Lab2-语义分析

程序功能说明

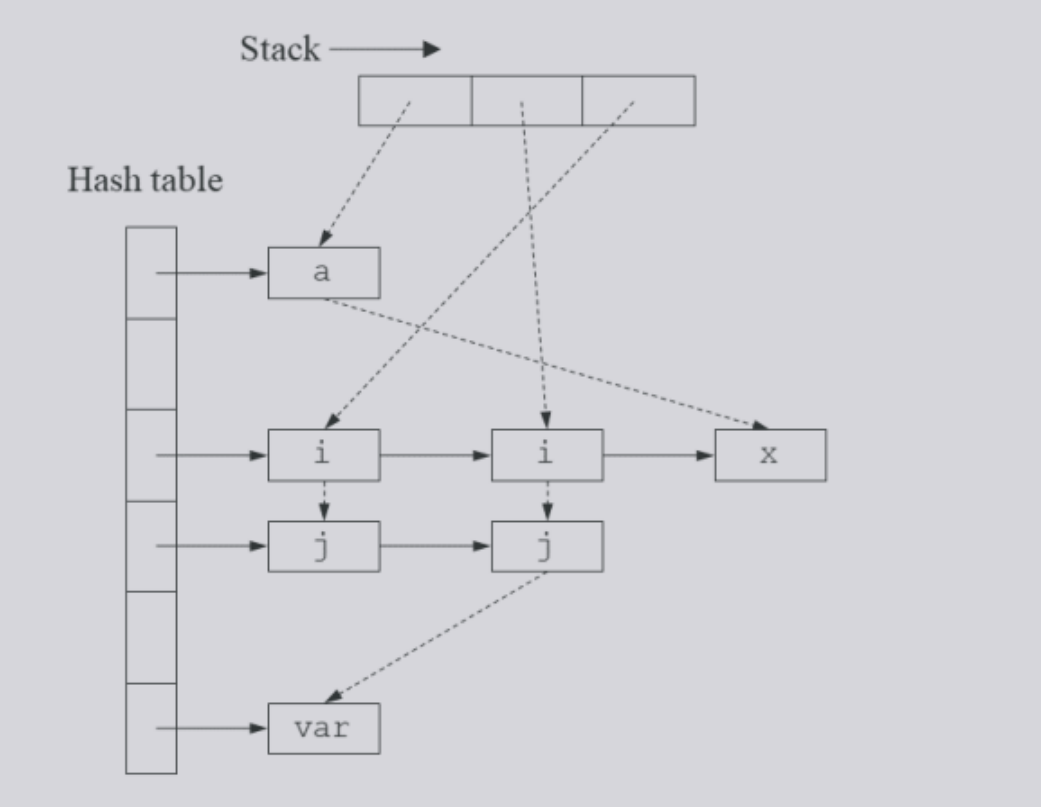
本实验基于C—文法实现了编译过程的语义分析功能，包含符号表的建立和语义检查。

实现方式

程序对语法树进行先序遍历，依次处理每一个语法单元，动态地维护一个符号表，并在使用符号时查表做语义检查。

符号表选用哈希表（散列表）的实现方式，散列表是一种可以达到搜索效率极致的数据结构。一个好的散列表实现可以让插入、查找和删除的平均时间复杂度都达到O(1)。使用哈希函数将任意长字符串映射到固定区间内，然后将符号表项插入到符号表数组的对应位置的单链表表头。

在哈希表的基础上同时支持了变量的多层作用域：维护一个深度数组（模拟栈），数组的每一项维护一个单链表，表项是已经插入到哈希表的元素，在结构上是一个十字链表的形式。



编译使用说明

源文件：`main.c, lexical.l, syntax.y, tree.h, semantic.h, semantic.c`

编译时直接执行`make`即可，`Makefile`指定的默认编译目标会首先执行`clean`目标，不需要手动`make clean`。

编译产生可执行程序`parser`

```
./parser [target-cmm-file]
```

细节说明

符号表表项与类型定义

```
// 类型，分为基本类(int, float)，数组，结构体，后两者允许嵌套定义
struct _type {
    enum { BASIC, ARRAY, STRUCTURE, NULL_TYPE } kind; // NULL for
error
    union {
        int basic; // 0 for int, 1 for float
        ArrayType* array;
        StructType* structure;
    };
};

struct _arraytype {
    Type* elem;
    int size;
};

struct _structtype {
    int with_tag;
    char* name;
    FieldList* structure;
};

struct _fieldList {  // 定义一个变量列表结构，利于函数参数和结构体的封装
    char* name;
    Type* type;
    FieldList* next;
};

// 符号表表项
struct _list_item {
```

```
    ListItem* next; // 在哈希表链表上的横向指针
    ListItem* depnext; // 在深度链表上的纵向指针
    enum { VARIABLE, FUNCTION, STRUCTTAG } kind; // 一个有tag的结构
体，同时作为一个类型和一个符号被存储。
    char* name;
    Type* type;
    int depth;
    // for functions
    int param_count;
    FieldList* params;
};
```

多重作用域的处理

涉及到符号语义检查的处理函数都需传入一个`depth`参数，每当进入一个作用域（CompSt），depth+1，退出时恢复；在创建新符号时，将符号表项插入哈希表的同时，也将其插入当前深度对应链表；在退出时，删除当前depth下的所有符号，以释放内存并确保后续作用域不会访问到已经退出的作用域内的符号。