编译原理实验一实验报告

191220163 计算机科学与技术系 张木子苗

完成的实验要求

在本次实验中，我完成了**全部的必做任务**和**绝大部分的选做任务。**其中选做部分完成了：

识别八进制数和十六进制数、识别指数形式的浮点数、识别“//”形式的注释（只能识别单行注释）

提供的测试方法

在Test目录下，我一共准备了9个代码文件用于测试：

`Test/test1-1.cmm` 到 `Test/test1-4.cmm` 即为必做内容的样例1到样例4，`Test/test2-1.cmm` 到 `Test/test2-4.cmm` 为选做内容的样例1到样例4。`Test/test2-5.cmm` 实际上就是 `Test/test1-4.cmm` 加上了一行注释，用于测试对单行注释的识别。

为了方便测试，我为我的代码写了个简单的脚本文件，`Code/Test.sh`，在Code目录下命令行界面输入：

```
./Test.sh x-x
```

即可自动编译并且执行对测试文件 `Test/testx-x.cmm` 的语法分析和词法分析，打印分析结果。

代码的实现思路

词法分析

在Flex的帮助下，实现词法分析只需要写好正则表达式即可。在我的代码中，我令识别到的每一个词法单元的属性为结构体 `Token`：

```
struct Token
{
    int col;
    int is_terminal;
    char token_type[100];
    char token_text[100];
    struct Token *firstChild;
    struct Token *nextSibling;
};
```

其中 col 为其出现的行号，is_terminal 代表其是否为终结符，token_type 为其在语法分析时对应的类型，token_text 为该词素对应的文本。firstChild 和 nextSibling 用于之后建立语法分析树。

每识别到一个词法单元，就调用函数 SetTokenValue 设置其属性，以便之后语法分析时使用，具体查看代码Code/lexical.l 的函数 SetTokenValue

语法分析

在写好文法之后，语法分析的代码可以由Bison帮我们自动生成，难点在于建立语法分析树。

由于我们使用的是“移入-规约”的方法进行语法分析，所以建树也是从根部开始的。我们使用一个产生式时，其产生式右边的符号所对应的节点必然已经建立完毕，此时我们只需要再建立一个节点，将它的col设置为产生式右边第一个文法符号的col，is_terminal设置为0，token_type设置为产生式左边的符号，token_text设置为空，再用孩子兄弟表示法，将该节点和它的第一个子节点，以及将其子节点连接起来即可。

通过以上的方法即可得到一棵语法分析树。如果程序没有错误，可以用递归的方法打印。

识别八进制数和十六进制数

在词法分析中，加入如下正则表达式：

```
OCT      (0[0-7]+)
OCTERROR (0[0-7]*[8-9]+[0-9]*)
HEX      ((0x|0X)[0-9a-fA-F]+)
HEXERROR ((0x|0X)[0-9a-fA-F]*[g-zG-Z]+[0-9a-zA-Z]*)
```

OCT 和 HEX 识别的是正确的八进制数和十六进制数，这时候只需要调用 SetTokenValue 函数，并且将其类型设置为INT即可；如果碰到错误形式的八进制数（0开头，却出现了8-9）或者错误形式的十六进制数（0x开头，却出现了g-z），那么词法分析就会报错，为了不影响后续的分析，我们也暂时返回了一个int类型的词法单元：

```
{OCT}   {SetTokenValue("INT",yytext);return INT;}
{OCTERROR} {
    Program_Is_Correct = 0;
    printf("Error type A at Line %d: Illegal octal number\'%s\'.\n",yylineno,
yytext);
    SetTokenValue("INT", "0");
    return INT; }
```

之后在打印语法分析树，将token_text转为int时，需要分情况讨论。

识别指数形式的浮点数

只需在浮点数的正则表达式中加入指数形式浮点数的正则表达式即可。如果碰到不合规则的指数显示浮点数，会被识别为多个此法单元而在语法分析时报错。

正则表达式为：`({digit}+\.{digit}*)([Ee]?[+-]?{digit}+)?`

识别“//”形式的注释

在词法分析时识别单行注释，每碰到 "//"，将这一行后的内容全部丢弃。

```
"//"    {char c = input(); while (c != '\n') c = input();}
```