实验一、词法分析与语法分析

姓名：瞿久尧　　学号：120L022314

1　功能介绍及实现

1.1 检查词法错误

对于 C—词法中未定义的字符以及任何不符合 C—词法单元定义的字符，程序会报 A 类错误，并在报错信息中显示出错行号和非法字符。

实现方式

编写 flex 的源程序 lexical.l，利用正则表达式匹配 C—中的所有合法字符，以下给出 INT、FLOAT、ID 以及空白符的正则表达式（其余类型较为简单此处省略），其中 INT 类型额外实现了八进制数和十六进制数的识别。

```
/* data */
FLOAT ({INT_D}\.[0-9]+)|([0-9]*\.[0-9]*[eE][+-]?[0-9]+)
INT_H 0[xX][0-9a-fA-F]+
INT_O 0[1-7][0-7]*
INT_D 0|[1-9][0-9]*
INT {INT_D}|{INT_O}|{INT_H}
```

对于未匹配的所有字符，采取以下规则输出，并设置 error_signal 为 1，不打印语法树。

```
. {error_signal = 1;
   printf("Error type A at line %d: Mysterious characters \'%s\'\n"
   , yylineno, yytext);}
```

此外，值得一提的是，在添加规则时，由于关键字类型是 ID 类型的子集，故将 ID 的规则放在倒数第二位以降低其匹配优先级，避免将关键字是被为 ID。

1.2 检查语法错误

对于不符合 C—语法的语句，程序会报 B 类错误，并在报错信息中显示出错行号。

实现方式

编写 bison 的源文件 syntax.y，根据指导书附录中给出的 C—语法规则添加产生式，并重写 yyerror()函数，使程序每次检查到语法错误后报 B 类错误并打印出错行号数(yylineno)，这里还额外添加输出了报错的位置，可以更方便的定位错误。

```
yyerror(char* msg)
{
    //printf("Error type B at Line %d: syntax error.", yylineno);
    printf("Error type B at Line %d: syntax error at \"%s\".\n", yylineno, yytext);
    error_signal = 1;
}
```

此外，为确保程序在遇到语法错误后能够进行错误恢复继续检查后序语法，需要在原文法产生式的基础上利用 bison 中的特殊符号 error 进行改写。以 Def 的产生式为例，可以做如下修改，使得以 Def 为根节点的子树出现错误时，程序会丢弃输入符号，直到遇到分号(SEMI)，再做规约并处理下一句。

```
Def:Specifier DecList SEMI // 原产生式
Def:Specifier DecList SEMI | error SEMI // 修改后产生式
```

1.3 构造并打印语法树

语法树节点的数据结构定义如下，所有的终结符和非终结符的数据类型均为该结构体指针类型。语法树采用孩子兄弟表示法。其中由于 ID、TYPE、INT 和 FLOAT 需要存储属性值，结构体中添加了 union 用于存储相应的属性值

```
typedef struct tree_node {
    int line;
    char* name;
    struct tree_node *fir_child, *bro;
    union {
        char* id_or_type;
        int it;
        float flt;
    };
} *ST, *Tnode;
```

此外，主要相关变量以及函数如下：

```
extern int error_signal;
extern int nodeNum;
extern Tnode nodeList[10000];
extern int nodeIsChild[10000];
void yyerror(char *msg);
// build new syntax tree
ST new_st(char *name, int num, ...);
// search syntax tree
void search_st(ST st, int level);
void setChildTag(Tnode node);
```

树节点构造函数 new_st()

它主要出现在词法分析的规则以及语法分析的动作中。对于每个终结符和非终结符，都要调用该函数构造相应树节点。在语法分析中，由于产生式右侧的符号数量不确定，即子节点数量不确定，故该函数采用变长参数，这里需要引入头文件<stdarg.h>。

在函数实现上，主要即是内存申请，符号名、属性值等赋值，若有子节点，则递归调用该函数继续构造树节点。

语法树遍历函数 search_st()

在程序对文件完成词法语法分析获得语法树后，程序会调用该函数打印语法树。打印语法树前，程序会检查 error_signal 来判断是否出现语法错误，若有语法错误，则只打印错误信息。否则，程序会首先通过 nodeIsChild 数组查找到根节点，再从根节点开始调用 search_st()函数，该函数采取深度优先遍历二叉树的策略，最终打印出完整语法树。

1.4 识别八进制和十六进制数

在“1.1 检查词法错误”中已经提到 INT 的正则表达式可以识别八进制和十六进制数。但还需要注意，在打印语法树时，由于 INT 类型需要打印属性值，而 atoi()不支持八进制或十六进制转十进制，故本程序实现了 my_atoi()函数实现三种进制字符串转十进制整型的功能。该函数实现流程是，首先通过识别前两个字符确定进制类型，在根据对应进制对每位字符计算再整体求和即可。

1.5 识别注释

注释的正则表达式以及动作如下，该正则表达式不接受“/**/”的嵌套。

```
/* comment */
COMMENT ("//".*)|("/*"[^(*/)]*"*/")
{COMMENT} {}
```

1.6 多文件处理

考虑到要验证多个文件，故 main()函数中会循环读取从 argv[1]开始的所有文件名并对其进行分析。需要注意的是，在每次循环开始，需要对部分变量进行重置，具体如下：

```
nodeNum = 0;
memset( Dst: nodeList,  Val: 0,  Size: sizeof(Tnode)*10000);
memset( Dst: nodeIsChild,  Val: 0,  Size: sizeof(int)*10000);
error_signal = 0;
yylineno = 1;
```

2 编译过程

源文件共包含 lexical.l、syntax,y、syntax_tree,h、syntax_tree.c 四个文件。首先利用 bison -d syntax.y 生成 syntax.tab.c、syntax.tab.h 文件，因为 lexical.l 文件依赖于 syntax.tab.h 头文件中的 Tnode 数据类型定义，此时已经得到主要的功能函数 yyparse()。然后利用 flex lexical.l 得到 lex.yy.c 文件。最后利用 gcc syntax.tab.c syntax_tree.c lex.yy.c -lfl -ly -lm -o parser 进行编译，其中添加-lm 是因为进行 INT 属性值计算时要用到<math.h>库函数。为方便使用，后期编写了 makefile 和 test，只需./test 即可重新编译并运行所有测试样例：

```
$(target):$(cfile)
    gcc $^ -lfl -ly -lm -o $@
$(bison_target):$(bison_file)
    bison -d syntax.y
lex.yy.c:$(flex_file)
    flex lexical.l
```

3 测试结果

本程序对指导书中的 10 个样例均进行了测试并获得了正确的结果，此处仅展示部分重要结果，您可以运行./test 来直接编译并检查测试样例，若需要，并可以将您想测试的样例命名为 testx.cmm(1~10)来检查，或./parser testx.cmm

测试的结果也会同时输出到文件 result.txt 中以便您进一步检查。

1) test1 词法错误：

```
/********** test1.cmm **********/
Error type A at line 4: Mysterious characters '~'
```

2) test2 多处语法错误：

```
/********** test2.cmm **********/
Error type B at Line 5: syntax error at ",".
Error type B at Line 6: syntax error at "else".
```

3) test4 和 test7(科学计数)浮点数识别：

```
Exp(8)
  FLOAT: 3.500000
```

```
Exp(3)
  FLOAT: 0.000105
```

4) test5 八进制和十六进制数识别：

```
Exp(3)
  INT: 83
```

```
Exp(4)
  INT: 63
```

5) test10"/**/"嵌套语法错误：

```
/********** test10.cmm **********/
Error type B at Line 3: syntax error at "/".
```

6) test9 注释测试生成语法树：(见右侧)

```
/********** test9.cmm **********/
Program(1)
  ExtDefList(1)
    ExtDef(1)
      Specifier(1)
        TYPE: int
      FunDec(1)
        ID: main
        LP
        RP
      CompSt(2)
        LC
        DefList(7)
          Def(7)
            Specifier(7)
              TYPE: int
            DecList(7)
              Dec(7)
                VarDec(7)
                  ID: i
                ASSIGNOP
                Exp(7)
                  INT: 1
            SEMI
        RC
```