编译原理实验报告

实验二：语义分析

学院：	计算学部	指导老师：	单丽莉
学号：	2021112845	姓名：	张智雄

一、实验目的

1. 巩固对语义分析的基本功能和原理的认识。
2. 能够基于语法指导翻译的知识进行语义分析。
3. 理解并处理语义分析中的异常和错误。

二、实验内容

在词法分析和语法分析程序的基础上编写一个程序， 对 C--源代码进行语义分析和类型检查， 并打印分析结果。要求能够如下类型的错误：

a) 变量（包括数组、指针、结构体） 或过程未经声明就使用；
b) 变量（包括数组、指针、结构体） 或过程名重复声明；
c) 运算分量类型不匹配；
d) 操作符与操作数之间的类型不匹配。

2.1 实验环境

Ubuntu 22.04；GCC version 11.4.0；GNU Flex version 2.6.4；GUN Bison version 3.8.2

2.2 实验过程

2.2.1 符号表的定义

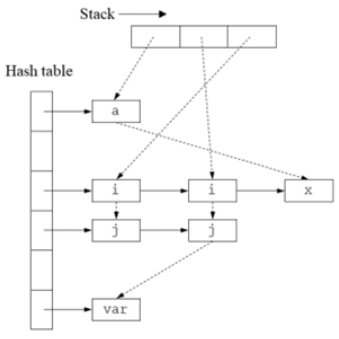
本实验使用基于十字链表和 open hashing 散列表的 Imperative Style 的符号表设计，除了散列表本身为了解决冲突问题所引入的链表之外，它从另一维度也引入链表将符号表中属于同一层作用域的所有变量都串起来。具体如下表所示：

```
typedef struct tableItem* pItem;
typedef struct tableItem {
    int symbolDepth;
    pFieldList field;
    pItem nextSymbol;
    pItem nextHash;
} TableItem;
```

```
typedef struct table* pTable;
typedef struct table {
    pHash hash;
    pStack stack;
    int unNamedStructNum;
} Table;
```

```
typedef struct stack* pStack;
typedef struct stack {
    pItem* stackArray;
    int curStackDepth;
} Stack;
```

```
typedef struct hashTable* pHash;
typedef struct hashTable {
    pItem* hashArray;
} HashTable;
```



对于此数据结构的具体解释如下：

- **TableItem（符号表项）：**这个结构定义了符号表中的每个条目。每个条目包含了**符号的深度(symbolDepth)，字段列表的指针(field)，以及两个指针**，分别指向同一深度下的下一个符号(nextSymbol)，以及同一散列码下的下一个符号(nextHash)。
- **HashTable（散列表）：**这个结构定义了符号表中使用的散列表，采用了**开放定址法。**散列表的每个槽位指向一个符号表项指针数组，可以用来**解决散列冲突**。

- **Stack（栈）：**栈结构用于维护当前符号表的深度。栈中的每个元素都是一个符号表项指针数组，从而能够**按照符号的作用域**进行管理，方便了符号的查找和作用域的处理。
- **Table（符号表）：**完整的符号表将散列表和栈结合起来，**包含了一个指向散列表的指针和一个指向栈的指针**，同时还记录了未定义结构体数目(unNamedStructNum)。

2.2.2 类型表示

对于基本类型，只需用不同的常数代表不同的类型；而对于数组和结构体，可以使用链表来表示，具体如下：

- **Type（类型）：**包括基本类型(Basic)、数组类型(Array)、结构体类型(Structure)和函数类型(Function)。通过枚举类型 Kind 来表示具体的类型种类；同时通过共用体 union 设置每种类型对应的数据结构来描述其具体信息。
  - Basic 类型：使用不同的常数代表不同的类型，如整数、浮点数等。
  - Array 类型：包含了元素类型(elem, for multiple levels)和数组大小(size)。
  - Structure 类型：包含了结构体的名称(structName)和字段列表(field)，通过链表的方式来描述结构体中的各个字段。
  - Function 类型：包含形参个数(argc)、形参列表(argv)以及返回类型(returnType)。
- **FieldList（字段列表）：**这个结构用来描述结构体或函数的字段信息。每个字段包含了**字段名(name)、字段类型(type)以及指向下一个字段的指针(tail)**。通过链表的形式构建了**结构体**的字段列表或**函数**的形参列表。

2.2.3 语义分析/检查

本实验在实验一所构建的语法树上进行遍历以进行符号表的相关操作以及类型的构造与检查，能够实现错误类型 1-17 的全部检查以及选做要求 2.2。

函数从根节点开始进行先序遍历，每当遇到 ExtDef（包含全局变量的定义、结构体的定义、数组、基本类型）时递归进行语义检查，即将信息通过对子结点们的遍历提炼出来进行处理并插入到符号表里（局部定义 Def 会被包含在递归过程内）。

处理过程中需要根据作用域维护局部变量定义和语句列表。而当处理语句和表达式时，需要进行相应的进行类型检查和更新符号表中的信息。

具体当遇到参数、函数和结构体定义时会对符号表进行查询，首先查询声明是否存在，对应错误类型 1, 2, 17；而后检查变量是否存在重复定义，分别对应错误类型 3, 4, 15, 16。

- 通过 searchTableItem 函数计算项目名称的哈希码，找到哈希表中对应的链表头，并在该链表中遍历查找项目。对于变量和函数，只需根据此结果即可判定是否已声明；而对于结构体，还需要通过其类型判定其是否为结构体定义。
- 而 checkTableItemConflict 函数则调用 searchTableItem 函数来查找是否存在与待插入项目名称相同的项目。如果找到了同名项目，继续**检查符号深度与当前栈深度相同**的项目（同一作用域），以及检查变量与前面定义过的结构体名字是否重复。
- 在局部声明 Dec 中，对于结构体内部，通过 VarDec 函数获取变量的字段列表，然后与已存在的结构体字段列表逐一比较，检查**域内定义是否重复**。

注：哈希码的计算是通过遍历字符串中的每个字符，将其 ASCII 值与当前哈希值相结合，并对结果进行异或修正，最终得到字符串的哈希码。

每当遇到语法单元 Exp，说明该结点及其子结点们会对变量或者函数进行使用，这个时候应当查符号表以确认这些变量或者函数是否存在以及它们的类型是什么。

1) 对于基本数学运算符，检查操作数类型是否匹配，处理赋值操作符的左值检查。对应错误类型 5, 6, 7；
2) 处理数组和结构体访问，检查索引和域名的合法性，对应错误类型 10, 12, 13, 14；
3) 处理单目运算符，检查操作数类型是否合法，对应错误类型 7；
4) 处理括号表达式，递归处理括号内的表达式。
5) 处理函数调用，检查函数是否定义，参数个数和类型是否匹配，并返回函数返回值的类型，对应错误类型 9, 11；
6) 处理变量和常量，检查变量是否定义，并返回其类型。

而在分析 Stmt 时遇到 Stmt → RETURN Exp SEMI 时，需检查返回类型是否一致，对应错误类型 8。

具体而言，类型检查时首先排除 NULL 类型，然后检查是否存在函数类型，根据类型种类继续比较，包括基本类型、数组元素类型和结构体名称（名等价），以确定它们是否相容。

最后，由于采用基于十字链表和 open hashing 散列表的 Imperative Style 的符号表设计，因而能够根据栈的深度正确识别变量的作用域。 具体而言，当编译器发现函数中出现了一个被“{”和“}”包含的语句块（CompSt/Varlist 语法单元）时，它会将当前深度符号表压栈；而当此代码块语法检查结束时又会将这个部分弹栈。

于是外层语句块中定义的变量可在内层语句块中重复定义，内层语句块中定义的变量到了外层语句块中就会消亡，不同函数体内定义的局部变量可以相互重名。

2.4 实验结果

在 linux 环境下，在文件夹目录下中运行 make 指令编译所有的文件，而后输入 make test 指令便可以测试所有的样例。测试结果如下，能够正确分析所有必做用例和部分选做用例：

图 1 测试结果

三、实验收获

1. 通过编写代码实现对 C--源代码的语义分析和类型检查，包括识别未声明的变量或过程的使用、重复声明、运算分量类型不匹配等错误类型，深入了解了语义分析的基本功能和原理，提高了对语义分析过程的理解。

2. 实验中采用了基于十字链表和开放定址法的符号表设计，同时根据作用域正确识别变量的作用域。这些操作加深了对符号表设计和作用域处理的理解和掌握。