Task1：词法分析器

学号：21311080

姓名：余俊欣

实验步骤及结果

- 本次实验需要补全词法分析器的代码从而得到正确结果，选用flex框架实现，主要修改以下四个文件的代码
    - lex.l：补充词法分析器的token匹配规则，以及对于不同token调用lex.cpp的函数进行信息（行，列等）更新
    - lex.hpp：在Id枚举类型中添加新加入的token名，以及新函数的声明
    - lex.cpp：添加信息更新函数
    - main.cpp：处理输出数据。
- lex.l文件代码解释（为缩短篇幅，仅放出添加的代码部分）
    - 首先定义部分，删除了原先的ADDCOL()函数，合并到COME()内，然后新添加了两个函数，GNF()用于处理文件路径，SC()用于处理空白符的行列信息变换，以及添加了F用于识别十六进制数

```
%{
...
#define COME(id) return come(id, yytext, yyleng, yylineno)
#define GFN() getfilename(yytext,yylineno)
#define SC() spacecontrol(yytext)
%}
F     [0-9a-f]
...
```

    - 规则部分只需要根据输入样例增加token类即可，仅展示部分重要例子

```
"<="    {  COME(LESSEQUAL); }
">="    {  COME(GREATEREQUAL); }
"!="    {  COME(EXCLAIMEQUAL); }
"=="    {  COME(EQUALEQUAL); }
"="     {  COME(EQUAL); }
"!"     {  COME(EXCLAIM); }
">"     {  COME(GREATER); }
"<"     {  COME(LESS); }
"||"    {  COME(PIPEPIPE); }
"&&"    {  COME(AMPAMP); }
"|"     {  COME(PIPE); }
"&"     {  COME(AMP); }
0x{F}*{IS}?    {  COME(CONSTANT); }/* 处理十六进制数*/
```

```
^#[^\n]*    { GFN();return ~YYEOF; } /* 预处理信息处理*/
[ \t\v\n\f]    { SC();return ~YYEOF; } /* 需要处理行号和列号信息 */
```

- lex.hpp
  - 在枚举类型中添加相应的token类型名即可，需要注意顺序要与lex.cpp中的kTokenNames数组的类型名顺序一致。最后添加getfilename()和spacecontrol()函数的声明。

```
enum Id
{
  YYEMPTY = -2,
  YYEOF = 0,     /* "end of file"  */
  YYerror = 256, /* error  */
  YYUNDEF = 257, /* "invalid token"  */
  IDENTIFIER,
  CONSTANT,
  STRING_LITERAL,
  INT,
  VOID,
  CONST,
  RETURN,
  IF,
  ELSE,
  WHILE,
  ...
};
...

void getfilename(const char* yytext, int yyleng);
void  spacecontrol(const char* yytext);
```

- lex.cpp
  - 首先需要在kTokenNames数组中添加要输出的token类名，顺序与Id保持一致：

```
static const char* kTokenNames[] = {
    "identifier",    "numeric_constant", "string_literal", "int",
    "void",          "const",            "return",         "if",
    "else",          "while",            "break",          "continue",
    "l_brace",       "r_brace",          "l_square",       "r_square",
    "l_paren",       "r_paren",          "semi",           "lessequal",
    "greaterequal",  "exclaimequal",     "equalequal",     "equal",
    "exclaim",       "greater",          "less",           "pipepipe",
    "ampamp",        "pipe",             "amp",            "plus",
    "minus",         "star",             "slash",          "percent",
    "comma"
};
```

- 其次修改come()函数和添加getfilename()和 spacecontrol()函数的实现，函数解释和代码如下：
  - getfilename()：该函数用于处理预处理信息，与处理信息中的有用信息有两个，一个是开头的数字，标识该预处理信息的下一行代码是第几行代码，其次是""内的文件路径，因此首先需要先从预处理信息中获取一个十进制整数，然后再获取文件名。由于该数字需要在come中用到，因此设置为全局变量brows，为方便后续使用，将brows设定为实际行数与设定行数（也就是预处理信息中的十进制整数）的差值。代码如下：

```
int brows=0;
void getfilename(const char* yytext, int yylineno){
  int i=0;
  std::string s;
  //找到整数位置
  while(yytext[i]<'0'||yytext[i]>'9')i++;
  brows=0;
  //按位读取整数
  while(yytext[i]>='0'&&yytext[i]<='9'){
    brows*=10;
    brows+=yytext[i++]-'0';
  }
  brows=yylineno-brows+1;
  //找到文件路径
  while(yytext[i]!='\"'){
    i++;
  }
  i++;
  //获取文件路径
  while(yytext[i]!='\"'){
    s+=yytext[i++];
  }
  g.mFile=s;
}
```

  - spacecontrol()：根据词素类型选择列信息g.mColumn的更新方式，用switch case语句实现。

```
void spacecontrol(const char* yytext){
    switch (*yytext)
    {
    //空格列数加1，且前导空格标志置为true
    case ' ':
      g.mLeadingSpace = true;
      g.mColumn+=1;
      break;
    //制表符向上取至8的整数倍加1，且前导空格置位true
    case '\t':
      g.mLeadingSpace = true;
      g.mColumn=(g.mColumn-1)/8*8+9;
      break;
```

```
    //换行符将列信息重置为1
    case '\n':
      g.mColumn=1;
      break;
    default:
      break;
    }
}
```

- come():首先需要处理行信息，当前的行信息应该为实际行信息减去brows，然后判断当前词素的行信息与上一个词素的行信息是否相同，从而判断该次数是否为本行第一个词素，更新行信息然后用print_token()函数进行输出，输出之后再更新列信息以及重置前导空格标志。

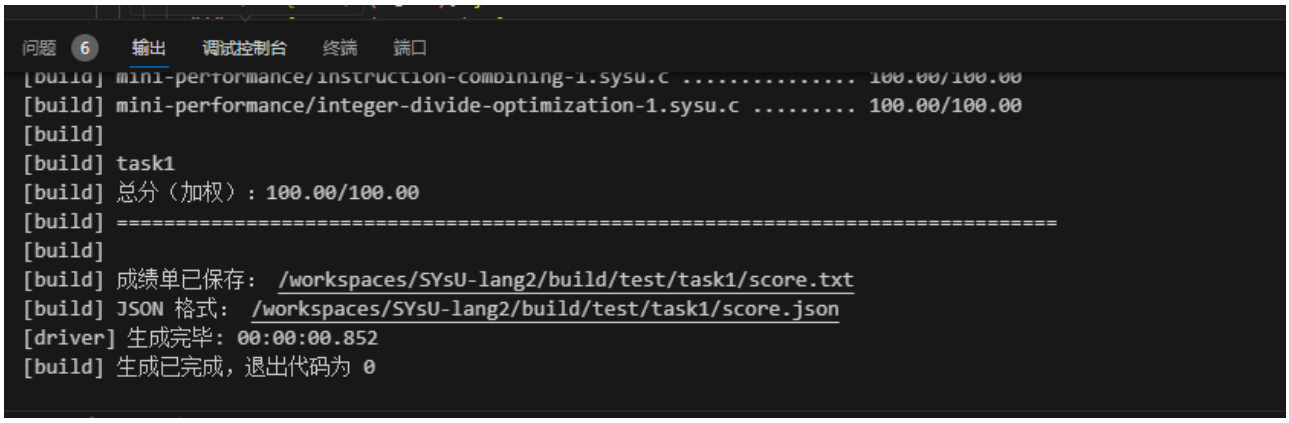
```
int come(int tokenId, const char* yytext, int yyleng, int
yylineno)
{
  g.mId = Id(tokenId);
  g.mText = { yytext, std::size_t(yyleng) };
  //计算当前行信息
  yylineno-=brows;
  //判断该词素是否为本行第一个词素
  g.mStartOfLine = g.mLine!=yylineno;
  //更新行信息
  if(g.mStartOfLine) {
    g.mLine = yylineno;
  }
  print_token();
  //更新列信息以及重置前导空格标志
  g.mColumn+=yyleng;
  g.mLeadingSpace = false;

  return tokenId;
}
```

- main.cpp:主要为修改print_token()函数输出文件路径和行列信息。

```
void
print_token()
{
  ......
  outFile << " \tLoc=<"<<lex::g.mFile<<":"<<lex::g.mLine<<":"
<<lex::g.mColumn<<">"<<"\n";
  outFile << std::flush;
}
```

- 最终结果：



实验感想

> 主要是不清楚flex框架的结构以及每个函数的用意，导致刚上手不知道要干什么，以及一开始以为是要调用什么函数来获取文件名，慢慢摸索才搞清楚需要自己写函数获取文件路径。以及非常坑的一点是，一开始不清楚预处理信息前后为什么有数字，实际跑起来才知道要根据预处理信息的数字计算行数。
>
> 实际做的过程中出现了一个很奇怪的错误，即将ADDCOL()放在COME后面(因为需要先输出列信息然后再更新列信息)会出现列信息更新异常，即g.mColunm始终为1，解决无果干脆就直接把ADDCOL删掉合并进了come函数中就解决了。总的来说本次实验上手理解需要点时间，实际做起来不难，问题都能自己解决。