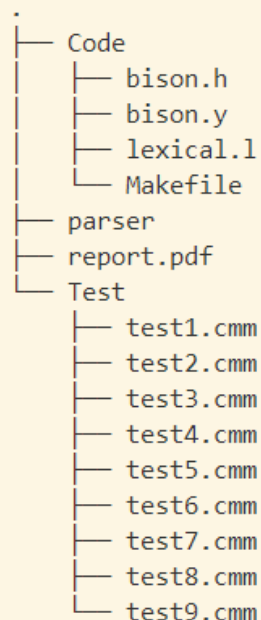


《编译原理》实验一实验报告

厉肖 161220076 计算机科学与技术系

1. 文件目录结构

右图为项目文件结构，子文件夹 Code 下包含实验代码和 Makefile 文件。parser 为实验可执行文件。Test 文件夹包含多个测试样例文件。



2. 运行

在 code 文件夹下执行 make 命令编译代码，编译得到的可执行文件为上一级目录的 parse 可执行文件。parse 接受一个参数（测试样例文件名），在上一级目录执行 ./parse Test/test.cmm 即可运行分析器，输出结果会在终端中打印出来。

3. 完成的功能点

- 正确识别语法错误和词法错误并报错
- 能够正确识别 8 进制，16 进制和带指数形式的浮点数
- 在程序正确时，能够正确打印语法树

4. 实验思路

1) 定义节点类型

如下图代码

```
%union{
|   struct value val;
|   node* _node;
}
```

```
typedef struct node
{
|   int level;
|   child_node *children;
|   char *code;
|   int lineno;
} node;
```

```
struct value
{
|   union {
|       int ival;
|       float fval;
|       char name[32];
|   };
|   int lineno;
};
```

```
typedef struct child_node
{
|   struct node *c;
|   struct child_node *next;
} child_node;
```

任一节点类型为 **struct value** 或 **node***，前者用于所有的终结符号，后者用于所有的非终结符号。终结符号如 **INT, FLOAT, ID** 等，它们的值分别存储在 **union** 中的 **ival, fval** 和 **name** 中。**lineno** 属性用于部分的非终结符号定位行号。**node** 结构体用于保存符号的结构信息，也是后面生成语法树的基础数据结构。属性 **level** 用于保存当前节点在树中的深度信息（从树的根节点开始，深度从 0 开始递增）。属性 **code** 用于保存当前节点的名称。**lineno** 用于保存节点的行号。某一节点的子节点用链表 **children** 存储。

2) 功能函数

在实验中实现了这几个函数：

```
node* node_con(char*);  
void add_child(node*, node*);  
void print_tree(node*, int);  
void free_all(node*);
```

函数 **node* node_con(char *)** 接受一个字符串指针为参数，用于生成一个新的节点并返回这个节点的指针。作为参数的字符串即为这个节点的名字（标识符）。函数 **void add_child(node*, node*)** 接受两个 **node*** 类型的参数，分别为父节点指针和子节点指针，函数将子节点添加到父节点中。函数 **void print_tree(node*, int)** 用于打印语法树，在所有的节点添加完成后调用这个函数，函数递归的打印树中所有的节点信息。参数分别为当前打印的节点和当前节点深度。函数 **void free_all()** 用于释放申请的内存。

3) 构造语法树

在文法匹配时调用 **node_con** 和 **add_child** 函数生成树节点并将节点添加到父节点中。当文法没有错误时程序最后打印语法树，当程序出错时打印出错信息。