Project 1词法分析实验报告

小组：罗旭川、万伟霖　　日期：2020.10.21

一、flex的用法

flex是一个能够进行词法分析的自动化工具。

1. 文件说明与文件之间的关系

- `lexer.h`

  定义了**所有token的类型编号、所有错误的类型编号**。

  为了更好的区分是否出错，我们将token的类型编号定义为大于0的整数；将错误编号定义为小于0的整数；而文件结束EOF则用0表示。

- `lexer.lex`

  定义了词法规则，也就是token的匹配规则。具体书写语法在下文给出。

  通过`flex`指令就可以将该文件转化为C程序 (在本实验中我们命名为`lexer.c`)，`lexer.c`中提供了许多可以供我们直接调用的函数和变量，利用它们可以实现词法分析过程，具体用到的函数和变量在下文给出。

- `main.cpp`

  主程序。用来开启词法分析过程，并进行输出规范。

  具体就是调用由`lexer.c`中提供的函数与变量来得到词法分析结果。

- `Makefile`

  为了自动化编译、链接过程，将编译、链接的执行指令写成了`Makefile`文件。

  具体行为就是，通过`flex`指令能将写好的词法规则文件`lexer.lex`转化为C程序`lexer.c`，然后将`lexer.c`编译为`lexer.o`后，用写了主程序的`main.cpp`链接它。这样一来，`main.cpp`中可以成功调用`lexer.c`中提供的相关函数、变量来实现词法分析。

2. `lexer.lex`文件的基本语法

```
[定义段]
%%
[规则段]
%%
[用户代码段]
```

- **定义段**

  包含启动选项和名字定义。

  - 启动选项，用`%option xxx`表示，本实验中用`%option nounput`和`%option noyywrap`来取消生成的`lexer.c`中的`nounput()`和`yywrap()`函数。
  - 名字定义，类似于C语言中的宏定义，用于简化规则段中的正则表达式。

    如定义了`WS [ \t]+`后，之后的正则表达式中就可以用`{WS}`来替代`[ \t]+`。

- **规则段**

由若干条词法规则构成。每条词法规则形式如下：

```
正则表达式  {C代码块}
```

每当当前token匹配上该规则的正则表达式时，就会执行其后的C代码块。

正则表达式中可以使用定义段中的名字定义。C代码块最终会被拷贝到 `lexer.c` 文件中，因此在它里面可以使用 `lexer.c` 中提供的变量，如 `yytext` 、`yyleng` 等。

- **用户代码段**

  主程序代码可以写在这里，但本实验中将主程序代码集中写在了 `main.cpp` 中，因此这里置空。

- **插入段**

  在定义段和规则段中，其实还可以随时插入C代码块，用 `%{` 和 `%}` 包含起来即可。这些C代码块也会被拷贝到生成文件 `lexer.c` 中。

3. `lexer.c` 中提供的函数与变量

只列举本次实验用到的函数、变量：

- `int yylex()` 进行词法分析，**每次调用都会匹配出下一个符合词法的token**，返回值由词法文件 `lexer.lex` 规则段中相应的词法正则表达式后的C语句块提供，在**实验中我们总是返回匹配出的token的类型编号或错误的类型编号**；
- `FILE *yyin` 是指向输入文件的指针，词法分析过程默认从这个指针指向的文件来读取文件；
- `FILE *yyout` 是指向输出文件的指针，通过 `ECHO` 指令可以将TOKEN输出到该文件中，但是为了打印出更加个性化的输出到文件中，实验中我们只是简单地将 `stdout` 重定向到这个指针中；
- `char *yytext` 当前匹配到的token值；
- `int yyleng` 当前匹配到的token的长度。

二、代码执行方法

```
make
./main
```

执行完后 `case_1` 到 `case_11` 的结果就会生成在 `/output` 文件夹中。

三、功能实现原理

1. 正则表达式

- `{NL}`

  表示换行。

  这里使用了名字定义 `NL [\n]`。

- `{WS}`

  表示若干空白符。

  这里使用了名字定义 `WS [ \t]+`。

- `<<EOF>>`

  表示文件末尾。

- `{DIGIT}+`

  表示整数 `integer` 。

  即由一个或多个数字构成整数，使用了名字定义 `DIGIT [0-9]`。由于 PCAT语言中未定义负数，因此无需考虑负数。**注意这里还包含了“过大的整数”，作为非法整型，它的区分将放在C代码块中进行**。

- `{DIGIT}+"."{DIGIT}*`

  表示实数 `real`。

  即包含一个小数点的数，小数部分可以省略。

- `"\""[^\"\n]*"\""`

  表示字符串 `string`。

  前后用 `"\""` 匹配前后双引号，并用 `\` 转义；中间 `[^\"\n]*` 若干个不包含双引号和换行符的字符。**注意这里还包含了“过长的字符串”和“包含\t的字符串”，作为非法字符串，它们的区分将放在C代码块中进行**。

- `{LETTER}({LETTER}|{DIGIT})*`

  表示标识符 `identifier`。

  即由字母开头的、由字母、数字组成的串。这里用到了名字定义 `LETTER [A-Za-z]`。**注意这里还包含了“过长的标识符”，作为非法标识符，它的区分将放在C代码块中进行**。

- `":="|"+"|"-"|"*"|"/"|"<"|"<="|">"|">="|"="|"<>"`

  表示运算符 `operator`。

- `":"|";"|","|"."|"("|")"|"["|"]"|"{"|"}"|"[<"|">]"|"\\"`

  表示分隔符 `delimiter`。

- `"(*"([^\*]*|[^\)]*|({ANY}*\*{ANY}+\){ANY}*))"*)"`

  表示注释 `comment`。

  前后分别用 `"(*"` 和 `"*)"` 匹配注释的前后表示符号；中间又分三部分，即 `[^\*]*`、`[^\)]*`、`({ANY}*\*{ANY}+\){ANY}*)`，分别表示不包含 `*` 的序列串、不包含 `)` 的序列串、同时包含 `*` 和 `)` 但二者不紧靠的序列串。**这里用到了名字定义 `ANY [\s\S]`，表示包含换行符的任意字符，因为注释允许中间存在换行**。

- `"\""[^\"\n]*`

  表示未写完的字符串。

  其实就是将上面字符串的正则表达式去掉右双引号得到。由于该规则放在后面，因此不会影响前面正常字符串的匹配。

- `"(*"([^\*]*|[^\)]*|({ANY}*\*{ANY}+\){ANY}*))`

  表示未写完的注释。

  其实就是将上面注释的正则表达式去掉右边注释号得到。由于该规则放在后面，因此不会影响前面注释的匹配。

- `.`

  由于放在最后，这里表示所有前面未匹配上的字符。

2. 行、列号计算

维护两个变量 `row` 和 `col` 即可。初始值都为1。在本实验中，它们的定义用插入段的形式写在了 `lexer.lex` 的定义段中。这样就既可以在规则段中使用它们，也可以在 `main.cpp` 中声明和使用它们了。

维护方法：

- 当匹配到换行符时：`row ++; col = 1;`
- 当匹配到其他所有token或空白符时：`col += yyleng;`
- 当匹配到EOF时，要重新初始化：`row = col = 1;`
- 因为注释中允许换行，因此**当匹配到注释时要特殊处理**：

  ```
  if(type == COMMENT)
  {
      for(int i = 0; i < yyleng; i ++)
      {
          if(yytext[i] == '\n') row ++, col = 1;
          else col ++;
      }
  }
  ```

  即通过遍历匹配到的token来统计其中的换行以及列号的变化。

3. 类型判断

词法分析器匹配到 `lexer.lex` 中的正则表示式后，会执行正则表达式后的C代码块，在本实验中，我们总是在正则表达式后的C代码块中 **返回匹配到的token类型编号或出现的错误编号**，而主程序 `main.cpp` 通过调用 `yylex()` 是可以得到这个返回值的。这时就可以通过编号来获得当前 token 的类型，或者当前遇到的是什么错误类型了。

4. 报错功能

本实验中一共有7种错误类型，和token类型一样，将它们在 `lexer.h` 中进行宏定义（用负值表示），并作为 `lexer.lex` 中C代码块的返回值，这样就可以像token类型判断一样类似的做法来进行错误类型判断了。

具体的错误和判定方法如下：

- `out of range integer`

  ```
  if(yyleng > 10 || atoll(yytext) >= (1LL << 31)) return INT_OUT_OF_RANGE;
  ```

  即当数值超过 2^31 - 1 时出现了整型超出范围的错误。为了防止超 `long long` 类型，这里先进行了一次长度的判断。

- `invalid string with tab in it`

  ```
  for(int i = 0; i < yyleng; i ++) if(yytext[i] == '\t') return STR_WITH_TAB;
  ```

  即简单地遍历一遍字符串来判断其中是否包含 `\t`。若包含了`\t`，则是非法字符串。

- `overly long string`

```
if(yyleng - 2 > 255) return STR_OVER_LONG;
```

长度超过255的字符串为非法字符串，注意要先减去匹配到的两个引号的长度。

- `overly long identifier`

```
if(yyleng > 255) return ID_OVER_LONG;
```

长度超过255的标识符为非法标识符。

- `bad character`

```
if(!isprint(yytext[0])) return BAD_CHAR;
```

不可打印字符即为坏字符。

- `unterminated string`

  就如前面正则表达式所说，匹配上 `"\""[^\"\n]*` 的token即为未写完的字符串。

- `unterminated comment`

  就如前面正则表达式所说，匹配上`"(*"([^\*]*|[^\)]*|({ANY}*\*{ANY}+\){ANY}*))` 的token即为未写完的注释。

四、组员与分工贡献百分比

姓名	学号	贡献百分比
罗旭川	17307130162	50%
万伟霖	17307130106	50%