实验 1——词法语法分析

本次完成的实验内容如下：

- 实现词法分析和语法分析器的基本功能
  - 识别未定义的字符和不符合词法单元定义的字符
  - 输出词法和语法错误类型和信息
  - 按照语法树遍历打印结点信息
- 实现词法分析和语法分析器的高级功能（选做）
  - 识别“//”和“/.../”形式的注释

实验特色：

因为不用实现浮点数的科学计数法，但需要将其识别出错误，故单独处理并不方便，于是我还是在词法中单独将其识别出来，并进行报错。
识别如下：

```
44 efloat   ({digit}*\.{digit}+|{digit}+\.)[eE][+-]?{digit}+
```

报错如下：

```
{efloat}        {
                        if(isNewError(yylineno)){
                                printf("Error type A at Line %d: syntax error at \"%s\".\n", yylineno, yytext);
                        }
                }
```

同时对 ID 的识别我是先进行全部接受，再直接判断首位是否为数字从而进行报错。

```
; ID        ({letter}|{digit}|_)+

{ID}    {
                //debug
                //printf("add in ID, %s\n",yytext);
                if(yytext[0] >= '0' && yytext[0] <= '9'){
                        if (isNewError(yylineno)) {
                                printf("Error type B at Line %d: syntax error at \"%s\".\n", yylineno, yytext);
                        }

                }
                yylval.type_string = strdup(yytext);
                return ID;
        }
```

对自己的选做部分注释的实现：

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

"/*"    {
            char c = input();
            while (1) {
                if (c == '*') {
                    c = input();
                    if (c == '/')
                        break;
                    else
                       unput(c);
                } else if (c == EOF) {
                    if (isNewError(yylineno)) {
                        printError('B', yylineno, "Unmatched \"/*\"");
                    }
                    break;
                }
                c = input();
            }
        }
"*/"    {
            if (isNewError(yylineno)) {
                printError('B', yylineno, "Unmatched \"*/\"");
            }
        }
```

其他词法照常从 C—文法中摘抄并仿照 PPT 格式书写即可。

语法树相关实现：

语法树节点定义（同 PPT）：

```
enum NodeType { NTML, NVL, VL };
struct Node {
    char* nodeName;
    enum NodeType nodeType;
    int lineNum;
    union {
        int Valint;
        float Valfloat;
        char* Valstr;
    };
    struct Node* firstChild;
    struct Node* bro;
};
```

建树：
利用了 va_list 实现参数的变化

```
void construct(struct Node* fatherNode, int index, ...) {
    va_list valist;
    va_start(valist, index);
    struct Node* firstChild = NULL;
    struct Node* lastChild = NULL;
    for (int i = 0; i < index; i++) {
        struct Node* nowNode = va_arg(valist, struct Node*);
        if (firstChild == NULL) {
            if (nowNode != NULL) {
                firstChild = nowNode;
                lastChild = firstChild;
            }
        } else {
            if (nowNode != NULL) {
                lastChild->bro = nowNode;
                lastChild = nowNode;
            }
        }
    }
    va_end(valist);
    fatherNode->firstChild = firstChild;
}
```

重写了 yyerror 进行报错：

```
void yyerror(const char* s) {
       if(isNewError(yylineno)){
              fprintf(stdout, "Error type B at Line %d: %s.\n", yylineno, yytext);
              errorflag = 1;
       }
}
```

实验感想：

一步步从什么也不清楚模糊探索到最后实现成功还是挺有成就感的，但是错误恢复部分确实不是很能摸到头脑，试了好多例子才最后成功，还有语法树的打印有点莫名其妙的，最开始怎么都过不去，最后破釜沉舟全部删掉重写就 pass 了，至今也不知道到底是什么原因，只能等实验结束之后放出样例再用保留的版本进行 debug 了。。。