实验一:词法分析

一.实验要求

- 使用Docker配置实验环境
- 完成词法分析实验,因为这个实验暂时没有什么卷头,所以完成要求的任务即可

二.实验器材

- 笔记本电脑一台，性能足够强大
- WSL系统
- Docker Desktop

三.实验过程

0.环境的配置

- 首先在WSL中把Docker镜像给pull下来

```
truestar@LAPTOP-8BPRCOHI:~$ docker pull wukan0621/sysu-lang
Using default tag: latest
latest: Pulling from wukan0621/sysu-lang
1e4aec178e08: Pull complete
75bfae38466a: Pull complete
5a95e753337e: Pull complete
ea5a365c209a: Pull complete
b7f58ca1da2d: Pull complete
Digest:
sha256:1bbb905b09cd57974d6c2eb929769135877fa6e10a60317912d87e7476ccd042
Status: Downloaded newer image for wukan0621/sysu-lang:latest
docker.io/wukan0621/sysu-lang:latest
```

- 启动并进入docker镜像,并且进入workspace文件夹,下载实验代码

```
truestar@LAPTOP-8BPRCOHI:~$ docker run \
>   --name sysu-lang \
>   -v "$PWD/workspace:/workspace" \
>   -it wukan0621/sysu-lang \
>   bash
root@df9b607b60d2:/workspace# ls
root@df9b607b60d2:/workspace# git clone https://github.com/arcsysu/SYsU-lang
Cloning into 'SYsU-lang'...
remote: Enumerating objects: 1845, done.
remote: Counting objects: 100% (1845/1845), done.
remote: Compressing objects: 100% (851/851), done.
remote: Total 1845 (delta 858), reused 1800 (delta 837), pack-reused 0
Receiving objects: 100% (1845/1845), 1.11 MiB | 4.25 MiB/s, done.
Resolving deltas: 100% (858/858), done.
root@df9b607b60d2:/workspace# cd SYsU-lang
```

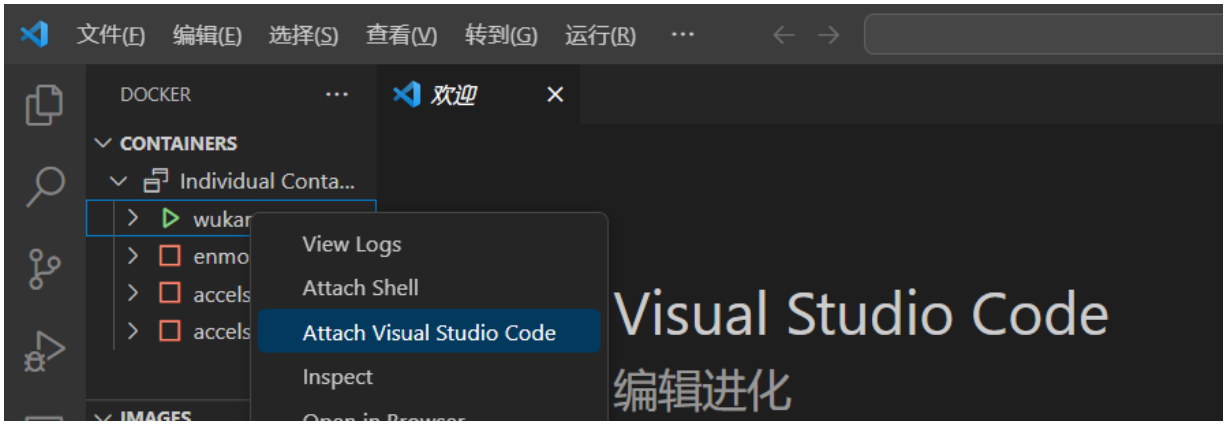
- 安装.注意每次改过需要重新执行这些命令.

```
root@df9b607b60d2:/workspace/SYsU-lang# rm -rf $HOME/sysu
root@df9b607b60d2:/workspace/SYsU-lang# cmake -G Ninja \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
  -DCMAKE_C_COMPILER=clang \
  -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_INSTALL_PREFIX=$HOME/sysu \
  -DCMAKE_PREFIX_PATH="$(llvm-config --cmakedir)" \
  -DCPACK_SOURCE_IGNORE_FILES=".git/;tester/third_party/" \
  -B $HOME/sysu/build
......
root@df9b607b60d2:/workspace/SYsU-lang# cmake --build $HOME/sysu/build
[20/20] Linking CXX executable optimizer/sysu-optimizer
root@df9b607b60d2:/workspace/SYsU-lang# cmake --build $HOME/sysu/build -t
install
[0/1] Install the project...
......
```

- 试一下下载下来的东西能不能用

```
root@df9b607b60d2:/workspace/SYsU-lang#  ( export PATH=$HOME/sysu/bin:$PATH
CPATH=$HOME/sysu/include:$CPATH   LIBRARY_PATH=$HOME/sysu/lib:$LIBRARY_PATH
LD_LIBRARY_PATH=$HOME/sysu/lib:$LD_LIBRARY_PATH &&   sysu-preprocessor
tester/functional/000_main.sysu.c |   sysu-lexer )
int 'int'               Loc=<tester/functional/000_main.sysu.c:1:1>
identifier 'main'               Loc=<tester/functional/000_main.sysu.c:1:5>
l_paren '('             Loc=<tester/functional/000_main.sysu.c:1:9>
r_paren ')'             Loc=<tester/functional/000_main.sysu.c:1:10>
l_brace '{'             Loc=<tester/functional/000_main.sysu.c:1:11>
return 'return'         Loc=<tester/functional/000_main.sysu.c:2:5>
numeric_constant '3'            Loc=<tester/functional/000_main.sysu.c:2:12>
semi ';'                Loc=<tester/functional/000_main.sysu.c:2:13>
r_brace '}'             Loc=<tester/functional/000_main.sysu.c:3:1>
eof ''          Loc=<tester/functional/000_main.sysu.c:3:2>
```

- 用VS Code连接到Docker容器,开发变得十分简单



1.词法分析实验

- 因为模板也有了,要求完成的token列表也有了,可以直接开搞

这里分为2个层次讲解实验过程

(1)实验要求1和2的完成

这一步主要就是模仿模板,加入新的token就可以

- 大部分工作由如下的脚本完成
- 助教内心OS:你管这叫脚本??????

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    ifstream fin("tokens.txt");
    ofstream fout("rawans.txt");
    string token,name;
    while(fin>>token>>name)
    {
        fout << token<<" {\n";
        fout << "  std::fprintf(yyout, \"" <<name << " \'%s\'\\t\\tLoc=<%s:%d:%d>\\n\",\n";
        fout << "               yytext,yyloc.c_str(), yyrow, yycolumn - yyleng);\n";
        fout << "  return ~YYEOF;\n";
        fout << "}\n";
    }
    return 0;
}
```

本次实验存在三个大坑,

- 第一个坑在于有的token不在那个表里面,比如do,long,float等
- 第二个坑在于各种奇形怪状的数字,比如说2022分组的95号样例

```
const float RADIUS = 5.5, PI = 03.141592653589793, EPS = 1e-6;

// hexadecimal float constant
const float PI_HEX = 0x1.921fb6p+1, HEX2 = 0x.AP-3;

// float constant evaluation
const float FACT = -.33E+5, EVAL1 = PI * RADIUS * RADIUS, EVAL2 = 2 * PI_HEX * RADIUS, EVAL3 = PI * 2 * RADIUS;

// float constant implicit conversion
const float CONV1 = 233, CONV2 = 0xfff;
const int MAX = 1e9, TWO = 2.9, THREE = 3.2, FIVE = TWO + THREE;
```

- 可以发现,不但出现了常用的整数(%d),小数(%f)(但是给的token里面只有intconst).还出现了十六进制数(%x,%X)和e计数法(%e.%E).甚至出现了极为冷门的十六进制小数p计数法!(%a,%A).甚至小数点前面

的0可以省略被!最后写出来的正则表达式是这样的

```
[0-9]+(\.[0-9]*)?((e|E)(\+|-)?[0-9]+)?|0(x|X)[0-9a-fA-F]+(\.[0-9a-fA-F]*)?((p|P)(\+|-)?[0-9a-fA-F]+)?|\.[0-9]+((e|E)(\+|-)?[0-9]+)?|0(x|X)(\.[0-9a-fA-F]+)((p|P)(\+|-)?[0-9a-fA-F]+)?
```

- 第三个坑在于字符串字面量,是两个最为邻近的双引号标识的区域,但是`\"`不算双引号.(`mizuno_ai`特别算例)这里是最困难的地方.最终正则表达式如下:
  - c - Regular expression for a string literal in flex/lex - Stack Overflow

```
\"(\\.|[^"\\])*\"
```

这三个问题解决了,就可以通过算例一和二了

(2)实验要求3的完成

实验要求三要求识别每一个token前面有没有空格,是否位于一行的首部.那么对于每一种token就有四种状态.

- 反正就四种,写死算了,string慢的要死,sprintf也不算快

```
char  sol[] = "[StartOfLine] ";
char  ls[] = "[LeadingSpace] ";
char solls[] = "[StartOfLine] [LeadingSpace] ";
char nosp[] = "";
char * loctext;
```

- 增加一个辅助变量newrow标识是否在新的一行,初始状态或遇到换行符置1,遇到其他token置0
- 一个辅助函数处理相关问题

```
void checker()
{ loctext = nosp;
  if(newrow == 1)
  { loctext = sol;
      if(yycolumn - yyleng != 1)
        loctext = solls;
    newrow = 0;
  }
  else
  {
    if(yycolumn - yyleng != yycolpre)
      loctext = ls;
  }
  yycolpre = yycolumn;
}
```

- 然后稍微改一下输出(查找替换就行)

```
\.\.\. {
  checker();
  std::fprintf(yyout, "ellipsis '%s'\t\t%sLoc=<%s:%d:%d>\n", yytext,loctext,
               yyloc.c_str(), yyrow, yycolumn - yyleng);
  return ~YYEOF;
}
```

就完成了要求三

四.实验结果

- 词法分析基础实验全部完成

```
root@df9b607b60d2:/workspace/SYsU-lang# CTEST_OUTPUT_ON_FAILURE=0 cmake --build $HOME/sysu/build -t test
[0/1] Running tests...
Test project /root/sysu/build
      Start  1: lexer-0
 1/10 Test  #1: lexer-0 ..............................   Passed    0.11 sec
      Start  2: lexer-1
 2/10 Test  #2: lexer-1 ..............................   Passed   25.36 sec
      Start  3: lexer-2
 3/10 Test  #3: lexer-2 ..............................   Passed   23.84 sec
      Start  4: lexer-3
 4/10 Test  #4: lexer-3 ..............................   Passed   23.65 sec
```

- (余下内容省略,因为还没做的实验当然不能通过了)

五.实验感想

- 第一个实验起手还是比较简单的(是怕上来就很难然后吓跑了大家吗hhh)
- 稍微难一点的就是正则文法吧,前面都有提到了
- 这真没什么感想.要是这都一堆感想,马上语法分析不得爆炸啊