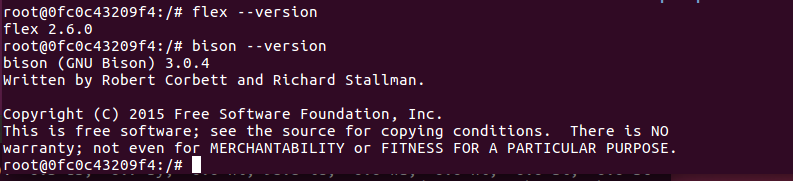
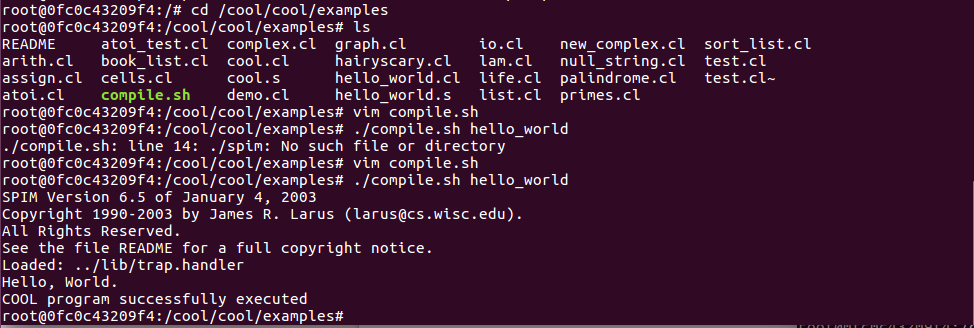
**第一次实验：**

环境配置：

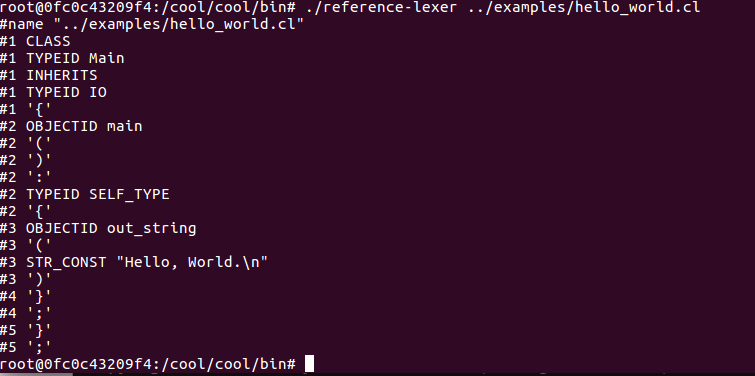
声明：本实验使用docker-Ubuntu，所以主机名称为root@xxxxxxx



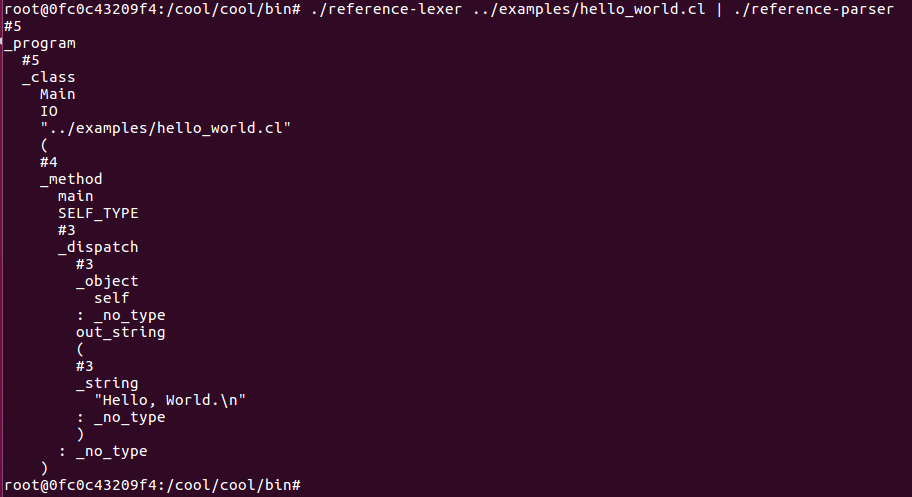
编译运行hello\_world：



词法分析：

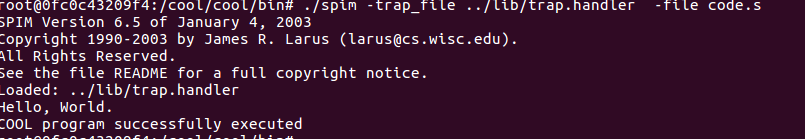


语法分析：



生成汇编代码：

15

17

**第二次实验：**

Stack.cl代码：

(\*

The class A2I provides integer-to-string and string-to-integer

conversion routines. To use these routines, either inherit them

in the class where needed, have a dummy variable bound to

something of type A2I, or simpl write (new A2I).method(argument).

\*)

(\*

c2i Converts a 1-character string to an integer. Aborts

if the string is not "0" through "9"

\*)

class A2I {

c2i(char : String) : Int {

if char = "0" then 0 else

if char = "1" then 1 else

if char = "2" then 2 else

if char = "3" then 3 else

if char = "4" then 4 else

if char = "5" then 5 else

if char = "6" then 6 else

if char = "7" then 7 else

if char = "8" then 8 else

if char = "9" then 9 else

{ abort(); 0; } -- the 0 is needed to satisfy the typchecker

fi fi fi fi fi fi fi fi fi fi

};

(\*

i2c is the inverse of c2i.

\*)

i2c(i : Int) : String {

if i = 0 then "0" else

if i = 1 then "1" else

if i = 2 then "2" else

if i = 3 then "3" else

if i = 4 then "4" else

if i = 5 then "5" else

if i = 6 then "6" else

if i = 7 then "7" else

if i = 8 then "8" else

if i = 9 then "9" else

{ abort(); ""; } -- the "" is needed to satisfy the typchecker

fi fi fi fi fi fi fi fi fi fi

};

(\*

a2i converts an ASCII string into an integer. The empty string

is converted to 0. Signed and unsigned strings are handled. The

method aborts if the string does not represent an integer. Very

long strings of digits produce strange answers because of arithmetic

overflow.

\*)

a2i(s : String) : Int {

if s.length() = 0 then 0 else

if s.substr(0,1) = "-" then ~a2i\_aux(s.substr(1,s.length()-1)) else

if s.substr(0,1) = "+" then a2i\_aux(s.substr(1,s.length()-1)) else

a2i\_aux(s)

fi fi fi

};

(\*

a2i\_aux converts the usigned portion of the string. As a programming

example, this method is written iteratively.

\*)

a2i\_aux(s : String) : Int {

(let int : Int <- 0 in

{

(let j : Int <- s.length() in

(let i : Int <- 0 in

while i < j loop

{

int <- int \* 10 + c2i(s.substr(i,1));

i <- i + 1;

}

pool

)

);

int;

}

)

};

(\*

i2a converts an integer to a string. Positive and negative

numbers are handled correctly.

\*)

i2a(i : Int) : String {

if i = 0 then "0" else

if 0 < i then i2a\_aux(i) else

"-".concat(i2a\_aux(i \* ~1))

fi fi

};

(\*

i2a\_aux is an example using recursion.

\*)

i2a\_aux(i : Int) : String {

if i = 0 then "" else

(let next : Int <- i / 10 in

i2a\_aux(next).concat(i2c(i - next \* 10))

)

fi

};

};

class List inherits IO

{

isNil() : Bool

{

{

--out\_string("list\n");

true;

}

};

head() : String

{

{

abort();

"";

}

};

tail() : List

{

{

abort();

self;

}

};

cons(i : String) : List

{

(new Cons).init(i, self)

};

};

class Cons inherits List

{

first : String;

rest : List;

isNil() : Bool

{

{

--out\_string("cons\n");

false;

}

};

head() : String

{

first

};

tail() : List

{

rest

};

init(head : String, next : List) : List

{

{

first <- head;

rest <- next;

self;

}

};

};

class Main inherits IO

{

stack : List;

newline() : Object

{

out\_string("\n")

};

prompt() : String

{

{

out\_string(">");

in\_string();

}

};

display\_stack(s : List) : Object

{

{

--out\_string("hello\n");

if s.isNil() then out\_string("")

else

{

out\_string(s.head());

out\_string("\n");

display\_stack(s.tail());

}

fi;

}

};

main():Object

{

( let z : A2I <- new A2I , stack : List <- new List in

while true loop

( let s : String <- prompt() in

if s = "x" then

abort()

else

if s = "d" then

display\_stack(stack)

else

if s = "e" then

{

if stack.isNil() then out\_string("")

else

if stack.head() = "+" then

{

stack <- stack.tail();

(let a : Int <- new Int, b : Int <- new Int in

{

--out\_string(stack.head());

a <- z.a2i(stack.head());

stack <- stack.tail();

b <- z.a2i(stack.head());

stack <- stack.tail();

a <- a + b;

--out\_string(z.i2a(a));

stack <- stack.cons(z.i2a(a));

}

);

}

else

if stack.head() = "s" then

{

stack <- stack.tail();

(let a : String <- new String , b : String <- new String in

{

a <- stack.head();

stack <- stack.tail();

b <- stack.head();

stack <- stack.tail();

stack <- stack.cons(a);

stack <- stack.cons(b);

}

);

}

else

out\_string("")

fi

fi

fi;

}

else

stack <- stack.cons(s)

fi

fi

fi

)

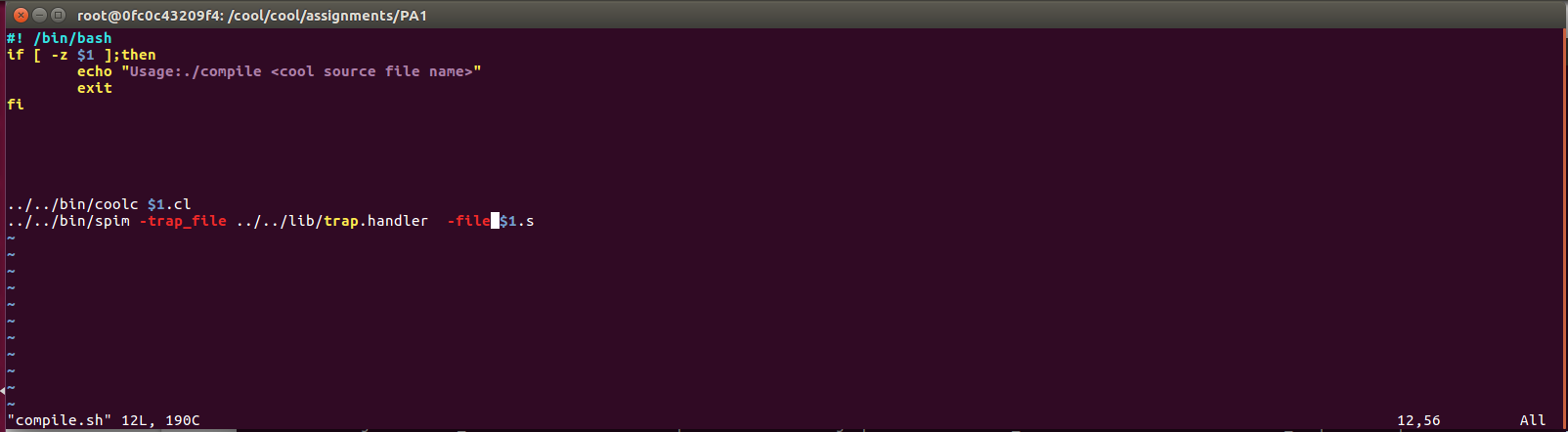
pool

)

};

};

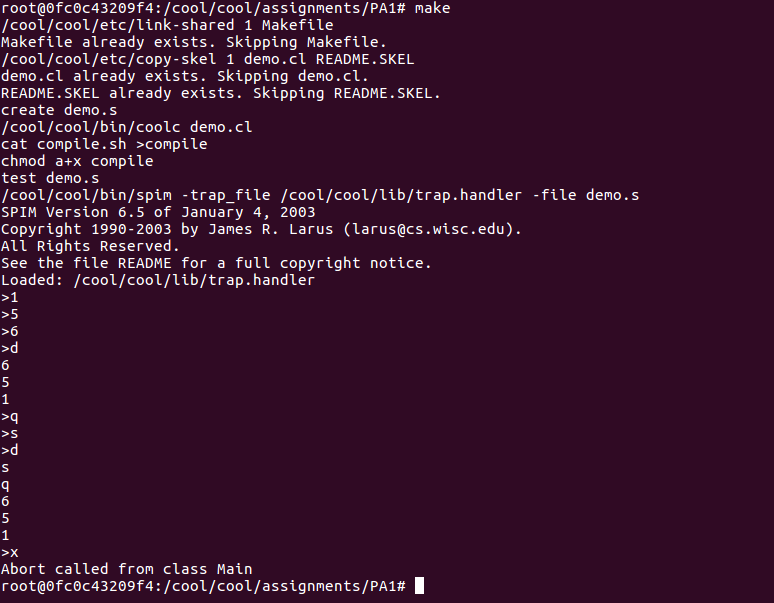
compile.sh实现自动化运行：



运行：

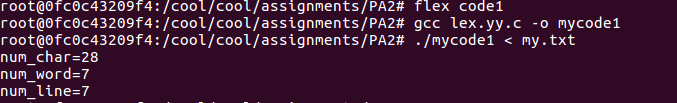
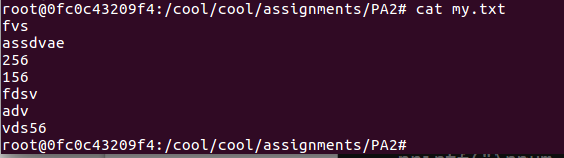
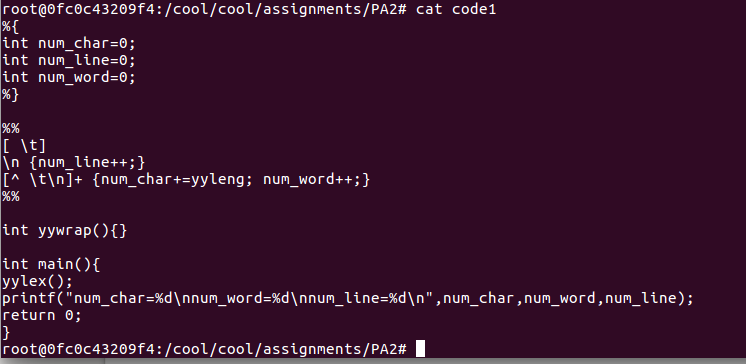


使用make运行：

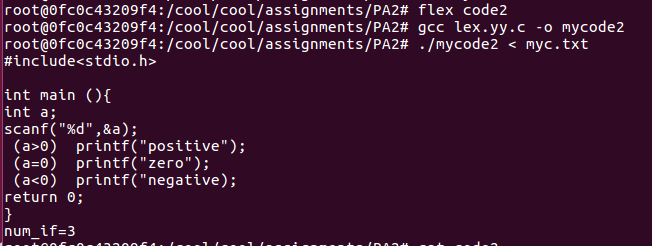
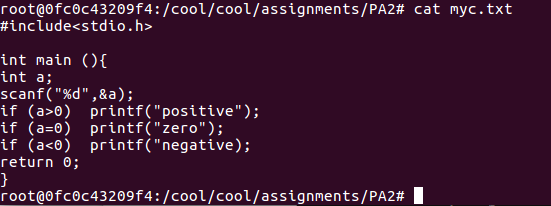
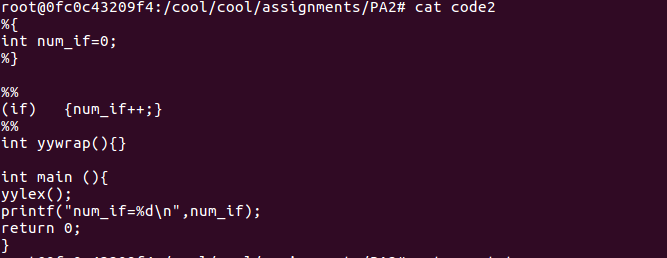


**第三次实验：**

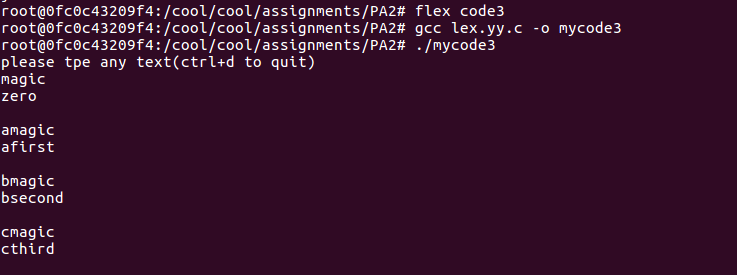
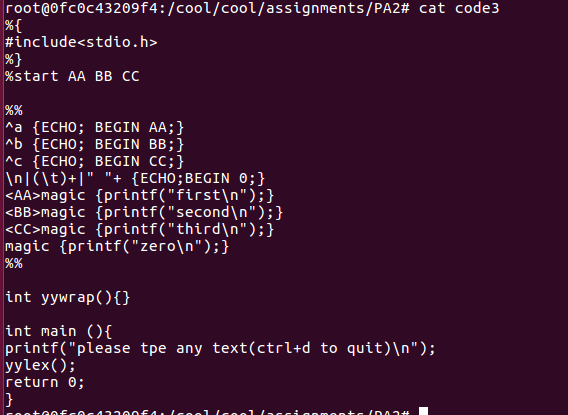
1. 统计字符个数，单词个数，行数（源代码）



1. 统计if语句



1. 源代码



**第四次实验：**

lexer.l文件：

%{

int uniqueIndex = 1;

int nline = 1;

%}

%s MutiCom

letter [A-Za-z\_]

digit [0-9]

identifier {letter}({letter}|{digit})\*

Int {digit}+

operator "-"|"+"|"="|"=="|"\*"|"/"|"<="|">="|"<"|">"|"<-"|"=>"

delimiters "("|")"|"{"|"}"|";"|","|"@"

keyword "class"|"public"|"fuction"|"if"|"else"|"then"|"return"|"let"|"in"|"while"|"loop"|"pool"|"case"|"esac"|"of"|"new"|"isvoid"|"true"|"false"|"not"|"le"|"Main"|"fi"|"Int"|"String"|"new"

float {digit}\*{\.{digit}+}?(e|E[+\-]?{digit}+)?

%%

"--"[^\n]\* {}

"(\*" {BEGIN(MutiCom);}

<MutiCom>[^\n\*]\* {}

<MutiCom>"\*"+[^\*)\n]\* {}

<MutiCom>\n {nline++;}

<MutiCom>"\*"+")" {BEGIN(INITIAL);}

{keyword} {

printf("\*\*\* Line %d: ", nline);

printf("<keyword> "); ECHO;

printf(" \*\*\*\n");

}

{identifier} {

printf("\*\*\* Line %d: ", nline);

printf("<identifier> "); ECHO;

printf("\*\*\*\n");

}

\".\*\" {

printf("\*\*\* Line %d: ", nline);

printf("<string> "); ECHO;

printf("\*\*\*\n");

}

{Int} {

printf("\*\*\* Line %d: ", nline);

printf("<Int> "); ECHO;

printf(" \*\*\*\n");

}

{operator} {

printf("\*\*\* Line %d:", nline);

printf("<operator> "); ECHO;

printf(" \*\*\*\n");

}

{delimiters} {

printf("\*\*\* Line %d :", nline);

printf("<delimiters> "); ECHO;

printf(" \*\*\*\n");

}

\n { nline ++; }

[ \t] { /\*do nothing\*/ }

%%

int main() {

yylex();

printf("\n lexical analysis finished!\n");

return 0;

}

int yywrap() {return 1;}

mytest.cl文件：

class Main inherits IO {

main(): SELF\_TYPE {

{

out\_string("This is a example\n");

let x:Int <- 5,y:Int in

{

out\_string("x=");

out\_int(x);

out\_string("\n");

y<-4

out\_string("y=");

out\_int(y);

out\_string("\n");

out\_int(x+y);

}

out\_string("\n");

}

};

};

test.txt文件:

\*\*\* Line 1: <keyword> class \*\*\*

\*\*\* Line 1: <keyword> Main \*\*\*

\*\*\* Line 1: <identifier> inherits\*\*\*

\*\*\* Line 1: <identifier> IO\*\*\*

\*\*\* Line 1 :<delimiters> { \*\*\*

\*\*\* Line 2: <identifier> main\*\*\*

\*\*\* Line 2 :<delimiters> ( \*\*\*

\*\*\* Line 2 :<delimiters> ) \*\*\*

:\*\*\* Line 2: <identifier> SELF\_TYPE\*\*\*

\*\*\* Line 2 :<delimiters> { \*\*\*

\*\*\* Line 3 :<delimiters> { \*\*\*

\*\*\* Line 4: <identifier> out\_string\*\*\*

\*\*\* Line 4 :<delimiters> ( \*\*\*

\*\*\* Line 4: <string> "This is a example\n"\*\*\*

\*\*\* Line 4 :<delimiters> ) \*\*\*

\*\*\* Line 4 :<delimiters> ; \*\*\*

\*\*\* Line 5: <keyword> let \*\*\*

\*\*\* Line 5: <identifier> x\*\*\*

:\*\*\* Line 5: <keyword> Int \*\*\*

\*\*\* Line 5:<operator> <- \*\*\*

\*\*\* Line 5: <Int> 5 \*\*\*

\*\*\* Line 5 :<delimiters> , \*\*\*

\*\*\* Line 5: <identifier> y\*\*\*

:\*\*\* Line 5: <keyword> Int \*\*\*

\*\*\* Line 5: <keyword> in \*\*\*

\*\*\* Line 6 :<delimiters> { \*\*\*

\*\*\* Line 7: <identifier> out\_string\*\*\*

\*\*\* Line 7 :<delimiters> ( \*\*\*

\*\*\* Line 7: <string> "x="\*\*\*

\*\*\* Line 7 :<delimiters> ) \*\*\*

\*\*\* Line 7 :<delimiters> ; \*\*\*

\*\*\* Line 8: <identifier> out\_int\*\*\*

\*\*\* Line 8 :<delimiters> ( \*\*\*

\*\*\* Line 8: <identifier> x\*\*\*

\*\*\* Line 8 :<delimiters> ) \*\*\*

\*\*\* Line 8 :<delimiters> ; \*\*\*

\*\*\* Line 9: <identifier> out\_string\*\*\*

\*\*\* Line 9 :<delimiters> ( \*\*\*

\*\*\* Line 9: <string> "\n"\*\*\*

\*\*\* Line 9 :<delimiters> ) \*\*\*

\*\*\* Line 9 :<delimiters> ; \*\*\*

\*\*\* Line 10: <identifier> y\*\*\*

\*\*\* Line 10:<operator> <- \*\*\*

\*\*\* Line 10: <Int> 4 \*\*\*

\*\*\* Line 11: <identifier> out\_string\*\*\*

\*\*\* Line 11 :<delimiters> ( \*\*\*

\*\*\* Line 11: <string> "y="\*\*\*

\*\*\* Line 11 :<delimiters> ) \*\*\*

\*\*\* Line 11 :<delimiters> ; \*\*\*

\*\*\* Line 12: <identifier> out\_int\*\*\*

\*\*\* Line 12 :<delimiters> ( \*\*\*

\*\*\* Line 12: <identifier> y\*\*\*

\*\*\* Line 12 :<delimiters> ) \*\*\*

\*\*\* Line 12 :<delimiters> ; \*\*\*

\*\*\* Line 13: <identifier> out\_string\*\*\*

\*\*\* Line 13 :<delimiters> ( \*\*\*

\*\*\* Line 13: <string> "\n"\*\*\*

\*\*\* Line 13 :<delimiters> ) \*\*\*

\*\*\* Line 13 :<delimiters> ; \*\*\*

\*\*\* Line 14: <identifier> out\_int\*\*\*

\*\*\* Line 14 :<delimiters> ( \*\*\*

\*\*\* Line 14: <identifier> x\*\*\*

\*\*\* Line 14:<operator> + \*\*\*

\*\*\* Line 14: <identifier> y\*\*\*

\*\*\* Line 14 :<delimiters> ) \*\*\*

\*\*\* Line 14 :<delimiters> ; \*\*\*

\*\*\* Line 15 :<delimiters> } \*\*\*

\*\*\* Line 16: <identifier> out\_string\*\*\*

\*\*\* Line 16 :<delimiters> ( \*\*\*

\*\*\* Line 16: <string> "\n"\*\*\*

\*\*\* Line 16 :<delimiters> ) \*\*\*

\*\*\* Line 16 :<delimiters> ; \*\*\*

\*\*\* Line 17 :<delimiters> } \*\*\*

\*\*\* Line 18 :<delimiters> } \*\*\*

\*\*\* Line 18 :<delimiters> ; \*\*\*

\*\*\* Line 19 :<delimiters> } \*\*\*

\*\*\* Line 19 :<delimiters> ; \*\*\*

lexical analysis finished!

test\_built-in.txt文档

#name "test.cl"

#5 CLASS

#5 TYPEID CellularAutomaton

#5 INHERITS

#5 TYPEID IO

#5 '{'

#6 OBJECTID population\_map

#6 ':'

#6 TYPEID String

#6 ';'

#8 OBJECTID init

#8 '('

#8 OBJECTID map

#8 ':'

#8 TYPEID String

#8 ')'

#8 ':'

#8 TYPEID SELF\_TYPE

#8 '{'

#9 '{'

#10 OBJECTID population\_map

#10 ASSIGN

#10 OBJECTID map

#10 ';'

#11 OBJECTID self

#11 ';'

#12 '}'

#13 '}'

#13 ';'

#15 OBJECTID print

#15 '('

#15 ')'

#15 ':'

#15 TYPEID SELF\_TYPE

#15 '{'

#16 '{'

#17 OBJECTID out\_string

#17 '('

#17 OBJECTID population\_map

#17 '.'

#17 OBJECTID concat

#17 '('

#17 STR\_CONST "\n"

#17 ')'

#17 ')'

#17 ';'

#18 OBJECTID self

#18 ';'

#19 '}'

#20 '}'

#20 ';'

#22 OBJECTID num\_cells

#22 '('

#22 ')'

#22 ':'

#22 TYPEID Int

#22 '{'

#23 OBJECTID population\_map

#23 '.'

#23 OBJECTID length

#23 '('

#23 ')'

#24 '}'

#24 ';'

#26 OBJECTID cell

#26 '('

#26 OBJECTID position

#26 ':'

#26 TYPEID Int

#26 ')'

#26 ':'

#26 TYPEID String

#26 '{'

#27 OBJECTID population\_map

#27 '.'

#27 OBJECTID substr

#27 '('

#27 OBJECTID position

#27 ','

#27 INT\_CONST 1

#27 ')'

#28 '}'

#28 ';'

#30 OBJECTID cell\_left\_neighbor

#30 '('

#30 OBJECTID position

#30 ':'

#30 TYPEID Int

#30 ')'

#30 ':'

#30 TYPEID String

#30 '{'

#31 IF

#31 OBJECTID position

#31 '='

#31 INT\_CONST 0

#31 THEN

#32 OBJECTID cell

#32 '('

#32 OBJECTID num\_cells

#32 '('

#32 ')'

#32 '-'

#32 INT\_CONST 1

#32 ')'

#33 ELSE

#34 OBJECTID cell

#34 '('

#34 OBJECTID position

#34 '-'

#34 INT\_CONST 1

#34 ')'

#35 FI

#36 '}'

#36 ';'

#38 OBJECTID cell\_right\_neighbor

#38 '('

#38 OBJECTID position

#38 ':'

#38 TYPEID Int

#38 ')'

#38 ':'

#38 TYPEID String

#38 '{'

#39 IF

#39 OBJECTID position

#39 '='

#39 OBJECTID num\_cells

#39 '('

#39 ')'

#39 '-'

#39 INT\_CONST 1

#39 THEN

#40 OBJECTID cell

#40 '('

#40 INT\_CONST 0

#40 ')'

#41 ELSE

#42 OBJECTID cell

#42 '('

#42 OBJECTID position

#42 '+'

#42 INT\_CONST 1

#42 ')'

#43 FI

#44 '}'

#44 ';'

#48 OBJECTID cell\_at\_next\_evolution

#48 '('

#48 OBJECTID position

#48 ':'

#48 TYPEID Int

#48 ')'

#48 ':'

#48 TYPEID String

#48 '{'

#49 IF

#49 '('

#49 IF

#49 OBJECTID cell

#49 '('

#49 OBJECTID position

#49 ')'

#49 '='

#49 STR\_CONST "X"

#49 THEN

#49 INT\_CONST 1

#49 ELSE

#49 INT\_CONST 0

#49 FI

#50 '+'

#50 IF

#50 OBJECTID cell\_left\_neighbor

#50 '('

#50 OBJECTID position

#50 ')'

#50 '='

#50 STR\_CONST "X"

#50 THEN

#50 INT\_CONST 1

#50 ELSE

#50 INT\_CONST 0

#50 FI

#51 '+'

#51 IF

#51 OBJECTID cell\_right\_neighbor

#51 '('

#51 OBJECTID position

#51 ')'

#51 '='

#51 STR\_CONST "X"

#51 THEN

#51 INT\_CONST 1

#51 ELSE

#51 INT\_CONST 0

#51 FI

#52 '='

#52 INT\_CONST 1

#52 ')'

#53 THEN

#54 STR\_CONST "X"

#55 ELSE

#56 ERROR "'"

#56 '.'

#56 ERROR "'"

#57 FI

#58 '}'

#58 ';'

#60 OBJECTID evolve

#60 '('

#60 ')'

#60 ':'

#60 TYPEID SELF\_TYPE

#60 '{'

#61 '('

#61 LET

#61 OBJECTID position

#61 ':'

#61 TYPEID Int

#61 IN

#62 '('

#62 LET

#62 OBJECTID num

#62 ':'

#62 TYPEID Int

#62 ASSIGN

#62 OBJECTID num\_cells

#62 ERROR "["

#62 ERROR "]"

#62 IN

#63 '('

#63 LET

#63 OBJECTID temp

#63 ':'

#63 TYPEID String

#63 IN

#64 '{'

#65 WHILE

#65 OBJECTID position

#65 '<'

#65 OBJECTID num

#65 LOOP

#66 '{'

#67 OBJECTID temp

#67 ASSIGN

#67 OBJECTID temp

#67 '.'

#67 OBJECTID concat

#67 '('

#67 OBJECTID cell\_at\_next\_evolution

#67 '('

#67 OBJECTID position

#67 ')'

#67 ')'

#67 ';'

#68 OBJECTID position

#68 ASSIGN

#68 OBJECTID position

#68 '+'

#68 INT\_CONST 1

#68 ';'

#69 '}'

#70 POOL

#70 ';'

#71 OBJECTID population\_map

#71 ASSIGN

#71 OBJECTID temp

#71 ';'

#72 OBJECTID self

#72 ';'

#73 '}'

#74 ')'

#74 ')'

#74 ')'

#75 '}'

#75 ';'

#76 '}'

#76 ';'

#78 CLASS

#78 TYPEID Main

#78 '{'

#79 OBJECTID cells

#79 ':'

#79 TYPEID CellularAutomaton

#79 ';'

#81 OBJECTID main

#81 '('

#81 ')'

#81 ':'

#81 TYPEID SELF\_TYPE

#81 '{'

#82 '{'

#83 OBJECTID cells

#83 ASSIGN

#83 '('

#83 NEW

#83 TYPEID CellularAutomaton

#83 ')'

#83 '.'

#83 OBJECTID init

#83 '('

#83 STR\_CONST " X "

#83 ')'

#83 ';'

#84 OBJECTID cells

#84 '.'

#84 OBJECTID print

#84 '('

#84 ')'

#84 ';'

#85 '('

#85 LET

#85 OBJECTID countdown

#85 ':'

#85 TYPEID Int

#85 ASSIGN

#85 INT\_CONST 20

#85 IN

#86 WHILE

#86 OBJECTID countdown

#86 ERROR ">"

#86 INT\_CONST 0

#86 LOOP

#87 '{'

#88 OBJECTID cells

#88 '.'

#88 OBJECTID evolve

#88 '('

#88 ')'

#88 ';'

#89 OBJECTID cells

#89 '.'

#89 OBJECTID print

#89 '('

#89 ')'

#89 ';'

#90 OBJECTID countdown

#90 ASSIGN

#90 OBJECTID countdown

#90 '-'

#90 INT\_CONST 1

#90 ';'

#92 POOL

#93 ')'

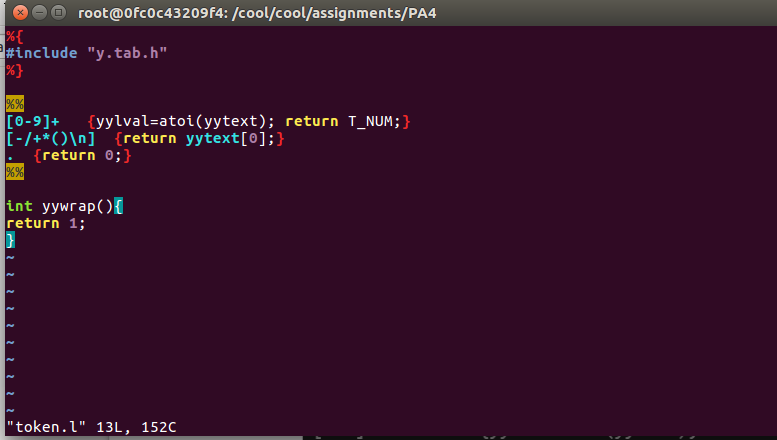
#93 ';'

#98 ERROR "EOF in comment"

**第五次实验：**

四则运算表达式：

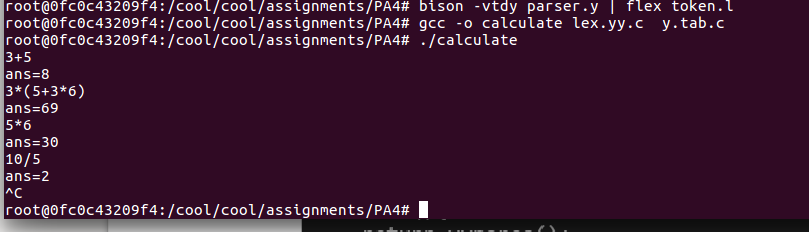
token.l文件：



parser.y文件：

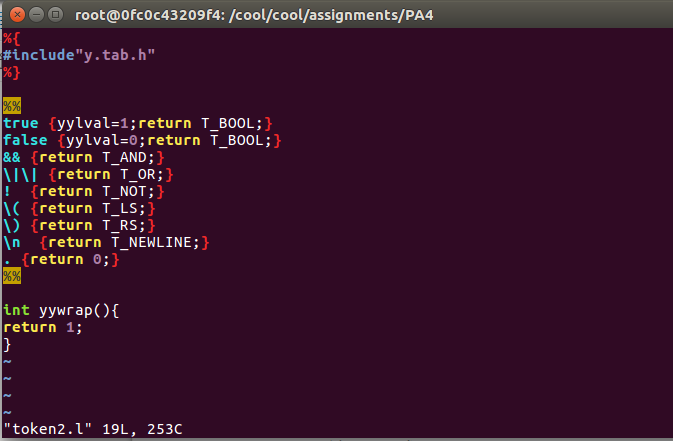


实际运行演示：

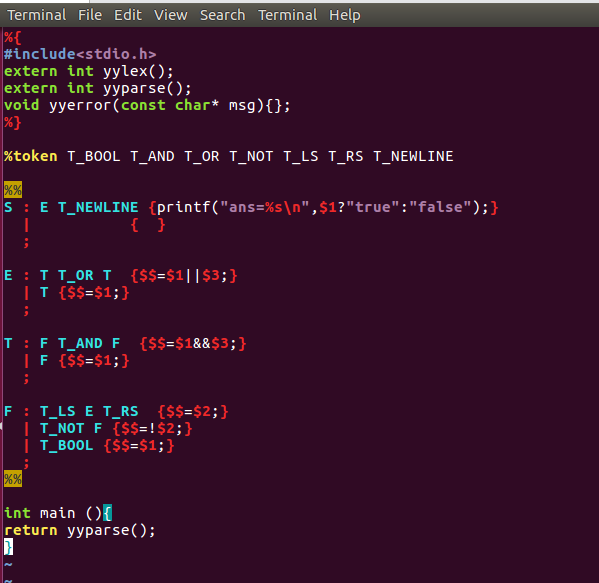


布尔表达式：

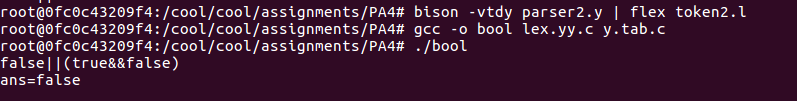
token2.l文件：



parser2.y文件：



实际运行演示：



**第六次实验：**