编译原理实验一实验报告

马珩峻　林立强

一、程序功能

程序实现了对 c—语言的词法和语法分析，其中选做 1.2 识别指数形式浮点数

使用 flex 进行词法分析，bison 进行语法分析

其中 node.h 是对语法树的数据结构设计，通过将相关函数定义为 static inline 函数，减少调用开销，因为 lexical.l 和 syntax.y 中有大量构建节点函数的重复调用，通过内联省去函数调用额外开销。

语法书节点的数据结构如下

```
// node type declared
typedef struct node {
    int lineNo;  //  node in which line
    //   int depth;   //  node depth, for count white space for print
    NodeType type;  // node type
    char* name;     //  node name
    char* val;      //  node value

    struct node* child;  //  non-terminals node first child node
    struct node* next;   //  non-terminals node next brother node

} Node;

typedef Node* pNode;
```

节点没有记录高度，高度在打印时候递归调用中计算。

构建节点时，因为语法产生式产生的变量是不定的，所以使用了变长参数...，但构建节点时需要传入变长参数列表的长度，例如 ExtDecList 的产生式有两个，定义为这样:

```
ExtDecList:         VarDec                          { $$ = newNode(@$.first_line, NOT_A_TOKEN, "ExtDecList", 1, $1); }
    |               VarDec COMMA ExtDecList          { $$ = newNode(@$.first_line, NOT_A_TOKEN, "ExtDecList", 3, $1, $2, $3); }
    ;
```

然后在 newNode 中会遍历参数列表设置当前节点的 child 和子节点的 next

```
pNode tempNode = va_arg(vaList, pNode);

curNode->child = tempNode;

for (int i = 1; i < argc; i++) {
    tempNode->next = va_arg(vaList, pNode);
    if (tempNode->next != NULL) {
        tempNode = tempNode->next;
    }
}
```

enum.h 是对 enum 类型变量的声明，本实验中只有 enum nodeType 这一种枚举变量，主要用于定义节点的类型（整数，浮点，id，类型声明，其他 Token 和非终结符），声明类型的主要作用是为了在打印节点时候分别处理。

lexical.l 和 syntax.y 的编写主要参照了实验手册的说明，重点针对各种指数类型的浮点

数错误定义以及数字开头的非法 ID（如果不处理会将开头数字构建为一个 INT 节点识别为语法错误 unexpected INT）进行了处理。

```
{digit}+{ID} {lexError = TRUE; printf("Error type A at Line %d: Illegal ID \"%s\".\n", yylineno, yytext); }
"."{digit}+ { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
{digit}+"." { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
{digit}*"."{digit}+[eE] { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
{digit}+"."{digit}*[eE] { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
{digit}+[eE][+-]?{digit}* { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
"."[eE][+-]?{digit}+ { lexError = TRUE; printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext); }
. { lexError = TRUE; printf("Error type A at Line %d: Mysterious character \'%s\'.\n", yylineno, yytext); }
```

二、程序运行

使用课程网站提供的 Makefile 进行编译运行即可，实验文件夹根目录下编写了 test.py 脚本用于执行测试，脚本会将 Test 文件夹下所有文件作为程序参数列表遍历执行 Make 出来的 parser 程序。