Lab 1 词法分析器　2023.3.13

20337256_徐浩

1. 实验目的

完成一个词法分析器，产生与 `clang -cc1 -dump-tokens 2>&1` 相当的内容。

2. 实验过程

使用docker+git配置好实验环境后，我们在 `/workspace/SYsU-lang` 目录下继续我们的实验。

我参照了大量的资料来了解此次实验，参照信息如下: 词法分析， flex指南，实验一词法分析wiki，etc.

大概了解了此次实验的任务，就是在 `lexer.l` 模板的基础上，增加正则表达式，使得由其生成的 `sysu-lexer` 达到与 `clang -cc1 -dump-tokens 2>&1` 同样的词法分析效果。

在编写正则表达式的过程中，我主要碰到了如下问题：

- 问题一：关于评测样例的检验

（个人认为**实验指南可以增加有关linux命令行的使用教程链接。**如下：linux 命令表）

为了更加简便地对照 `clang -cc1 -dump-tokens 2>&1` 的标准输出，与 `sysu-lexer` 输出的差别，我们可以自己写出如下命令：

```
# 每一次改动lexer.l后，重新生成新的sysu-lexer
rm -rf $HOME/sysu
cmake -G Ninja \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
  -DCMAKE_C_COMPILER=clang \
  -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_INSTALL_PREFIX=$HOME/sysu \
  -DCMAKE_PREFIX_PATH="$(llvm-config --cmakedir)" \
  -DCPACK_SOURCE_IGNORE_FILES=".git/;tester/third_party/" \
  -B $HOME/sysu/build
cmake --build $HOME/sysu/build
cmake --build $HOME/sysu/build -t install

# 对于单个评样例，我们可以通过创建两个`comp_clang.txt`与`comp_sysu.txt`文件，将clang 与sysu-lexer 词法分析的输出分别重定向到它们中，再用diff命令比较两个文件中的差异。
( 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
/workspace/SYsU-lang/lexer/test.sysu.c | \
clang -cc1 -dump-tokens   &> /workspace/SYsU-lang/lexer/comp_clang.txt )

( 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
/workspace/SYsU-lang/lexer/test.sysu.c | \
sysu-lexer  &> /workspace/SYsU-lang/lexer/comp_sysu.txt )

diff /workspace/SYsU-lang/lexer/comp_clang.txt /workspace/SYsU-
lang/lexer/comp_sysu.txt &> /workspace/SYsU-lang/lexer/di.txt

# 评测tester文件夹中的所有样例
( 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-compiler --unittest=lexer-1 "/workspace/SYsU-lang/tester/*/*.sysu.c" )

# 评测所有的lexer[0-3]
CTEST_OUTPUT_ON_FAILURE=1 cmake --build $HOME/sysu/build -t test
```

- 问题一：实验提供的有效token表不全。

实验给出的有效token表为：

```
token       name

const       const
,           comma
;           semi
int         int
[           l_square
]           r_square
=           equal
{           l_brace
}           r_brace
ident       identifier
(           l_paren
)           r_paren
void        void
if          if
else        else
while       while
break       break
continue    continue
return      return
intconst    numeric_constant
+           plus
-           minus
!           exclaim
*           star
/           slash
%           percent
```

```
<           less
>           greater
<=          lessequal
>=          greaterequal
==          equalequal
!=          exclaimequal
&&          ampamp
||          pipepipe
string      string_literal
char        char
```

在实际实验过程中，我实现的有效token表应该如下：

```
token           name            regular expression(by me)

void            void            void
const           const           const
char            char            char
long            long            long
int             int             int
float           float           float
double          double          double
if              if              if
else            else            else
do              do              do
while           while           while
break           break           break
continue        continue        continue
return          return          return
(               l_paren         \(
)               r_paren         \)
[               l_square        \[
]               r_square        \]
{               l_brace         \{
}               r_brace         \}
,               comma           ,
;               semi            ;
=               equal           =
!               exclaim         !
+               plus            "+"
-               minus           -
*               star            "*"
/               slash           "/"
%               percent         %
<               less            "<"
>               greater         ">"
<=              lessequal       "<="
>=              greaterequal    ">="
==              equalequal      "=="
!=              exclaimequal    "!="
```

```
&&              ampamp          "&&"
||              pipepipe        "||"
...             ellipsis        "..."
identifier      identifier          [a-zA-Z_][a-zA-Z_0-9]*

b_num(2)        numeric_constant    0[bB][01]+
d_num(10)       numeric_constant    ([0-9]*\.[0-9]+|[0-9]+(\.[0-9]*)?)([eE][-+]?[0-
9]+)?
h_num(16)       numeric_constant    0[xX]([0-9a-fA-F]*\.[0-9a-fA-F]+|[0-9a-fA-F]+(\.
[0-9a-fA-F]*)?)([pP][-+]?[0-9]+)?

eof             eof                 <<EOF>>
string          string_literal      \"[^"\\]*((\\[onrtvabf'"\\?])*[^"\\]*)*\"
char            char                \'[^"\\]|(\\[onrtvabf'"\\?])\'

unknown         unknown             .
```

- 问题二：浮点数，科学计数法的表示。

如何写出与clang对标的科学计数法的正则表达式是一个折磨人的问题，在实现过程中，我发现一些有趣的东西：

1. clang 无法识别负数的科学计数法形式：

   例如对于表达式 `a -  -2e3`：

   在我们的直觉下，我实现的的其中一版sysu-lexer将它识别为：`identifier  a`；`minus  -`；`numeric_constant  -2e3`

   但是!!!!!! clang会识别为：`identifier  a`；`minus  -`；`minus  -`；`numeric_constant  2e3` 一共四项。参照以下`comp_clang.txt`文件。

   ```
   workspace > SYsU-lang > lexer > ≡ comp_clang.txt
   1   identifier 'a'   [StartOfLine]  Loc=</workspace/SYsU-lang/lexer/test.sysu.c:1:1>
   2   minus '-'    [LeadingSpace] Loc=</workspace/SYsU-lang/lexer/test.sysu.c:1:3>
   3   minus '-'    [LeadingSpace] Loc=</workspace/SYsU-lang/lexer/test.sysu.c:1:5>
   4   numeric_constant '2e3'      Loc=</workspace/SYsU-lang/lexer/test.sysu.c:1:6>
   5   eof ''      Loc=</workspace/SYsU-lang/lexer/test.sysu.c:1:9>
   6
   ```

2. 关于负号`-`与减号`-`的分辨：[lex中减号与负号的分辨]， [正则表达式--断言]

   在实现过程中，加入了科学计数法的正则表达式后，出现了诸如将`1-2`识别为`numeric_constant  1`;`numeric_constant  -2`的情况，为了解决这个问题，我查阅了相关资料。

   在查询了许多资料后，我发现一般的词法分析器都是通过**正后顾断言**来分别减号与负号的，但是，我通过实践也证实了lex并不支持断言，无法识别加入了断言的正则表达式。

   解决方法：通过问题1， clang 无法识别负数的科学计数法形式， 将两者联系起来不难推断，**clang关于科学计数法的正则表达式是不包含符号的**，将所有的+-识别为加号减号即可。

3. 一点吐槽:

`-.33E+5`； `0x.AP-3`； `02.`

为什么会有人这么写浮点数？？？？？真是扭曲恶心肮脏的数字呢，知不知道鼠鼠我写正则表达式通式写得有多折磨？？！

通过此次实验，我也明白了平时的公式，科学计数法的规范书写是很重要的，词法分析器认不出来，或者认成一些奇奇怪怪的东西，就真的很烦。

- 问题三： 关于字符char与字符串string的正则表达式：

  肯定是要考虑转义字符的，字符串中出现了双引号`"`，转义符号`\`，这两个特殊字符，都需要拿出来专门讨论， 于是乎，在查阅了相关资料，以及尽量贴合本次实验的前提下，我写的正则表达式如下：

  字符串string：`\"[^"\\]*((\\[onrtvabf'"\\?])*[^"\\]*)*\"`

  字符char: `\'[^"\\]|(\\[onrtvabf'"\\?])\'`

- 问题四：关于无关字符的识别， 实现`lexer-3`。

  仔细检查`comp_clang.txt`文件，可以发现只需要实现两个无关字符的识别`[StartOfLine]`和`[LeadingSpace]`。

  分别表示此次识别的`yytext`是否是新的一行，以及其前方是否留有空格。

  为此，我在`lexer.l`中加入了两个全局变量`yynewline = 1`与`yyspace = 0`， 它们的逻辑是：

  - **每当识别到`\n`时， `yynewline`更新为1， 每当识别到其他需要打印出来的`yytext`，打印后将`yynewline`设置为0；**
  - **每当识别到空白格`\0`时， `yyspace`更新为1， 每当识别到其他需要打印出来的`yytext`，打印后将`yyspace`设置为0**；

- **每识别到需要打印的 `yytext`， 如果 `yynewline = 1`，则插入 `[StartOfLine]`；如果 `yyspace = 1`，则插入 `[LeadingSpace]`；**

这样一来，就实现了无关字符的识别。

3. 实验源码

```
%{
#include <cctype>
#include <cstdio>
#include <string>
#define YYEOF 0
int yylex();
int main() {
  do {
  } while (yylex() != YYEOF);
}
std::string yyloc = "<stdin>";
int yyrow = 1, yycolumn = 1, yycolpre = 1, yynewline = 1, yyspace = 0;
#define YY_USER_ACTION                                                  \
  do {                                                                  \
    yycolumn += yyleng;                                                 \
  } while (0);
%}
%option noyywrap
%%
#.* {
  std::string s(yytext);
  auto l = s.find("\""), r = s.rfind("\"");
  yyloc = s.substr(l + 1, r - l - 1);
  for (int i = 0; i < s.size(); ++i)
    if (std::isdigit(s[i])) {
      for (yyrow = 0; i < s.size() && std::isdigit(s[i]); ++i)
        yyrow = yyrow * 10 + s[i] - '0';
      --yyrow;
      break;
    }
}
\n {
  ++yyrow;
  yycolpre = yycolumn;
  yycolumn = 1;
  yynewline = 1;
}
[ ]+ {
  yyspace = 1;
}
[\f\r\t\v] {}
void {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
```

```
  std::fprintf(yyout, "void '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
const {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "const '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
char {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "char '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
long {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "long '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
int {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "int '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
float {
```

```
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "float '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
double {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "double '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
if {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "if '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
else {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "else '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
do {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "do '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
```

```
  return ~YYEOF;
}
while {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "while '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
break {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "break '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
continue {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "continue '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
return {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "return '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\( {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "l_paren '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
```

```
                yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\) {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "r_paren '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
                yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\{ {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "l_brace '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
                yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\} {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "r_brace '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
                yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\[ {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "l_square '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
                yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\] {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
```

```
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "r_square '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
, {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "comma '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
; {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "semi '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
= {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "equal '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
! {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "exclaim '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
```

```
"+" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "plus '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
- {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "minus '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"*" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "star '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"/" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "slash '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
% {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "percent '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
```

```
  yyspace = 0;
  return ~YYEOF;
}
"<" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "less '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
">" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "greater '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"<=" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "lessequal '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
">=" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "greaterequal '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
== {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
```

```
  std::fprintf(yyout, "equalequal '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
!= {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "exclaimequal '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"&&" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "ampamp '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"||" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "pipepipe '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
"..." {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "ellipsis '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
[0-9]+ {
```

```
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "numeric_constant '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
0[bB][01]+ {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "numeric_constant '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
0[xX][0-9a-fA-F]+ {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "numeric_constant '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
([0-9]*\.[0-9]+|[0-9]+(\.[0-9]*)?)([eE][-+]?[0-9]+)? {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "numeric_constant '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
0[xX]([0-9a-fA-F]*\.[0-9a-fA-F]+|[0-9a-fA-F]+(\.[0-9a-fA-F]*)?)([pP][-+]?[0-9]+)? {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "numeric_constant '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
[a-zA-Z_][a-zA-Z_0-9]* {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
```

```
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "identifier '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\"[^"\\]*((\\[onrtvabf'"\\?])*[^"\\]*)*\" {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "string_literal '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
\'[^"\\]|(\\[onrtvabf'"\\?])\' {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "char '%s'\t%sLoc=<%s:%d:%d>\n", yytext,
               s.c_str(), yyloc.c_str(), yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
<<EOF>> {
  std::fprintf(yyout, "eof ''\t\tLoc=<%s:%d:%d>\n", yyloc.c_str(), yyrow - 1,
               yycolpre - yyleng);
  return YYEOF;
}
. {
  std::string s = "";
  if(yynewline) s.append("[StartOfLine] ");
  if(yyspace) s.append("[LeadingSpace] ");
  std::fprintf(yyout, "unknown '%s'\t%sLoc=<%s:%d:%d>\n", yytext, s.c_str(),
yyloc.c_str(),
               yyrow, yycolumn - yyleng);
  yynewline = 0;
  yyspace = 0;
  return ~YYEOF;
}
%%
```

4. 实验结果

```
root@caae647447ae:/workspace/SYsU-lang# CTEST_OUTPUT_ON_FAILURE=1 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    0.45 sec
      Start  2: lexer-1
 2/10 Test  #2: lexer-1 ..............................   Passed   81.30 sec
      Start  3: lexer-2
 3/10 Test  #3: lexer-2 ..............................   Passed   79.05 sec
      Start  4: lexer-3
 4/10 Test  #4: lexer-3 ..............................   Passed   86.09 sec
      Start  5: parser-0
```

最终，评测项目`lexer-[0-3]`全部`Passed`。实验一圆满完成。