编译原理实验报告

刘驭壬
匡亚明学院
141242026
141242026@smail.nju.edu.cn

实验进度：

我完成了必做内容：
1.能够检测C—语言的17种类型错误。
2.未完成选做内容
git完整记录了实验的全过程。同步github:
https://github.com/AlexLiuyuren/cmm-compile/

运行：

进入src文件夹
make: 编译得到parser可执行文件（位于根目录）
make test2: 测试根目录test文件夹上所有实验二的17个测试用例，结果位于test_result文件夹下。
make clean: 删除中间文件和parser可执行文件
最后make的时候会报warning，不影响运行。
由于我注释了打印语法树的代码，所以此时如果make test1(运行实验1的测试用例)，结果将不会输出语法树。

代码介绍：

我的整体思路是采用先建立语法树再遍历节点执行语义分析 。
实验二中，我添加的源文件有symbol_table.c 和 symbol_table_toolfunction.c。
symbol_table.c：
该文件中主要包含了具体遍历语法树，根据语法树构造符号表的函数。
入口函数如下：

```
//main function
void symbolTableMain(TreeNode *p){
        kSymbolStackHead = (SymbolStackNode *)malloc(sizeof(SymbolStackNode));
        kSymbolStackHead->symbol_head = NULL;
        kSymbolStackHead->func_ptr = NULL;
        kSymbolStackHead->next = NULL;
        buildSymbolTable(p);
        return;
}
```

symbol_table_toolfunction.c:
该文件中包含了构造符号表所需的基本函数，给symbol_table.c中的函数调用。

头文件有symbol_table.h，头文件中包含了两个源文件中的类型定义与函数声明。
详情见代码和注释。

符号表和变量类型的设计：

符号表的设计我是采用了散列表+链表的方式（openhashing)
我的变量类型的设计主要参照讲义内容，如下所示：

```
typedef struct Type{
        enum{BASIC, ARRAY, STRUCT, OTHER, NOTDEF} type;
        union{
                enum{B_INT, B_FLOAT} basic;
                struct { struct Type *element; int size;} array;
                struct StructContent *structure;
        };
} Type;

typedef struct StructContent{
        char name[kMaxLen];
        Type *type;
        struct StructContent *next;
} StructContent;
```

```
typedef struct SymbolNode{
        char name[kMaxLen];
        int lineno;
        // either isfunc is true or isdef is true
        int is_func;
        int is_def;
        union{
                struct{
                        Type return_value;
                        int argument_num;
                        Type *argument_type;
                }func_info;
                Type *def_info; //symbol is basic/array/struct
        };
        struct SymbolNode *stack_next, *hash_next;
} SymbolNode;

typedef struct SymbolStackNode{
        struct SymbolNode *symbol_head;
        struct SymbolNode *func_ptr;
        struct SymbolStackNode *next;
} SymbolStackNode;
```