相应类型的错误

二．编译方法

使用文件提供的 Makefile 文件，执行 make 进行编译

三．代码亮点

由于本实验需要完成的是嵌套作用域部分的选作内容，因此对符号表更加需要精心设计，本实验采用了讲义提供的哈希表 + 十字链表的方式作为符号表数据结构的基础，具体定义如下：

```
typedef struct HashNode_* HashNode;
HashNode hash_table[HASH_SIZE];
HashNode stack_table[100];
extern int DEPTH;
```

```
struct HashNode_
{
    FieldList data;//域的信息
    int depth;//作用域深度
    HashNode next_in_bucket;// 下一个桶内节点
    HashNode next_in_same_scope;//下一个作用域内节点
};
```

在哈希节点的定义中，为了实现十字链表的功能，需要定义其横向和纵向的下一个节点，另外用 depth 表示该变量的作用域深度，并维护一个全局变量 DEPTH 作为当前的遍历深度（只针对{}内的内容）。

在哈希节点的构造中，需要区分全局变量和局部变量，局部变量只在当前深度及以下发挥作用，结束后消亡，而全局变量作用于整个程序，程序执行结束前永远不会消亡。

```
void insert_hashnode(HashNode node_);
void insert0_hashnode(HashNode node_);//将该哈希节点插入到最外层（结构体都是全局的）所以不会被删除，干脆直接插入到哈希
void push_stacktable();
void pop_stacktable();
```

而对于深度的维护将由栈实现，每次深度增加一层，这个栈 push 进去一个空结点，深度加一，所有在这一层内的变量与这个空结点首尾相连，而 pop 的过程需要依次删除空结点和连接的所有结点，表示消亡。

那么在什么情况下应该执行 push 和 pop 操作呢？首先在程序执行开始进行一次 push，表示深度为 0，在此深度下定义的变量均为全局变量。

```
void Program_init(Node* root)
{
    init_hashtable();
    init_stacktable();
    push_stacktable();
    if(root==NULL) return;
    ExtDefList(root->child);
}
```

然后，在定义函数形参（如 int main(int i)）的时候，进入 int i 之前进行一次 push，表示该函数的全局作用域，另外在遇到函数内出现{}（语句块开始）的时候需要 push,表示进入了函数内部的某深度的局部作用域，对于 pop 操作，面对上述两种情况都只需要在语句块结束之前执行 pop 即可。

```
if(strcmp(t1->silbing->silbing->name,"RP")==0){
    type1->u.function.num_of_parameter=0;
    type1->u.function.parameters=NULL;
    push_stacktable();
}
else if(strcmp(t1->silbing->silbing->name,"VarList")==0){
    type1->u.function.num_of_parameter=0;
    type1->u.function.parameters=NULL;
    Node*t2=t1->silbing->silbing;
    Node*t3=t2->child;//VarList → ParamDec COMMA VarList | ParamDec
    push_stacktable();
    while(t3!=NULL){
```

```
if(strcmp(t1->name,"CompSt")==0)
{
    push_stacktable();
    //printf("haha\n");
    CompSt(t1,ret_type);
}
else if(strcmp(t1->name,"RETURN")==0)
```

```
void CompSt(Node* root,Type ret_type){//由于函数形参也是函数的局部作用域，因此Co
    Node* t1=root->child->silbing;
    Node* t2=NULL;
    Node* t3=NULL;
    if(t1->silbing!=NULL)
    {
        t2=t1->silbing;
        if(strcmp(t1->name,"DefList")==0) DefList(t1);
        else if(strcmp(t1->name,"StmtList")==0) StmtList(t1,ret_type);
        if (t2->silbing!=NULL)
        {
            if(strcmp(t2->name,"StmtList")==0) StmtList(t2,ret_type);
        }
    }
    pop_stacktable();
}
```

以上就是对符号表的结构定义和对嵌套作用域的实现。