编译器设计专题实验
实验报告

实验 6 符号表

班级：计算机 2102
姓名：李芝塬
学号：2216113163

目录

实验要求 1
实验分析 1
实验结果 2
实验总结 5
附录 6

实验要求

设计和实现一个符号表。
使用散列或线性结构实现符号表数据结构
设计符号表可存储程序语言中的各种标识符（变量、常量、数组、结构、指针、函数和过程）及其属性和作用域信息。
主要操作:填表和查表
•1)填表:当分析到程序中的说明或定义语句时，将说明或定义的名字，以及与之有关的信息填入符号表中。
•2)查表:填表前查表，检查在程序的同一作用域内名字是否重复定义；检查名字的种类是否与说明一致；对于强类型语言，要检查表达式中各变量的类型是否一致；生成目标指令时，要取得所需要的地址

实验分析

每个表项可以保存一个符号，需要定义结构体用于存储一个表项。用一个指针数组存储所有的表项，使用 hash 函数根据变量名查找表项的下标。为了解决 hash 冲突，使用链表存储冲突的表项，即：如果两个表项变量名相同或者变量名不同但 hash 值相同，则存储在同一个链上。
本符号表可以表示 7 个类型：0-void 1-int 2-char 3-function 4-intArray/int* 5-charArray/char* 6-constString
每次插入符号之前，需要先查表，判断这个符号是否已经存在。
记录变量的作用域是通过记录符号所在函数实现的。变量的作用域是这个变量所在函数。全局变量的作用域是一个专门的函数 globalVariable。插入符号时，如果不存在对应的作用域，则会自动将作用域设置为全局，并使用 Warning 提示。constString 的作用域是全局，不需要也不能指定。

实验结果

考虑到本次实验的词法分析使用 flex 完成，语法分析使用 bison 完成，如果将符号表与词法分析、语法分析、语义分析直接结合起来，需要自己设计一套完整的正则表达式、属性文法，并在 flex 和 bison 完成结合，实现难度太大，所以只提供了符号表的接口，可以插入、查询符号，并自动完成查重工作。

使用结构体 SymbolEntry 记录一个表项：

```
struct SymbolEntry
{
    char *name;
    int type;
    int val; // 对于int 与char 是值，  对于int arrary, char arrary, constStr 是长度
    function *fun;
    char *constString;
    function *level; // globalVariable 为全局变量，function 为所在的函数
    struct SymbolEntry *next;
};
```

每个表项记录了符号的名称、类型、值、作用域。fun 仅作用于函数类型的符号，constString 仅作用于字符串常量。next 用于组成链表。

使用结构体 function 记录一个函数：

```
typedef struct
{
    char *name;
    int returnType;
    int arg0_type;
    char *arg0_name;
    int arg1_type;
    char *arg1_name;
    int arg2_type;
    char *arg2_name;
    int arg3_type;
    char *arg3_name;
} function;
```

每个 function 记录函数名、返回类型，以及 4 个形参的类型与名称。

对于全局变量，定义一个特殊的函数 `function globalVariable` 用于表示全局变量。

使用 hash 函数，根据变量名获得一个在 0-TABLE_SIZE 之间的数字：

```
int hashFunction(char *str)
{
    unsigned int hash = 0;
    for (int i = 0; str[i] != '\0'; i++)
    {
        hash = (hash << 5) - hash + str[i];
```

```
        hash &= 0x7FFFFFFF; // 保持 hash 值非负
    }
    return hash % TABLE_SIZE;
}
```

查表时，对于 constString，使用字符串内容查找；对于其他类型的符号，使用符号名查找。返回值是一个链表，需要调用者自己根据符号名称和类型遍历链表。

代码整体框架如图所示：

```
/*
 * @Author: 李芝塬 Li Zhiyuan
 * @Email zhiyuanli0122@outlook.com
 */
#include <stdio.h>···
#define TABLE_SIZE 100
/*···
typedef struct···
} function;
char *type[] = {"void", "int", "char", "function", "int *", "char *", "constString"};
struct SymbolEntry···
function globalVariable;
struct SymbolEntry *symbolTable[TABLE_SIZE];
int constStringCount = 0;
void initSymbolTable()···
int hashFunction(char *str)···
void insertInt(char *name, char value, function *level)···
void insertChar(char *name, char value, function *level)···
void insertFunction(char *name, function *level, int returnType, int arg0_type, char *arg0_name, int arg1_type, char
void insertIntArray(char *name, int length, function *level)···
void insertCharArray(char *name, int length, function *level)···
void insertConstString(char *str)···
struct SymbolEntry *findSymbol(char *name)···
struct SymbolEntry *findSymbolByConstString(char *str)···
void printSymbolTable()···
void readData()···
int main()···
```

运行程序：./run.sh

会自动从 input.txt 中读取输入，并输出项目表中的所有内容。如图所示：

```
lzy@iZj6c34iaw7zzdcsox1suvZ:~/compile/6$ ./run.sh
编译实验6
现在的时间:
Tue Jul  2 04:07:15 PM CST 2024
开始编译:
运行程序:
输入的内容:
// 这是输入文档，  使用//写注释
// int/char 变量名 值 作用域
// function 函数名 作用域 返回类型 形参0类型 形参0名称 形参1类型 形参1名称 形参2类型 形参2名称 形参3类型 形参3名称
// intArray/charArray 数组名 数组长度 作用域
// constString 字符串内容
// 类型: 0-void 1-int 2-char 3-function 4-intArray/int* 5-charArray/char* 6-constString

int i0 10086 0
char i0 $ 0
char c # 0
// 这是一行注释
function main 0 0 1 argc 5 argv 0 a2 0 a3
function printf main 1 5 str0 5 str1 5 str2 1 num
intArray list 30 main
charArray str 500 notExistFunc
constString zhiyuanli0122@outlook.com
constString zhiyuanli0122@outlook.com
constString 李芝塬
输出的内容:
Warning: function notExistFunc not found!! Used globalVariable instead.
Info: constString zhiyuanli0122@outlook.com already exists
name:   main
        type: function
        level: globalVariable
        returnType: void,
        arg0_type: int, arg0_name: argc,
        arg1_type: char *, arg1_name: argv,
        arg2_type: void, arg2_name: a2,
        arg3_type: void, arg3_name: a3
```

```
name:   i0
        type: int
        val: 102
        level: globalVariable
name:   i0
        type: char
        val: $
        level: globalVariable
name:   constString0
        type: const string
        value: zhiyuanli0122@outlook.com
        level:globalVariable
name:   constString1
        type: const string
        value: 李芝塬
        level:globalVariable
name:   list
        type: int array
        length: 30
        level:main
name:   printf
        type: function
        level: main
        returnType: int,
        arg0_type: char *, arg0_name: str0,
        arg1_type: char *, arg1_name: str1,
        arg2_type: char *, arg2_name: str2,
        arg3_type: int, arg3_name: num
name:   c
        type: char
        val: #
        level: globalVariable
```

如图所示。这个程序可以在输入文件中写注释，注释不会影响运行结果。

在插入 charArray 时，所属的作用域（函数名）为 notExistFunc。由于这个函数不存在，所以会自动将这个符号的作用域变为 globalVariable，即将这个符号变为全局变量，并输出 Warning 提醒用户。

在插入重复的字符串常量时，会先查表，发现有相同的内容，所以直接忽略掉第二次插入。

在插入字符串常量时，会自动为字符串生成内部的变量名 conString*，其中*为数字。

在插入符号时，会首先查找是否存在相同的符号：

- 使用符号名称、类型、作用范围三者进行比较
- 如果两个符号的名称、类型、作用范围之中有一个不同，则允许同时存在
- 如果两个符号的三个属性都相同，则仅保留最先插入的符号
- 例如，int a=10 和 char a=$允许同时存在，但是 int a=1 和 int a=2 不允许同时存在，只保留 int a=1
- 重复的字符串只保存一份，并使用 Info 提示

在运行结果中，可以看到有 2 个符号 i0，一个是 int，一个是 char，则允许同时存在。

```
name:   i0
        type: int
        val: 102
        level: globalVariable
name:   i0
        type: char
        val: $
        level: globalVariable
```

实验总结

本次实验实现了符号表，可以插入符号、查找符号，并自动检查重复符号、检查作用域合法性，并且使用了链表，没有存储符号的数量限制，兼具查表速度与容量，已经实现了对7种类型的支持。

不足之处在于：没有与词法分析、语法分析、语义分析结合起来，仅实现了调用的接口；类型设计没有实现对指针的支持，只能实现int数组和char数组，不能实现int*, int**, int***。

实验本身并不困难，但是在实现的过程中有大量的空指针、内存分配，导致调试过程中经常出现Segmentation Fault。又由于使用远程服务器进行开发，没有配置Debug,，不能直接打断点查看变量，所以写起来很麻烦。

目前为止，已经对编译原理有了较好的理解，但是目前的若干次实验还不能将各模块串联在一起，对编译原理的理解还不够深入。

附录

源代码：Github

使用的 AI：

- Kimi
- Kimi
- Kimi
- Code Whisper