**COMPILER EXP1 REPORT**

**1 The Cool Language**

The language chosen for constructing our lexer is named Classroom Object Oriented Language (Cool). Cool is a language designed to be implementable in a short time, which is created by Alex Aiken, the Professor of Computer Science from Stanford. Using this language, we can design and implement our own compiler in a shorter time and at the same time wouldn’t lose learning value of compilers. Still, Cool retains many of the features of modern programming languages including objects, static typing, and automatic memory management.

Our final goal is to make a complete compiler that can compile Cool to MIPS assembly language. Yet, in this experiment, we will focus on lexical analysis and try to implement a lexical analyzer.

**2 Lexical Structure of Cool**

The lexical units of Cool are integers, type identifiers, object identifiers, special notation, strings, keywords, and white space.

**2.1 Integers, Identifiers, and Special Notation**

Integers are non-empty strings of digits 0-9.

Identifiers are strings (other than keywords) consisting of letters, digits, and the underscore character.

Type identifiers begin with a capital letter;

object identifiers begin with a lower case letter.

There are two other identifiers, self and SELF TYPE that are treated specially by Cool but are not treated as keywords. The special syntactic symbols (e.g., parentheses, assignment operator, etc.) are given in Figure 1.

**10.2 Strings**

Strings are enclosed in double quotes "...". Within a string, a sequence ‘\c’ denotes the character ‘c’,

with the exception of the following:

\b backspace

\t tab

\n newline

\f formfeed

A non-escaped newline character may not appear in a string:

"This \

is OK"

"This is not

OK"

A string may not contain EOF. A string may not contain the null (character \0). Any other character may be included in a string. Strings cannot cross file boundaries.

**10.3 Comments**

There are two forms of comments in Cool. Any characters between two dashes “--” and the next newline (or EOF, if there is no next newline) are treated as comments. Comments may also be written by enclosing text in (∗ . . . ∗). The latter form of comment may be nested. Comments cannot cross file boundaries.

**10.4 Keywords**

The keywords of cool are: **class, else, false, fi, if, in, inherits, isvoid, let, loop, pool, then, while, case, esac, new, of, not, true.** Except for the constants **true** and **false**, keywords are case insensitive. To conform to the rules for other objects, the first letter of true and false must be lowercase; the trailing

letters may be upper or lower case.

**10.5 White Space**

White space consists of any sequence of the characters: blank (ascii 32), \n (newline, ascii 10), \f (form feed, ascii 12), \r (carriage return, ascii 13), \t (tab, ascii 9), \v (vertical tab, ascii 11).

**3 Experiment Environment Building**

Due to the complexity of getting all the needed tools installed correctly, a pre-configured Linux system via VirtualBox VM is provided.

VirtualBox allows us to install the VM on our own computer.

**3.1 VirtualBox VM Setup**

**Getting VirtualBox**

* First, download Oracle's VirtualBox, either from the VirtualBox website at <https://www.virtualbox.org/wiki/Downloads>, or from Oracle's download page at <http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html>. Choose the version for the operating system you are running on your computer.
* Once the download completes, run the installer to install VirtualBox. You will need administrator access on your computer to do so.

**Getting our VM image**

* Download our linux virtual machine image [here](https://s3-us-west-1.amazonaws.com/prod-edx/Compilers/VM/compilers-vm.zip), or if you have a BitTorrent client installed, you can use the torrent [here](http://spark-university.s3.amazonaws.com/stanford-compilers/vm/compilers-vm.zip?torrent). Note that this download is approximately 750MB.
* Unzip the file into a convenient directory. The unzipped files are about 2GB, so make sure you have enough disk space available. If you are a Windows XP user and have trouble opening the zip file, try WinZip (shareware) or 7-Zip (free, open source) instead of the built-in zip support in Windows.
* Once you have unzipped the VM, double-click on the file "Compilers.vbox". This should open the VM in VirtualBox.

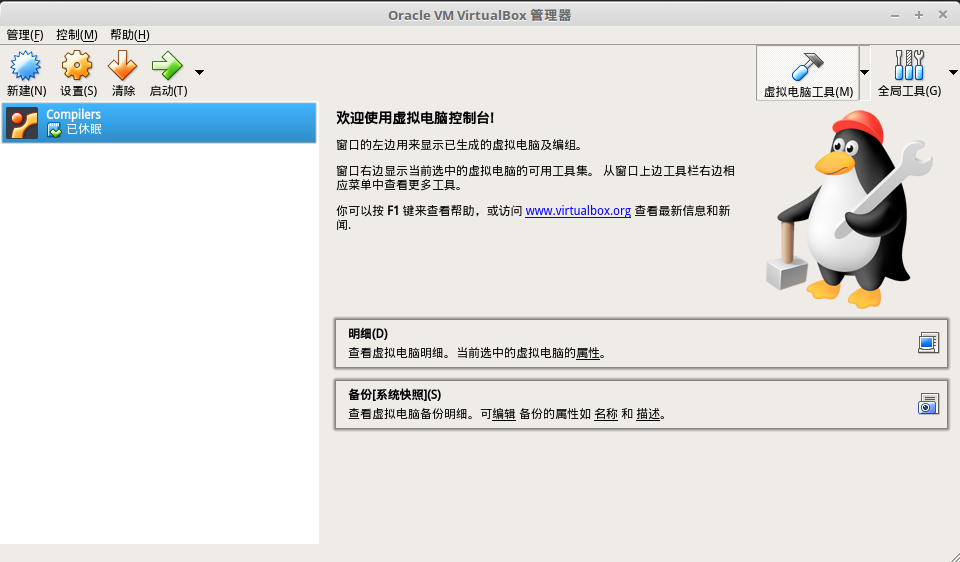
**Using the VM**

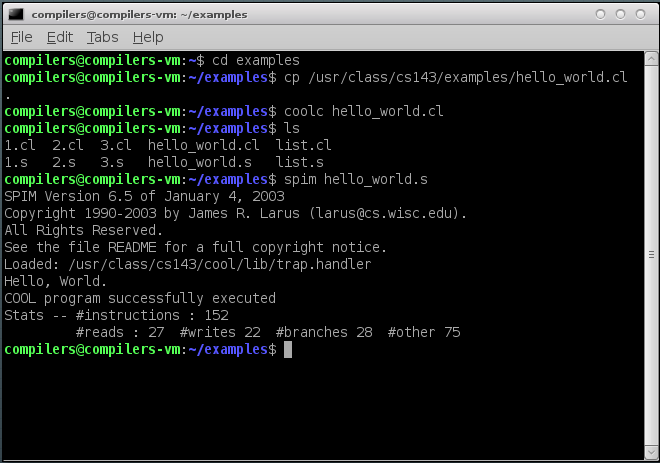
* The provided account is "compilers" and the password is "cool".
* To start the VM, click the green "Start" button. This should make the VM boot.
* To shut down the VM, click on the round "Bodhi" button at the bottom left and click System. Then choose Power Off.
* We have installed what you need to do the assignments, plus a few other programs, such as vim and emacs. If you want to install other packages, you can use the Aptitude graphical package manager (under the Bodhi menu->Applications->Preferences) or the apt-get command line tool. If you are not familiar with these, there are many tutorials online that you can find through a quick Google search.
* To get a terminal, click on the terminal icon at the bottom of the screen. This should get you to the point where you can start the assignments.

This VM is based on Bodhi Linux, which is itself based on the popular linux distribution Ubuntu. We used Bodhi Linux in order to keep the download size more manageable, since a full Ubuntu install is quite large. However, since it is based on Ubuntu, most Ubuntu software packages can be installed on Bodhi Linux as well.

Once the VM is set up, we can compile the example programs.

To play around with the example cool programs, make a directory and copy over one or more examples from the /usr/class/cs143/examples directory. These examples are the same as the ones posted on the website. The coolc command will run the reference compiler to generate the assembly output (.s) file, which you can run using the spim simulator. For example, to compile and run "hello\_world.cl", run in a terminal (where $ represents the prompt):





We can see the hello\_world.cl is compiled successfully and then be executed by SPIM, a [MIPS Simulator](http://spimsimulator.sourceforge.net/).

Here is the code of hello\_world.cl:

class Main inherits IO {

main(): SELF\_TYPE {

out\_string("Hello, World.\n")

};

};

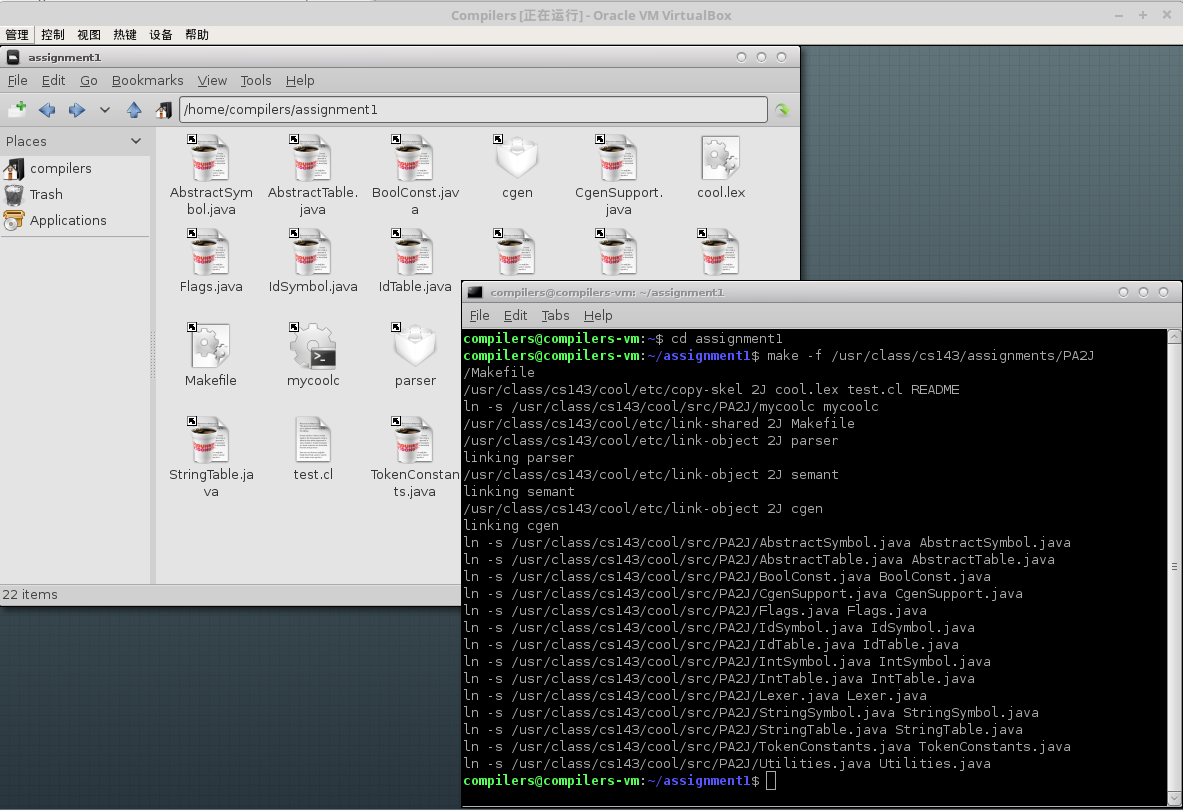
**3.2 Begin Our Experiment**

For this experiment, we are to write a lexical analyzer, also called a scanner, using a lexical analyzer generator. (The C++ tool is called flex; the Java tool is called jlex.) We will describe the set of tokens for Cool in an appropriate input format, and the analyzer generator will generate the actual code for recognizing tokens in Cool programs.

We choose jlex, the Java version of lex.

To get started, we created a directory where I want to do this experiment and executed the following command in that directory:

make -f /usr/class/cs143/assignments/PA2J/Makefile



Then I am going to modify some files to implement our lexer:

• cool.lex

This file contains a skeleton for a lexical description for Cool.

• test.cl

This file contains some sample input to be scanned. It does not exercise all of the lexical specification, but it is nevertheless an interesting test.

**4 Experiment Content**

**4.1 Introduction to Jlex**

Jlex allows us to implement a lexical analyzer by writing rules that match on user-defined regular expressions and performing a specified action for each matched pattern. Jlex compiles our rule file to Java source code implementing a finite automaton recognizing the regular expressions that we specify in our rule file.

Rule files in flex are structured as follows:

%{

Declarations

%}

Definitions

%%

Rules

%%

User subroutines

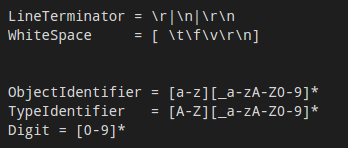
**4.2 The Implementation of our lexer**

First of all, according to the lexical structure of Cool, we divide the lexer into three states: BLOCK\_COMMENT, lINE\_COMMENT, STRING. So we declare it in Jlex:

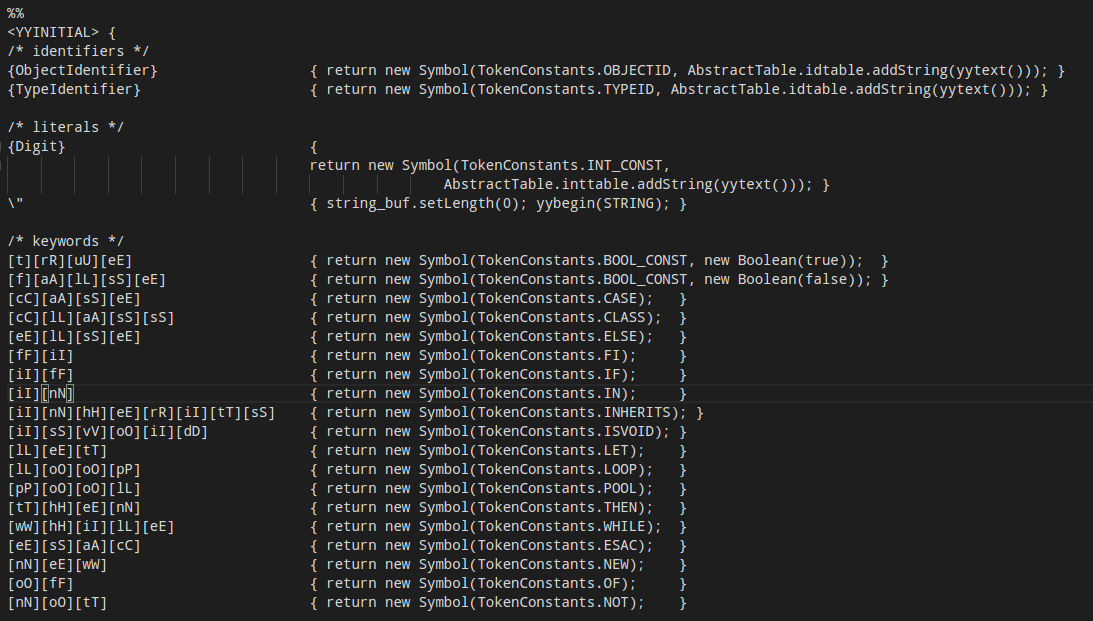


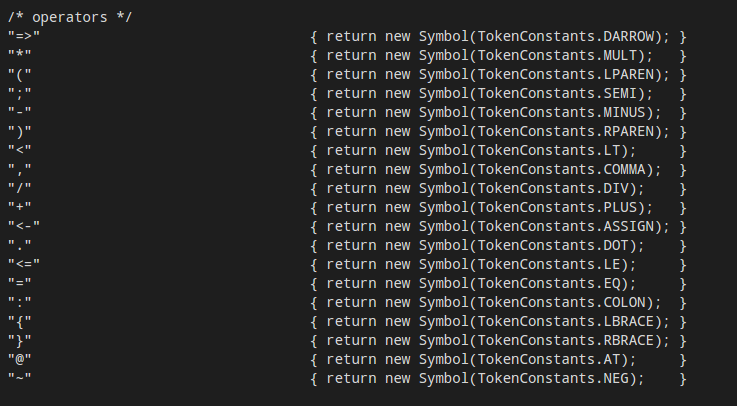
We declare an int variable **comment\_depth** and initiate it to zero account for the possibility of being nested of block comments.

For the sake of simplicity, we define names for some regular expressions as following:

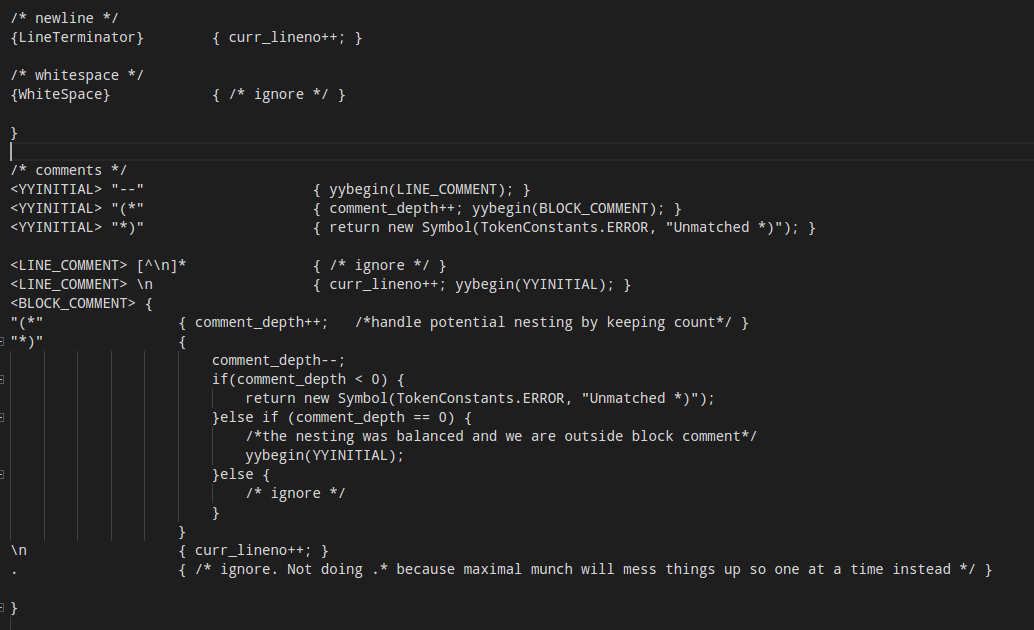


Then, we process those keywords and operators because they are the easiest. The keywords of cool are: **class, else, false, fi, if, in, inherits, isvoid, let, loop, pool, then, while, case, esac, new, of, not, true**. Except for the constants true and false, keywords are case insensitive. To conform to the rules for other objects, the first letter of true and false must be lowercase; the trailing letters may be upper or lower case.

There is the rules and relative actions:

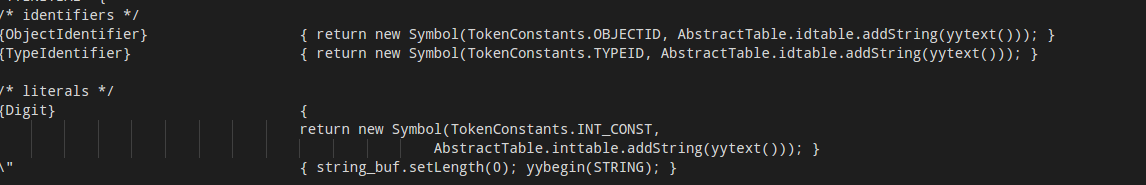


Next we set out to handle the newline, some whitespace, and the comments.

Once the lexer is in LINE\_COMMENT state, it will ignore all character until the newline character. When it encounters a newline, it will get into the initial state.

Notice that the block comments could be nested, thus a counter for the depth of comments is needed.

And the identifiers and literals:

Notice that when the lexer encounter the symbol “, it will change into STRING state and set the length of the string buffer to 0.

Finally. We are going to handle the most difficult but interesting token-strings.

As the definition of Cool language, Strings are enclosed in double quotes "...". Within a string, a sequence ‘\c’ denotes the character ‘c’,

with the exception of the following:

\b

\t

\n

\f

backspace

tab

newline

formfeed

A non-escaped newline character may not appear in a string:

"This \

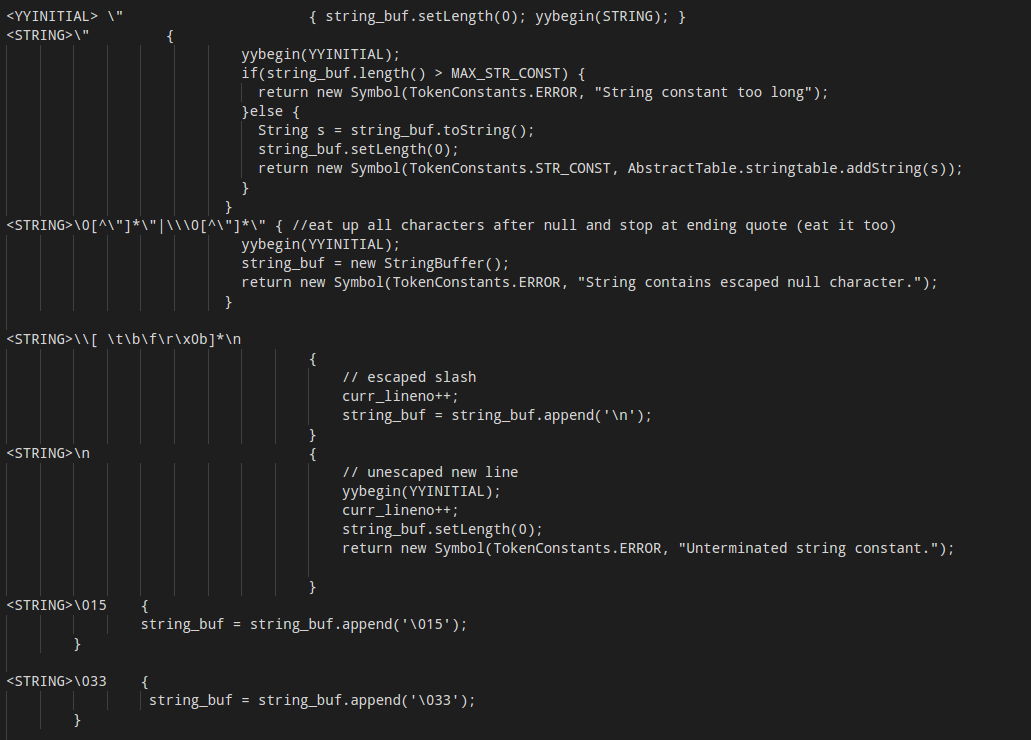
is OK"

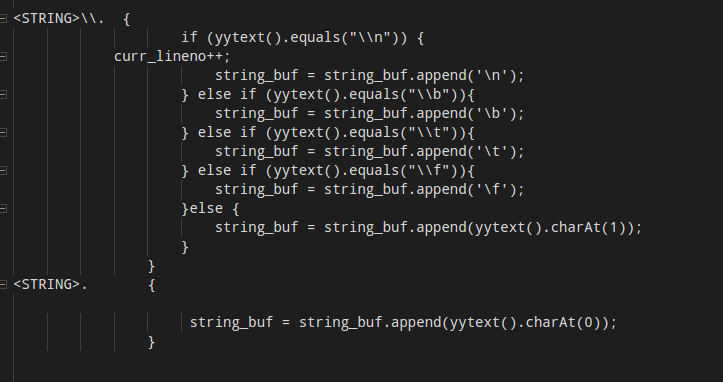
"This is not

OK"

A string may not contain EOF. A string may not contain the null (character \0). Any other character

may be included in a string. Strings cannot cross file boundaries.



Let me explain the string rules and actions in detail.

Firstly, when our lexer is in the initial state and it meet a “\”” character, it will set the string buffer’s length to zero and change to the STRING state.

Then in the STRING state, when we encounter another “\””, it means that the string should end, and we thus change the state to YYINITIAL, and check whether the length of this string is longer than its limit or not. If not, then return a STR\_CONST token and store the value of this string into our string table.

When we encounter the null character, our lexer will ignore all characters after null and stop at ending quote.

And when we encounter a ‘\’, following any whitespace and a ‘\n’ at last, which means that a part of that string start at the next line. To make you understand, there is a simple example:

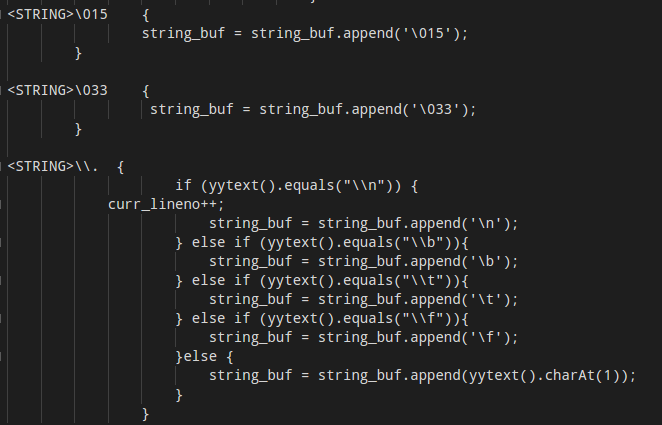
"This \

is OK"

"This is not

OK"

Then, when we encounter a carriage or escape or some other escape character, we append the relative characters.

****

**5 Testing Result**

Firstly, I gave my lexer a complicate Cool program (which contain some bugs deliberately) whose duty is do some mathematical job.

Here is the code:

(\* models one-dimensional cellular automaton on a circle of finite radius

arrays are faked as Strings,

X's respresent live cells, dots represent dead cells,

no error checking is done \*)

class CellularAutomaton inherits IO {

population\_map : String;

init(map : String) : SELF\_TYPE {

{

population\_map <- map;

self;

}

};

print() : SELF\_TYPE {

{

out\_string(population\_map.concat("\n"));

self;

}

};

num\_cells() : Int {

population\_map.length()

};

cell(position : Int) : String {

population\_map.substr(position, 1)

};

cell\_left\_neighbor(position : Int) : String {

if position = 0 then

cell(num\_cells() - 1)

else

cell(position - 1)

fi

};

cell\_right\_neighbor(position : Int) : String {

if position = num\_cells() - 1 then

cell(0)

else

cell(position + 1)

fi

};

(\* a cell will live if exactly 1 of itself and it's immediate

neighbors are alive \*)

cell\_at\_next\_evolution(position : Int) : String {

if (if cell(position) = "X" then 1 else 0 fi

+ if cell\_left\_neighbor(position) = "X" then 1 else 0 fi

+ if cell\_right\_neighbor(position) = "X" then 1 else 0 fi

= 1)

then

"X"

else

'.'

fi

};

evolve() : SELF\_TYPE {

(let position : Int in

(let num : Int <- num\_cells[] in

(let temp : String in

{

while position < num loop

{

temp <- temp.concat(cell\_at\_next\_evolution(position));

position <- position + 1;

}

pool;

population\_map <- temp;

self;

}

) ) )

};

};

class Main {

cells : CellularAutomaton;

main() : SELF\_TYPE {

{

cells <- (new CellularAutomaton).init(" X ");

cells.print();

(let countdown : Int <- 20 in

while countdown > 0 loop

{

cells.evolve();

cells.print();

countdown <- countdown - 1;

pool

); (\* end let countdown

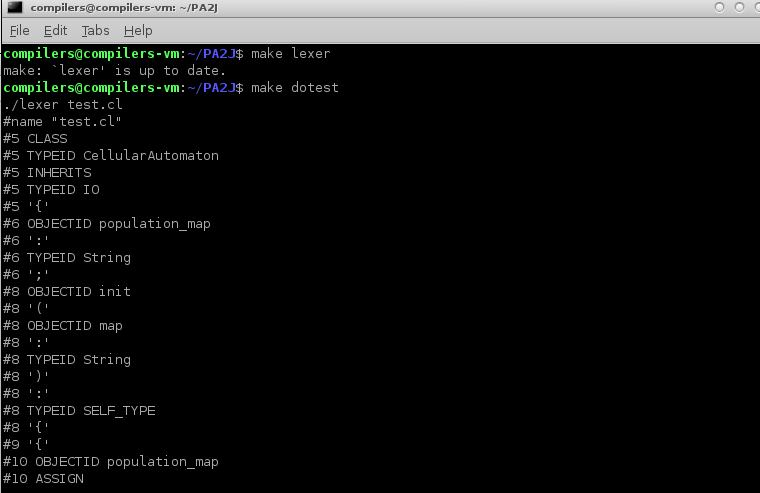
self;

}

};

};

And here is the result after scanning:



#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

LEXER BUG - UNMATCHED: '

#56 ERROR "'"

#56 '.'

LEXER BUG - UNMATCHED: '

#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

LEXER BUG - UNMATCHED: [

#62 ERROR "["

LEXER BUG - UNMATCHED: ]

#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

LEXER BUG - UNMATCHED: >

#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 ';'

#97 ERROR "EOF encontered in block comment."

Notice that there are some error report:

LEXER BUG - UNMATCHED: '

#56 ERROR "'"

#56 '.'

LEXER BUG - UNMATCHED: '

#56 ERROR "'"

This is because that it is illegal for single quote strings.

LEXER BUG - UNMATCHED: [

#62 ERROR "["

LEXER BUG - UNMATCHED: ]

#62 ERROR "]"

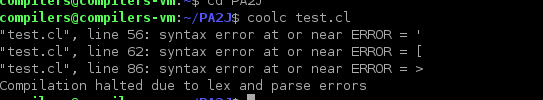
“[“ and “]” is also invalid in Cool

LEXER BUG - UNMATCHED: >

#86 ERROR ">"

Cool only support <.

We can use the official cool compiler to confirm that my lexer reported the expected errors.



However, one program cannot confirm that our lexer is right. Therefore, we use lots of test cases to test our lexer to find any faultiness. There are some test cases:

backslash.cool

"\0\1\2\\0\\1\\2\t\f\b\n\r\a\b\c\d\e\f\g"

"\"this is still inside the string"

"I'm about to excape a newline \

"

backslash.cool.out

#name "backslash.cool"

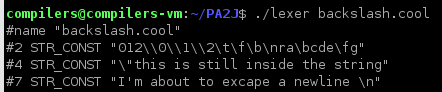
#1 STR\_CONST "012\\0\\1\\2\t\f\b\nra\bcde\fg"

#3 STR\_CONST "\"this is still inside the string"

#6 STR\_CONST "I'm about to excape a newline \n"

***测试完全符合***

*First I found that the line number is wrong*

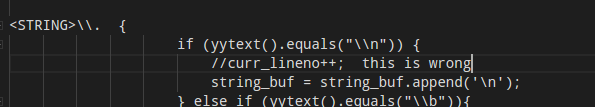
**

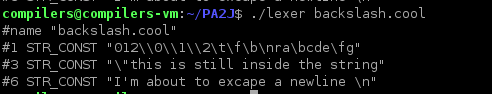
I suddenly realized that if we find \\n in a string, there is no need to increment the line number.

For example, “Hello\n” is just in one line in the text although the output is:

Hello

Thus I modified the rule:

Then the test result is the same as expected.



badidentifiers.cool

inky.binky.winky.dot

1thatwasanumber

two-objects

Two-Types

\_underscore

Type-object

badidentifiers.cool

#name "badidentifiers.cool"

#1 OBJECTID inky

#1 '.'

#1 OBJECTID binky

#1 '.'

#1 OBJECTID winky

#1 '.'

#1 OBJECTID dot

#2 INT\_CONST 1

#2 OBJECTID thatwasanumber

#3 OBJECTID two

#3 '-'

#3 OBJECTID objects

#4 TYPEID Two

#4 '-'

#4 TYPEID Types

#5 ERROR "\_"

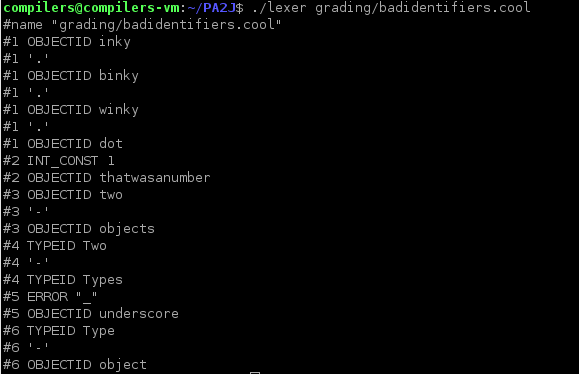
#5 OBJECTID underscore

#6 TYPEID Type

#6 '-'

#6 OBJECTID object

**测试完全符合**



bothcomments.cool

-- (\* This isn't

what the programmer thought it was \*)

.

(\* -- neither is this \*) if 5 then

bothcomments.cool.out

#name "bothcomments.cool"

#2 OBJECTID what

#2 OBJECTID the

#2 OBJECTID programmer

#2 OBJECTID thought

#2 OBJECTID it

#2 OBJECTID was

#2 ERROR "Unmatched \*)"

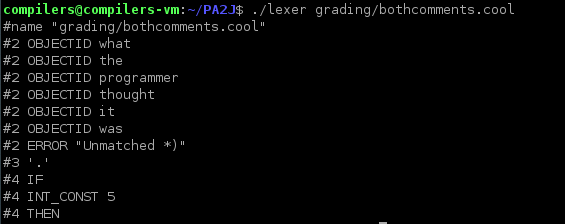
#3 '.'

#4 IF

#4 INT\_CONST 5

#4 THEN

**测试完全符合**



**all\_else\_true.cl.cool**

else

Else

eLse

elSe

elsE

ELse

ElSe

ElsE

eLSe

eLsE

elSE

eLSE

ElSE

ELsE

ELSe

ELSE

true

tRue

trUe

truE

tRUe

tRuE

trUE

tRUE

True

TRue

TrUe

TruE

TRUe

TRuE

TrUE

TRUE

#name "all\_else\_true.cl.cool"

#1 ELSE

#2 ELSE

#3 ELSE

#4 ELSE

#5 ELSE

#6 ELSE

#7 ELSE

#8 ELSE

#9 ELSE

#10 ELSE

#11 ELSE

#12 ELSE

#13 ELSE

#14 ELSE

#15 ELSE

#16 ELSE

#18 BOOL\_CONST true

#19 BOOL\_CONST true

#20 BOOL\_CONST true

#21 BOOL\_CONST true

#22 BOOL\_CONST true

#23 BOOL\_CONST true

#24 BOOL\_CONST true

#25 BOOL\_CONST true

#26 TYPEID True

#27 TYPEID TRue

#28 TYPEID TrUe

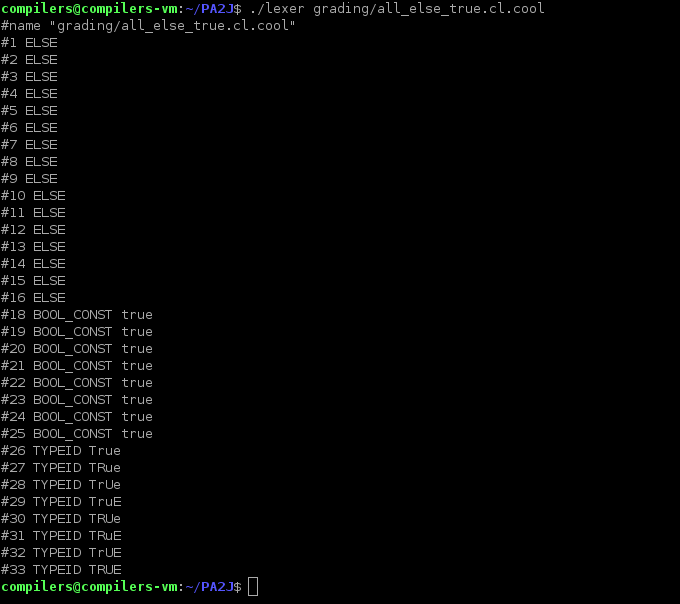
#29 TYPEID TruE

#30 TYPEID TRUe

#31 TYPEID TRuE

#32 TYPEID TrUE

#33 TYPEID TRUE



**arith.cool**

(\*

\* A contribution from Anne Sheets (sheets@cory)

\*

\* Tests the arithmetic operations and various other things

\*)

class A {

var : Int <- 0;

value() : Int { var };

set\_var(num : Int) : SELF\_TYPE {

{

var <- num;

self;

}

};

method1(num : Int) : SELF\_TYPE { -- same

self

};

method2(num1 : Int, num2 : Int) : B { -- plus

(let x : Int in

{

x <- num1 + num2;

(new B).set\_var(x);

}

)

};

method3(num : Int) : C { -- negate

(let x : Int in

{

x <- ~num;

(new C).set\_var(x);

}

)

};

method4(num1 : Int, num2 : Int) : D { -- diff

if num2 < num1 then

(let x : Int in

{

x <- num1 - num2;

(new D).set\_var(x);

}

)

else

(let x : Int in

{

x <- num2 - num1;

(new D).set\_var(x);

}

)

fi

};

method5(num : Int) : E { -- factorial

(let x : Int <- 1 in

{

(let y : Int <- 1 in

while y <= num loop

{

x <- x \* y;

y <- y + 1;

}

pool

);

(new E).set\_var(x);

}

)

};

};

class B inherits A { -- B is a number squared

method5(num : Int) : E { -- square

(let x : Int in

{

x <- num \* num;

(new E).set\_var(x);

}

)

};

};

class C inherits B {

method6(num : Int) : A { -- negate

(let x : Int in

{

x <- ~num;

(new A).set\_var(x);

}

)

};

method5(num : Int) : E { -- cube

(let x : Int in

{

x <- num \* num \* num;

(new E).set\_var(x);

}

)

};

};

class D inherits B {

method7(num : Int) : Bool { -- divisible by 3

(let x : Int <- num in

if x < 0 then method7(~x) else

if 0 = x then true else

if 1 = x then false else

if 2 = x then false else

method7(x - 3)

fi fi fi fi

)

};

};

class E inherits D {

method6(num : Int) : A { -- division

(let x : Int in

{

x <- num / 8;

(new A).set\_var(x);

}

)

};

};

(\* The following code is from atoi.cl in ~cs164/examples \*)

(\*

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 Main inherits IO {

char : String;

avar : A;

a\_var : A;

flag : Bool <- true;

menu() : String {

{

out\_string("\n\tTo add a number to ");

print(avar);

out\_string("...enter a:\n");

out\_string("\tTo negate ");

print(avar);

out\_string("...enter b:\n");

out\_string("\tTo find the difference between ");

print(avar);

out\_string("and another number...enter c:\n");

out\_string("\tTo find the factorial of ");

print(avar);

out\_string("...enter d:\n");

out\_string("\tTo square ");

print(avar);

out\_string("...enter e:\n");

out\_string("\tTo cube ");

print(avar);

out\_string("...enter f:\n");

out\_string("\tTo find out if ");

print(avar);

out\_string("is a multiple of 3...enter g:\n");

out\_string("\tTo divide ");

print(avar);

out\_string("by 8...enter h:\n");

out\_string("\tTo get a new number...enter j:\n");

out\_string("\tTo quit...enter q:\n\n");

in\_string();

}

};

prompt() : String {

{

out\_string("\n");

out\_string("Please enter a number... ");

in\_string();

}

};

get\_int() : Int {

{

(let z : A2I <- new A2I in

(let s : String <- prompt() in

z.a2i(s)

)

);

}

};

is\_even(num : Int) : Bool {

(let x : Int <- num in

if x < 0 then is\_even(~x) else

if 0 = x then true else

if 1 = x then false else

is\_even(x - 2)

fi fi fi

)

};

class\_type(var : A) : SELF\_TYPE {

case var of

a : A => out\_string("Class type is now A\n");

b : B => out\_string("Class type is now B\n");

c : C => out\_string("Class type is now C\n");

d : D => out\_string("Class type is now D\n");

e : E => out\_string("Class type is now E\n");

o : Object => out\_string("Oooops\n");

esac

};

print(var : A) : SELF\_TYPE {

(let z : A2I <- new A2I in

{

out\_string(z.i2a(var.value()));

out\_string(" ");

}

)

};

main() : Object {

{

avar <- (new A);

while flag loop

{

-- avar <- (new A).set\_var(get\_int());

out\_string("number ");

print(avar);

if is\_even(avar.value()) then

out\_string("is even!\n")

else

out\_string("is odd!\n")

fi;

-- print(avar); -- prints out answer

class\_type(avar);

char <- menu();

if char = "a" then -- add

{

a\_var <- (new A).set\_var(get\_int());

avar <- (new B).method2(avar.value(), a\_var.value());

} else

if char = "b" then -- negate

case avar of

c : C => avar <- c.method6(c.value());

a : A => avar <- a.method3(a.value());

o : Object => {

out\_string("Oooops\n");

abort(); 0;

};

esac else

if char = "c" then -- diff

{

a\_var <- (new A).set\_var(get\_int());

avar <- (new D).method4(avar.value(), a\_var.value());

} else

if char = "d" then avar <- (new C)@A.method5(avar.value()) else

-- factorial

if char = "e" then avar <- (new C)@B.method5(avar.value()) else

-- square

if char = "f" then avar <- (new C)@C.method5(avar.value()) else

-- cube

if char = "g" then -- multiple of 3?

if ((new D).method7(avar.value()))

then -- avar <- (new A).method1(avar.value())

{

out\_string("number ");

print(avar);

out\_string("is divisible by 3.\n");

}

else -- avar <- (new A).set\_var(0)

{

out\_string("number ");

print(avar);

out\_string("is not divisible by 3.\n");

}

fi else

if char = "h" then

(let x : A in

{

x <- (new E).method6(avar.value());

(let r : Int <- (avar.value() - (x.value() \* 8)) in

{

out\_string("number ");

print(avar);

out\_string("is equal to ");

print(x);

out\_string("times 8 with a remainder of ");

(let a : A2I <- new A2I in

{

out\_string(a.i2a(r));

out\_string("\n");

}

); -- end let a:

}

); -- end let r:

avar <- x;

}

) -- end let x:

else

if char = "j" then avar <- (new A)

else

if char = "q" then flag <- false

else

avar <- (new A).method1(avar.value()) -- divide/8

fi fi fi fi fi fi fi fi fi fi;

}

pool;

}

};

};

#name "arith.cool"

#7 CLASS

#7 TYPEID A

#7 '{'

#9 OBJECTID var

#9 ':'

#9 TYPEID Int

#9 ASSIGN

#9 INT\_CONST 0

#9 ';'

#11 OBJECTID value

#11 '('

#11 ')'

#11 ':'

#11 TYPEID Int

#11 '{'

#11 OBJECTID var

#11 '}'

#11 ';'

#13 OBJECTID set\_var

#13 '('

#13 OBJECTID num

#13 ':'

#13 TYPEID Int

#13 ')'

#13 ':'

#13 TYPEID SELF\_TYPE

#13 '{'

#14 '{'

#15 OBJECTID var

#15 ASSIGN

#15 OBJECTID num

#15 ';'

#16 OBJECTID self

#16 ';'

#17 '}'

#18 '}'

#18 ';'

#20 OBJECTID method1

#20 '('

#20 OBJECTID num

#20 ':'

#20 TYPEID Int

#20 ')'

#20 ':'

#20 TYPEID SELF\_TYPE

#20 '{'

#21 OBJECTID self

#22 '}'

#22 ';'

#24 OBJECTID method2

#24 '('

#24 OBJECTID num1

#24 ':'

#24 TYPEID Int

#24 ','

#24 OBJECTID num2

#24 ':'

#24 TYPEID Int

#24 ')'

#24 ':'

#24 TYPEID B

#24 '{'

#25 '('

#25 LET

#25 OBJECTID x

#25 ':'

#25 TYPEID Int

#25 IN

#26 '{'

#27 OBJECTID x

#27 ASSIGN

#27 OBJECTID num1

#27 '+'

#27 OBJECTID num2

#27 ';'

#28 '('

#28 NEW

#28 TYPEID B

#28 ')'

#28 '.'

#28 OBJECTID set\_var

#28 '('

#28 OBJECTID x

#28 ')'

#28 ';'

#29 '}'

#30 ')'

#31 '}'

#31 ';'

#33 OBJECTID method3

#33 '('

#33 OBJECTID num

#33 ':'

#33 TYPEID Int

#33 ')'

#33 ':'

#33 TYPEID C

#33 '{'

#34 '('

#34 LET

#34 OBJECTID x

#34 ':'

#34 TYPEID Int

#34 IN

#35 '{'

#36 OBJECTID x

#36 ASSIGN

#36 '~'

#36 OBJECTID num

#36 ';'

#37 '('

#37 NEW

#37 TYPEID C

#37 ')'

#37 '.'

#37 OBJECTID set\_var

#37 '('

#37 OBJECTID x

#37 ')'

#37 ';'

#38 '}'

#39 ')'

#40 '}'

#40 ';'

#42 OBJECTID method4

#42 '('

#42 OBJECTID num1

#42 ':'

#42 TYPEID Int

#42 ','

#42 OBJECTID num2

#42 ':'

#42 TYPEID Int

#42 ')'

#42 ':'

#42 TYPEID D

#42 '{'

#43 IF

#43 OBJECTID num2

#43 '<'

#43 OBJECTID num1

#43 THEN

#44 '('

#44 LET

#44 OBJECTID x

#44 ':'

#44 TYPEID Int

#44 IN

#45 '{'

#46 OBJECTID x

#46 ASSIGN

#46 OBJECTID num1

#46 '-'

#46 OBJECTID num2

#46 ';'

#47 '('

#47 NEW

#47 TYPEID D

#47 ')'

#47 '.'

#47 OBJECTID set\_var

#47 '('

#47 OBJECTID x

#47 ')'

#47 ';'

#48 '}'

#49 ')'

#50 ELSE

#51 '('

#51 LET

#51 OBJECTID x

#51 ':'

#51 TYPEID Int

#51 IN

#52 '{'

#53 OBJECTID x

#53 ASSIGN

#53 OBJECTID num2

#53 '-'

#53 OBJECTID num1

#53 ';'

#54 '('

#54 NEW

#54 TYPEID D

#54 ')'

#54 '.'

#54 OBJECTID set\_var

#54 '('

#54 OBJECTID x

#54 ')'

#54 ';'

#55 '}'

#56 ')'

#57 FI

#58 '}'

#58 ';'

#60 OBJECTID method5

#60 '('

#60 OBJECTID num

#60 ':'

#60 TYPEID Int

#60 ')'

#60 ':'

#60 TYPEID E

#60 '{'

#61 '('

#61 LET

#61 OBJECTID x

#61 ':'

#61 TYPEID Int

#61 ASSIGN

#61 INT\_CONST 1

#61 IN

#62 '{'

#63 '('

#63 LET

#63 OBJECTID y

#63 ':'

#63 TYPEID Int

#63 ASSIGN

#63 INT\_CONST 1

#63 IN

#64 WHILE

#64 OBJECTID y

#64 LE

#64 OBJECTID num

#64 LOOP

#65 '{'

#66 OBJECTID x

#66 ASSIGN

#66 OBJECTID x

#66 '\*'

#66 OBJECTID y

#66 ';'

#67 OBJECTID y

#67 ASSIGN

#67 OBJECTID y

#67 '+'

#67 INT\_CONST 1

#67 ';'

#68 '}'

#69 POOL

#70 ')'

#70 ';'

#71 '('

#71 NEW

#71 TYPEID E

#71 ')'

#71 '.'

#71 OBJECTID set\_var

#71 '('

#71 OBJECTID x

#71 ')'

#71 ';'

#72 '}'

#73 ')'

#74 '}'

#74 ';'

#76 '}'

#76 ';'

#78 CLASS

#78 TYPEID B

#78 INHERITS

#78 TYPEID A

#78 '{'

#80 OBJECTID method5

#80 '('

#80 OBJECTID num

#80 ':'

#80 TYPEID Int

#80 ')'

#80 ':'

#80 TYPEID E

#80 '{'

#81 '('

#81 LET

#81 OBJECTID x

#81 ':'

#81 TYPEID Int

#81 IN

#82 '{'

#83 OBJECTID x

#83 ASSIGN

#83 OBJECTID num

#83 '\*'

#83 OBJECTID num

#83 ';'

#84 '('

#84 NEW

#84 TYPEID E

#84 ')'

#84 '.'

#84 OBJECTID set\_var

#84 '('

#84 OBJECTID x

#84 ')'

#84 ';'

#85 '}'

#86 ')'

#87 '}'

#87 ';'

#89 '}'

#89 ';'

#91 CLASS

#91 TYPEID C

#91 INHERITS

#91 TYPEID B

#91 '{'

#93 OBJECTID method6

#93 '('

#93 OBJECTID num

#93 ':'

#93 TYPEID Int

#93 ')'

#93 ':'

#93 TYPEID A

#93 '{'

#94 '('

#94 LET

#94 OBJECTID x

#94 ':'

#94 TYPEID Int

#94 IN

#95 '{'

#96 OBJECTID x

#96 ASSIGN

#96 '~'

#96 OBJECTID num

#96 ';'

#97 '('

#97 NEW

#97 TYPEID A

#97 ')'

#97 '.'

#97 OBJECTID set\_var

#97 '('

#97 OBJECTID x

#97 ')'

#97 ';'

#98 '}'

#99 ')'

#100 '}'

#100 ';'

#102 OBJECTID method5

#102 '('

#102 OBJECTID num

#102 ':'

#102 TYPEID Int

#102 ')'

#102 ':'

#102 TYPEID E

#102 '{'

#103 '('

#103 LET

#103 OBJECTID x

#103 ':'

#103 TYPEID Int

#103 IN

#104 '{'

#105 OBJECTID x

#105 ASSIGN

#105 OBJECTID num

#105 '\*'

#105 OBJECTID num

#105 '\*'

#105 OBJECTID num

#105 ';'

#106 '('

#106 NEW

#106 TYPEID E

#106 ')'

#106 '.'

#106 OBJECTID set\_var

#106 '('

#106 OBJECTID x

#106 ')'

#106 ';'

#107 '}'

#108 ')'

#109 '}'

#109 ';'

#111 '}'

#111 ';'

#113 CLASS

#113 TYPEID D

#113 INHERITS

#113 TYPEID B

#113 '{'

#115 OBJECTID method7

#115 '('

#115 OBJECTID num

#115 ':'

#115 TYPEID Int

#115 ')'

#115 ':'

#115 TYPEID Bool

#115 '{'

#116 '('

#116 LET

#116 OBJECTID x

#116 ':'

#116 TYPEID Int

#116 ASSIGN

#116 OBJECTID num

#116 IN

#117 IF

#117 OBJECTID x

#117 '<'

#117 INT\_CONST 0

#117 THEN

#117 OBJECTID method7

#117 '('

#117 '~'

#117 OBJECTID x

#117 ')'

#117 ELSE

#118 IF

#118 INT\_CONST 0

#118 '='

#118 OBJECTID x

#118 THEN

#118 BOOL\_CONST true

#118 ELSE

#119 IF

#119 INT\_CONST 1

#119 '='

#119 OBJECTID x

#119 THEN

#119 BOOL\_CONST false

#119 ELSE

#120 IF

#120 INT\_CONST 2

#120 '='

#120 OBJECTID x

#120 THEN

#120 BOOL\_CONST false

#120 ELSE

#121 OBJECTID method7

#121 '('

#121 OBJECTID x

#121 '-'

#121 INT\_CONST 3

#121 ')'

#122 FI

#122 FI

#122 FI

#122 FI

#123 ')'

#124 '}'

#124 ';'

#126 '}'

#126 ';'

#128 CLASS

#128 TYPEID E

#128 INHERITS

#128 TYPEID D

#128 '{'

#130 OBJECTID method6

#130 '('

#130 OBJECTID num

#130 ':'

#130 TYPEID Int

#130 ')'

#130 ':'

#130 TYPEID A

#130 '{'

#131 '('

#131 LET

#131 OBJECTID x

#131 ':'

#131 TYPEID Int

#131 IN

#132 '{'

#133 OBJECTID x

#133 ASSIGN

#133 OBJECTID num

#133 '/'

#133 INT\_CONST 8

#133 ';'

#134 '('

#134 NEW

#134 TYPEID A

#134 ')'

#134 '.'

#134 OBJECTID set\_var

#134 '('

#134 OBJECTID x

#134 ')'

#134 ';'

#135 '}'

#136 ')'

#137 '}'

#137 ';'

#139 '}'

#139 ';'

#155 CLASS

#155 TYPEID A2I

#155 '{'

#157 OBJECTID c2i

#157 '('

#157 OBJECTID char

#157 ':'

#157 TYPEID String

#157 ')'

#157 ':'

#157 TYPEID Int

#157 '{'

#158 IF

#158 OBJECTID char

#158 '='

#158 STR\_CONST "0"

#158 THEN

#158 INT\_CONST 0

#158 ELSE

#159 IF

#159 OBJECTID char

#159 '='

#159 STR\_CONST "1"

#159 THEN

#159 INT\_CONST 1

#159 ELSE

#160 IF

#160 OBJECTID char

#160 '='

#160 STR\_CONST "2"

#160 THEN

#160 INT\_CONST 2

#160 ELSE

#161 IF

#161 OBJECTID char

#161 '='

#161 STR\_CONST "3"

#161 THEN

#161 INT\_CONST 3

#161 ELSE

#162 IF

#162 OBJECTID char

#162 '='

#162 STR\_CONST "4"

#162 THEN

#162 INT\_CONST 4

#162 ELSE

#163 IF

#163 OBJECTID char

#163 '='

#163 STR\_CONST "5"

#163 THEN

#163 INT\_CONST 5

#163 ELSE

#164 IF

#164 OBJECTID char

#164 '='

#164 STR\_CONST "6"

#164 THEN

#164 INT\_CONST 6

#164 ELSE

#165 IF

#165 OBJECTID char

#165 '='

#165 STR\_CONST "7"

#165 THEN

#165 INT\_CONST 7

#165 ELSE

#166 IF

#166 OBJECTID char

#166 '='

#166 STR\_CONST "8"

#166 THEN

#166 INT\_CONST 8

#166 ELSE

#167 IF

#167 OBJECTID char

#167 '='

#167 STR\_CONST "9"

#167 THEN

#167 INT\_CONST 9

#167 ELSE

#168 '{'

#168 OBJECTID abort

#168 '('

#168 ')'

#168 ';'

#168 INT\_CONST 0

#168 ';'

#168 '}'

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#170 FI

#171 '}'

#171 ';'

#176 OBJECTID i2c

#176 '('

#176 OBJECTID i

#176 ':'

#176 TYPEID Int

#176 ')'

#176 ':'

#176 TYPEID String

#176 '{'

#177 IF

#177 OBJECTID i

#177 '='

#177 INT\_CONST 0

#177 THEN

#177 STR\_CONST "0"

#177 ELSE

#178 IF

#178 OBJECTID i

#178 '='

#178 INT\_CONST 1

#178 THEN

#178 STR\_CONST "1"

#178 ELSE

#179 IF

#179 OBJECTID i

#179 '='

#179 INT\_CONST 2

#179 THEN

#179 STR\_CONST "2"

#179 ELSE

#180 IF

#180 OBJECTID i

#180 '='

#180 INT\_CONST 3

#180 THEN

#180 STR\_CONST "3"

#180 ELSE

#181 IF

#181 OBJECTID i

#181 '='

#181 INT\_CONST 4

#181 THEN

#181 STR\_CONST "4"

#181 ELSE

#182 IF

#182 OBJECTID i

#182 '='

#182 INT\_CONST 5

#182 THEN

#182 STR\_CONST "5"

#182 ELSE

#183 IF

#183 OBJECTID i

#183 '='

#183 INT\_CONST 6

#183 THEN

#183 STR\_CONST "6"

#183 ELSE

#184 IF

#184 OBJECTID i

#184 '='

#184 INT\_CONST 7

#184 THEN

#184 STR\_CONST "7"

#184 ELSE

#185 IF

#185 OBJECTID i

#185 '='

#185 INT\_CONST 8

#185 THEN

#185 STR\_CONST "8"

#185 ELSE

#186 IF

#186 OBJECTID i

#186 '='

#186 INT\_CONST 9

#186 THEN

#186 STR\_CONST "9"

#186 ELSE

#187 '{'

#187 OBJECTID abort

#187 '('

#187 ')'

#187 ';'

#187 STR\_CONST ""

#187 ';'

#187 '}'

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#188 FI

#189 '}'

#189 ';'

#199 OBJECTID a2i

#199 '('

#199 OBJECTID s

#199 ':'

#199 TYPEID String

#199 ')'

#199 ':'

#199 TYPEID Int

#199 '{'

#200 IF

#200 OBJECTID s

#200 '.'

#200 OBJECTID length

#200 '('

#200 ')'

#200 '='

#200 INT\_CONST 0

#200 THEN

#200 INT\_CONST 0

#200 ELSE

#201 IF

#201 OBJECTID s

#201 '.'

#201 OBJECTID substr

#201 '('

#201 INT\_CONST 0

#201 ','

#201 INT\_CONST 1

#201 ')'

#201 '='

#201 STR\_CONST "-"

#201 THEN

#201 '~'

#201 OBJECTID a2i\_aux

#201 '('

#201 OBJECTID s

#201 '.'

#201 OBJECTID substr

#201 '('

#201 INT\_CONST 1

#201 ','

#201 OBJECTID s

#201 '.'

#201 OBJECTID length

#201 '('

#201 ')'

#201 '-'

#201 INT\_CONST 1

#201 ')'

#201 ')'

#201 ELSE

#202 IF

#202 OBJECTID s

#202 '.'

#202 OBJECTID substr

#202 '('

#202 INT\_CONST 0

#202 ','

#202 INT\_CONST 1

#202 ')'

#202 '='

#202 STR\_CONST "+"

#202 THEN

#202 OBJECTID a2i\_aux

#202 '('

#202 OBJECTID s

#202 '.'

#202 OBJECTID substr

#202 '('

#202 INT\_CONST 1

#202 ','

#202 OBJECTID s

#202 '.'

#202 OBJECTID length

#202 '('

#202 ')'

#202 '-'

#202 INT\_CONST 1

#202 ')'

#202 ')'

#202 ELSE

#203 OBJECTID a2i\_aux

#203 '('

#203 OBJECTID s

#203 ')'

#204 FI

#204 FI

#204 FI

#205 '}'

#205 ';'

#211 OBJECTID a2i\_aux

#211 '('

#211 OBJECTID s

#211 ':'

#211 TYPEID String

#211 ')'

#211 ':'

#211 TYPEID Int

#211 '{'

#212 '('

#212 LET

#212 OBJECTID int

#212 ':'

#212 TYPEID Int

#212 ASSIGN

#212 INT\_CONST 0

#212 IN

#213 '{'

#214 '('

#214 LET

#214 OBJECTID j

#214 ':'

#214 TYPEID Int

#214 ASSIGN

#214 OBJECTID s

#214 '.'

#214 OBJECTID length

#214 '('

#214 ')'

#214 IN

#215 '('

#215 LET

#215 OBJECTID i

#215 ':'

#215 TYPEID Int

#215 ASSIGN

#215 INT\_CONST 0

#215 IN

#216 WHILE

#216 OBJECTID i

#216 '<'

#216 OBJECTID j

#216 LOOP

#217 '{'

#218 OBJECTID int

#218 ASSIGN

#218 OBJECTID int

#218 '\*'

#218 INT\_CONST 10

#218 '+'

#218 OBJECTID c2i

#218 '('

#218 OBJECTID s

#218 '.'

#218 OBJECTID substr

#218 '('

#218 OBJECTID i

#218 ','

#218 INT\_CONST 1

#218 ')'

#218 ')'

#218 ';'

#219 OBJECTID i

#219 ASSIGN

#219 OBJECTID i

#219 '+'

#219 INT\_CONST 1

#219 ';'

#220 '}'

#221 POOL

#222 ')'

#223 ')'

#223 ';'

#224 OBJECTID int

#224 ';'

#225 '}'

#226 ')'

#227 '}'

#227 ';'

#232 OBJECTID i2a

#232 '('

#232 OBJECTID i

#232 ':'

#232 TYPEID Int

#232 ')'

#232 ':'

#232 TYPEID String

#232 '{'

#233 IF

#233 OBJECTID i

#233 '='

#233 INT\_CONST 0

#233 THEN

#233 STR\_CONST "0"

#233 ELSE

#234 IF

#234 INT\_CONST 0

#234 '<'

#234 OBJECTID i

#234 THEN

#234 OBJECTID i2a\_aux

#234 '('

#234 OBJECTID i

#234 ')'

#234 ELSE

#235 STR\_CONST "-"

#235 '.'

#235 OBJECTID concat

#235 '('

#235 OBJECTID i2a\_aux

#235 '('

#235 OBJECTID i

#235 '\*'

#235 '~'

#235 INT\_CONST 1

#235 ')'

#235 ')'

#236 FI

#236 FI

#237 '}'

#237 ';'

#241 OBJECTID i2a\_aux

#241 '('

#241 OBJECTID i

#241 ':'

#241 TYPEID Int

#241 ')'

#241 ':'

#241 TYPEID String

#241 '{'

#242 IF

#242 OBJECTID i

#242 '='

#242 INT\_CONST 0

#242 THEN

#242 STR\_CONST ""

#242 ELSE

#243 '('

#243 LET

#243 OBJECTID next

#243 ':'

#243 TYPEID Int

#243 ASSIGN

#243 OBJECTID i

#243 '/'

#243 INT\_CONST 10

#243 IN

#244 OBJECTID i2a\_aux

#244 '('

#244 OBJECTID next

#244 ')'

#244 '.'

#244 OBJECTID concat

#244 '('

#244 OBJECTID i2c

#244 '('

#244 OBJECTID i

#244 '-'

#244 OBJECTID next

#244 '\*'

#244 INT\_CONST 10

#244 ')'

#244 ')'

#245 ')'

#246 FI

#247 '}'

#247 ';'

#249 '}'

#249 ';'

#251 CLASS

#251 TYPEID Main

#251 INHERITS

#251 TYPEID IO

#251 '{'

#253 OBJECTID char

#253 ':'

#253 TYPEID String

#253 ';'

#254 OBJECTID avar

#254 ':'

#254 TYPEID A

#254 ';'

#255 OBJECTID a\_var

#255 ':'

#255 TYPEID A

#255 ';'

#256 OBJECTID flag

#256 ':'

#256 TYPEID Bool

#256 ASSIGN

#256 BOOL\_CONST true

#256 ';'

#259 OBJECTID menu

#259 '('

#259 ')'

#259 ':'

#259 TYPEID String

#259 '{'

#260 '{'

#261 OBJECTID out\_string

#261 '('

#261 STR\_CONST "\n\tTo add a number to "

#261 ')'

#261 ';'

#262 OBJECTID print

#262 '('

#262 OBJECTID avar

#262 ')'

#262 ';'

#263 OBJECTID out\_string

#263 '('

#263 STR\_CONST "...enter a:\n"

#263 ')'

#263 ';'

#264 OBJECTID out\_string

#264 '('

#264 STR\_CONST "\tTo negate "

#264 ')'

#264 ';'

#265 OBJECTID print

#265 '('

#265 OBJECTID avar

#265 ')'

#265 ';'

#266 OBJECTID out\_string

#266 '('

#266 STR\_CONST "...enter b:\n"

#266 ')'

#266 ';'

#267 OBJECTID out\_string

#267 '('

#267 STR\_CONST "\tTo find the difference between "

#267 ')'

#267 ';'

#268 OBJECTID print

#268 '('

#268 OBJECTID avar

#268 ')'

#268 ';'

#269 OBJECTID out\_string

#269 '('

#269 STR\_CONST "and another number...enter c:\n"

#269 ')'

#269 ';'

#270 OBJECTID out\_string

#270 '('

#270 STR\_CONST "\tTo find the factorial of "

#270 ')'

#270 ';'

#271 OBJECTID print

#271 '('

#271 OBJECTID avar

#271 ')'

#271 ';'

#272 OBJECTID out\_string

#272 '('

#272 STR\_CONST "...enter d:\n"

#272 ')'

#272 ';'

#273 OBJECTID out\_string

#273 '('

#273 STR\_CONST "\tTo square "

#273 ')'

#273 ';'

#274 OBJECTID print

#274 '('

#274 OBJECTID avar

#274 ')'

#274 ';'

#275 OBJECTID out\_string

#275 '('

#275 STR\_CONST "...enter e:\n"

#275 ')'

#275 ';'

#276 OBJECTID out\_string

#276 '('

#276 STR\_CONST "\tTo cube "

#276 ')'

#276 ';'

#277 OBJECTID print

#277 '('

#277 OBJECTID avar

#277 ')'

#277 ';'

#278 OBJECTID out\_string

#278 '('

#278 STR\_CONST "...enter f:\n"

#278 ')'

#278 ';'

#279 OBJECTID out\_string

#279 '('

#279 STR\_CONST "\tTo find out if "

#279 ')'

#279 ';'

#280 OBJECTID print

#280 '('

#280 OBJECTID avar

#280 ')'

#280 ';'

#281 OBJECTID out\_string

#281 '('

#281 STR\_CONST "is a multiple of 3...enter g:\n"

#281 ')'

#281 ';'

#282 OBJECTID out\_string

#282 '('

#282 STR\_CONST "\tTo divide "

#282 ')'

#282 ';'

#283 OBJECTID print

#283 '('

#283 OBJECTID avar

#283 ')'

#283 ';'

#284 OBJECTID out\_string

#284 '('

#284 STR\_CONST "by 8...enter h:\n"

#284 ')'

#284 ';'

#285 OBJECTID out\_string

#285 '('

#285 STR\_CONST "\tTo get a new number...enter j:\n"

#285 ')'

#285 ';'

#286 OBJECTID out\_string

#286 '('

#286 STR\_CONST "\tTo quit...enter q:\n\n"

#286 ')'

#286 ';'

#287 OBJECTID in\_string

#287 '('

#287 ')'

#287 ';'

#288 '}'

#289 '}'

#289 ';'

#291 OBJECTID prompt

#291 '('

#291 ')'

#291 ':'

#291 TYPEID String

#291 '{'

#292 '{'

#293 OBJECTID out\_string

#293 '('

#293 STR\_CONST "\n"

#293 ')'

#293 ';'

#294 OBJECTID out\_string

#294 '('

#294 STR\_CONST "Please enter a number... "

#294 ')'

#294 ';'

#295 OBJECTID in\_string

#295 '('

#295 ')'

#295 ';'

#296 '}'

#297 '}'

#297 ';'

#299 OBJECTID get\_int

#299 '('

#299 ')'

#299 ':'

#299 TYPEID Int

#299 '{'

#300 '{'

#301 '('

#301 LET

#301 OBJECTID z

#301 ':'

#301 TYPEID A2I

#301 ASSIGN

#301 NEW

#301 TYPEID A2I

#301 IN

#302 '('

#302 LET

#302 OBJECTID s

#302 ':'

#302 TYPEID String

#302 ASSIGN

#302 OBJECTID prompt

#302 '('

#302 ')'

#302 IN

#303 OBJECTID z

#303 '.'

#303 OBJECTID a2i

#303 '('

#303 OBJECTID s

#303 ')'

#304 ')'

#305 ')'

#305 ';'

#306 '}'

#307 '}'

#307 ';'

#309 OBJECTID is\_even

#309 '('

#309 OBJECTID num

#309 ':'

#309 TYPEID Int

#309 ')'

#309 ':'

#309 TYPEID Bool

#309 '{'

#310 '('

#310 LET

#310 OBJECTID x

#310 ':'

#310 TYPEID Int

#310 ASSIGN

#310 OBJECTID num

#310 IN

#311 IF

#311 OBJECTID x

#311 '<'

#311 INT\_CONST 0

#311 THEN

#311 OBJECTID is\_even

#311 '('

#311 '~'

#311 OBJECTID x

#311 ')'

#311 ELSE

#312 IF

#312 INT\_CONST 0

#312 '='

#312 OBJECTID x

#312 THEN

#312 BOOL\_CONST true

#312 ELSE

#313 IF

#313 INT\_CONST 1

#313 '='

#313 OBJECTID x

#313 THEN

#313 BOOL\_CONST false

#313 ELSE

#314 OBJECTID is\_even

#314 '('

#314 OBJECTID x

#314 '-'

#314 INT\_CONST 2

#314 ')'

#315 FI

#315 FI

#315 FI

#316 ')'

#317 '}'

#317 ';'

#319 OBJECTID class\_type

#319 '('

#319 OBJECTID var

#319 ':'

#319 TYPEID A

#319 ')'

#319 ':'

#319 TYPEID SELF\_TYPE

#319 '{'

#320 CASE

#320 OBJECTID var

#320 OF

#321 OBJECTID a

#321 ':'

#321 TYPEID A

#321 DARROW

#321 OBJECTID out\_string

#321 '('

#321 STR\_CONST "Class type is now A\n"

#321 ')'

#321 ';'

#322 OBJECTID b

#322 ':'

#322 TYPEID B

#322 DARROW

#322 OBJECTID out\_string

#322 '('

#322 STR\_CONST "Class type is now B\n"

#322 ')'

#322 ';'

#323 OBJECTID c

#323 ':'

#323 TYPEID C

#323 DARROW

#323 OBJECTID out\_string

#323 '('

#323 STR\_CONST "Class type is now C\n"

#323 ')'

#323 ';'

#324 OBJECTID d

#324 ':'

#324 TYPEID D

#324 DARROW

#324 OBJECTID out\_string

#324 '('

#324 STR\_CONST "Class type is now D\n"

#324 ')'

#324 ';'

#325 OBJECTID e

#325 ':'

#325 TYPEID E

#325 DARROW

#325 OBJECTID out\_string

#325 '('

#325 STR\_CONST "Class type is now E\n"

#325 ')'

#325 ';'

#326 OBJECTID o

#326 ':'

#326 TYPEID Object

#326 DARROW

#326 OBJECTID out\_string

#326 '('

#326 STR\_CONST "Oooops\n"

#326 ')'

#326 ';'

#327 ESAC

#328 '}'

#328 ';'

#330 OBJECTID print

#330 '('

#330 OBJECTID var

#330 ':'

#330 TYPEID A

#330 ')'

#330 ':'

#330 TYPEID SELF\_TYPE

#330 '{'

#331 '('

#331 LET

#331 OBJECTID z

#331 ':'

#331 TYPEID A2I

#331 ASSIGN

#331 NEW

#331 TYPEID A2I

#331 IN

#332 '{'

#333 OBJECTID out\_string

#333 '('

#333 OBJECTID z

#333 '.'

#333 OBJECTID i2a

#333 '('

#333 OBJECTID var

#333 '.'

#333 OBJECTID value

#333 '('

#333 ')'

#333 ')'

#333 ')'

#333 ';'

#334 OBJECTID out\_string

#334 '('

#334 STR\_CONST " "

#334 ')'

#334 ';'

#335 '}'

#336 ')'

#337 '}'

#337 ';'

#339 OBJECTID main

#339 '('

#339 ')'

#339 ':'

#339 TYPEID Object

#339 '{'

#340 '{'

#341 OBJECTID avar

#341 ASSIGN

#341 '('

#341 NEW

#341 TYPEID A

#341 ')'

#341 ';'

#342 WHILE

#342 OBJECTID flag

#342 LOOP

#343 '{'

#345 OBJECTID out\_string

#345 '('

#345 STR\_CONST "number "

#345 ')'

#345 ';'

#346 OBJECTID print

#346 '('

#346 OBJECTID avar

#346 ')'

#346 ';'

#347 IF

#347 OBJECTID is\_even

#347 '('

#347 OBJECTID avar

#347 '.'

#347 OBJECTID value

#347 '('

#347 ')'

#347 ')'

#347 THEN

#348 OBJECTID out\_string

#348 '('

#348 STR\_CONST "is even!\n"

#348 ')'

#349 ELSE

#350 OBJECTID out\_string

#350 '('

#350 STR\_CONST "is odd!\n"

#350 ')'

#351 FI

#351 ';'

#353 OBJECTID class\_type

#353 '('

#353 OBJECTID avar

#353 ')'

#353 ';'

#354 OBJECTID char

#354 ASSIGN

#354 OBJECTID menu

#354 '('

#354 ')'

#354 ';'

#355 IF

#355 OBJECTID char

#355 '='

#355 STR\_CONST "a"

#355 THEN

#356 '{'

#357 OBJECTID a\_var

#357 ASSIGN

#357 '('

#357 NEW

#357 TYPEID A

#357 ')'

#357 '.'

#357 OBJECTID set\_var

#357 '('

#357 OBJECTID get\_int

#357 '('

#357 ')'

#357 ')'

#357 ';'

#358 OBJECTID avar

#358 ASSIGN

#358 '('

#358 NEW

#358 TYPEID B

#358 ')'

#358 '.'

#358 OBJECTID method2

#358 '('

#358 OBJECTID avar

#358 '.'

#358 OBJECTID value

#358 '('

#358 ')'

#358 ','

#358 OBJECTID a\_var

#358 '.'

#358 OBJECTID value

#358 '('

#358 ')'

#358 ')'

#358 ';'

#359 '}'

#359 ELSE

#360 IF

#360 OBJECTID char

#360 '='

#360 STR\_CONST "b"

#360 THEN

#361 CASE

#361 OBJECTID avar

#361 OF

#362 OBJECTID c

#362 ':'

#362 TYPEID C

#362 DARROW

#362 OBJECTID avar

#362 ASSIGN

#362 OBJECTID c

#362 '.'

#362 OBJECTID method6

#362 '('

#362 OBJECTID c

#362 '.'

#362 OBJECTID value

#362 '('

#362 ')'

#362 ')'

#362 ';'

#363 OBJECTID a

#363 ':'

#363 TYPEID A

#363 DARROW

#363 OBJECTID avar

#363 ASSIGN

#363 OBJECTID a

#363 '.'

#363 OBJECTID method3

#363 '('

#363 OBJECTID a

#363 '.'

#363 OBJECTID value

#363 '('

#363 ')'

#363 ')'

#363 ';'

#364 OBJECTID o

#364 ':'

#364 TYPEID Object

#364 DARROW

#364 '{'

#365 OBJECTID out\_string

#365 '('

#365 STR\_CONST "Oooops\n"

#365 ')'

#365 ';'

#366 OBJECTID abort

#366 '('

#366 ')'

#366 ';'

#366 INT\_CONST 0

#366 ';'

#367 '}'

#367 ';'

#368 ESAC

#368 ELSE

#369 IF

#369 OBJECTID char

#369 '='

#369 STR\_CONST "c"

#369 THEN

#370 '{'

#371 OBJECTID a\_var

#371 ASSIGN

#371 '('

#371 NEW

#371 TYPEID A

#371 ')'

#371 '.'

#371 OBJECTID set\_var

#371 '('

#371 OBJECTID get\_int

#371 '('

#371 ')'

#371 ')'

#371 ';'

#372 OBJECTID avar

#372 ASSIGN

#372 '('

#372 NEW

#372 TYPEID D

#372 ')'

#372 '.'

#372 OBJECTID method4

#372 '('

#372 OBJECTID avar

#372 '.'

#372 OBJECTID value

#372 '('

#372 ')'

#372 ','

#372 OBJECTID a\_var

#372 '.'

#372 OBJECTID value

#372 '('

#372 ')'

#372 ')'

#372 ';'

#373 '}'

#373 ELSE

#374 IF

#374 OBJECTID char

#374 '='

#374 STR\_CONST "d"

#374 THEN

#374 OBJECTID avar

#374 ASSIGN

#374 '('

#374 NEW

#374 TYPEID C

#374 ')'

#374 '@'

#374 TYPEID A

#374 '.'

#374 OBJECTID method5

#374 '('

#374 OBJECTID avar

#374 '.'

#374 OBJECTID value

#374 '('

#374 ')'

#374 ')'

#374 ELSE

#376 IF

#376 OBJECTID char

#376 '='

#376 STR\_CONST "e"

#376 THEN

#376 OBJECTID avar

#376 ASSIGN

#376 '('

#376 NEW

#376 TYPEID C

#376 ')'

#376 '@'

#376 TYPEID B

#376 '.'

#376 OBJECTID method5

#376 '('

#376 OBJECTID avar

#376 '.'

#376 OBJECTID value

#376 '('

#376 ')'

#376 ')'

#376 ELSE

#378 IF

#378 OBJECTID char

#378 '='

#378 STR\_CONST "f"

#378 THEN

#378 OBJECTID avar

#378 ASSIGN

#378 '('

#378 NEW

#378 TYPEID C

#378 ')'

#378 '@'

#378 TYPEID C

#378 '.'

#378 OBJECTID method5

#378 '('

#378 OBJECTID avar

#378 '.'

#378 OBJECTID value

#378 '('

#378 ')'

#378 ')'

#378 ELSE

#380 IF

#380 OBJECTID char

#380 '='

#380 STR\_CONST "g"

#380 THEN

#381 IF

#381 '('

#381 '('

#381 NEW

#381 TYPEID D

#381 ')'

#381 '.'

#381 OBJECTID method7

#381 '('

#381 OBJECTID avar

#381 '.'

#381 OBJECTID value

#381 '('

#381 ')'

#381 ')'

#381 ')'

#382 THEN

#383 '{'

#384 OBJECTID out\_string

#384 '('

#384 STR\_CONST "number "

#384 ')'

#384 ';'

#385 OBJECTID print

#385 '('

#385 OBJECTID avar

#385 ')'

#385 ';'

#386 OBJECTID out\_string

#386 '('

#386 STR\_CONST "is divisible by 3.\n"

#386 ')'

#386 ';'

#387 '}'

#388 ELSE

#389 '{'

#390 OBJECTID out\_string

#390 '('

#390 STR\_CONST "number "

#390 ')'

#390 ';'

#391 OBJECTID print

#391 '('

#391 OBJECTID avar

#391 ')'

#391 ';'

#392 OBJECTID out\_string

#392 '('

#392 STR\_CONST "is not divisible by 3.\n"

#392 ')'

#392 ';'

#393 '}'

#394 FI

#394 ELSE

#395 IF

#395 OBJECTID char

#395 '='

#395 STR\_CONST "h"

#395 THEN

#396 '('

#396 LET

#396 OBJECTID x

#396 ':'

#396 TYPEID A

#396 IN

#397 '{'

#398 OBJECTID x

#398 ASSIGN

#398 '('

#398 NEW

#398 TYPEID E

#398 ')'

#398 '.'

#398 OBJECTID method6

#398 '('

#398 OBJECTID avar

#398 '.'

#398 OBJECTID value

#398 '('

#398 ')'

#398 ')'

#398 ';'

#399 '('

#399 LET

#399 OBJECTID r

#399 ':'

#399 TYPEID Int

#399 ASSIGN

#399 '('

#399 OBJECTID avar

#399 '.'

#399 OBJECTID value

#399 '('

#399 ')'

#399 '-'

#399 '('

#399 OBJECTID x

#399 '.'

#399 OBJECTID value

#399 '('

#399 ')'

#399 '\*'

#399 INT\_CONST 8

#399 ')'

#399 ')'

#399 IN

#400 '{'

#401 OBJECTID out\_string

#401 '('

#401 STR\_CONST "number "

#401 ')'

#401 ';'

#402 OBJECTID print

#402 '('

#402 OBJECTID avar

#402 ')'

#402 ';'

#403 OBJECTID out\_string

#403 '('

#403 STR\_CONST "is equal to "

#403 ')'

#403 ';'

#404 OBJECTID print

#404 '('

#404 OBJECTID x

#404 ')'

#404 ';'

#405 OBJECTID out\_string

#405 '('

#405 STR\_CONST "times 8 with a remainder of "

#405 ')'

#405 ';'

#406 '('

#406 LET

#406 OBJECTID a

#406 ':'

#406 TYPEID A2I

#406 ASSIGN

#406 NEW

#406 TYPEID A2I

#406 IN

#407 '{'

#408 OBJECTID out\_string

#408 '('

#408 OBJECTID a

#408 '.'

#408 OBJECTID i2a

#408 '('

#408 OBJECTID r

#408 ')'

#408 ')'

#408 ';'

#409 OBJECTID out\_string

#409 '('

#409 STR\_CONST "\n"

#409 ')'

#409 ';'

#410 '}'

#411 ')'

#411 ';'

#412 '}'

#413 ')'

#413 ';'

#414 OBJECTID avar

#414 ASSIGN

#414 OBJECTID x

#414 ';'

#415 '}'

#416 ')'

#417 ELSE

#418 IF

#418 OBJECTID char

#418 '='

#418 STR\_CONST "j"

#418 THEN

#418 OBJECTID avar

#418 ASSIGN

#418 '('

#418 NEW

#418 TYPEID A

#418 ')'

#419 ELSE

#420 IF

#420 OBJECTID char

#420 '='

#420 STR\_CONST "q"

#420 THEN

#420 OBJECTID flag

#420 ASSIGN

#420 BOOL\_CONST false

#421 ELSE

#422 OBJECTID avar

#422 ASSIGN

#422 '('

#422 NEW

#422 TYPEID A

#422 ')'

#422 '.'

#422 OBJECTID method1

#422 '('

#422 OBJECTID avar

#422 '.'

#422 OBJECTID value

#422 '('

#422 ')'

#422 ')'

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 FI

#423 ';'

#424 '}'

#425 POOL

#425 ';'

#426 '}'

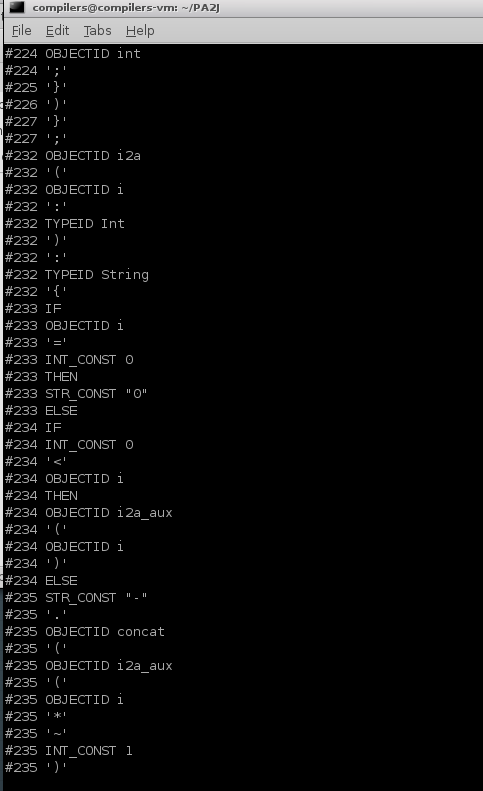
#427 '}'

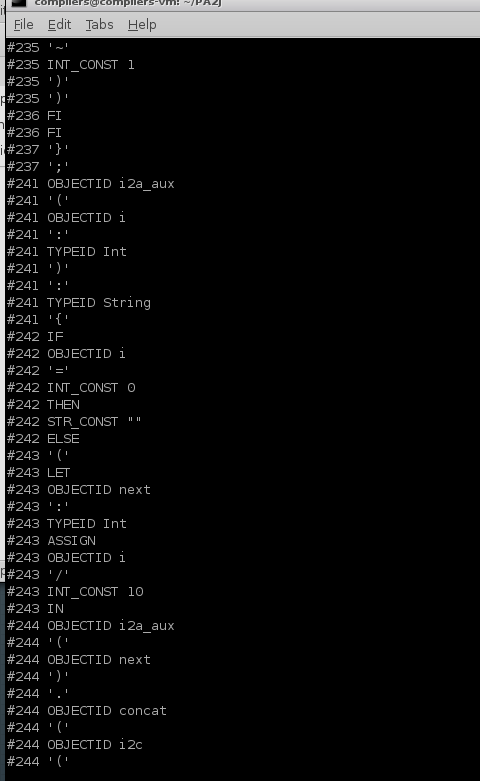
#427 ';'

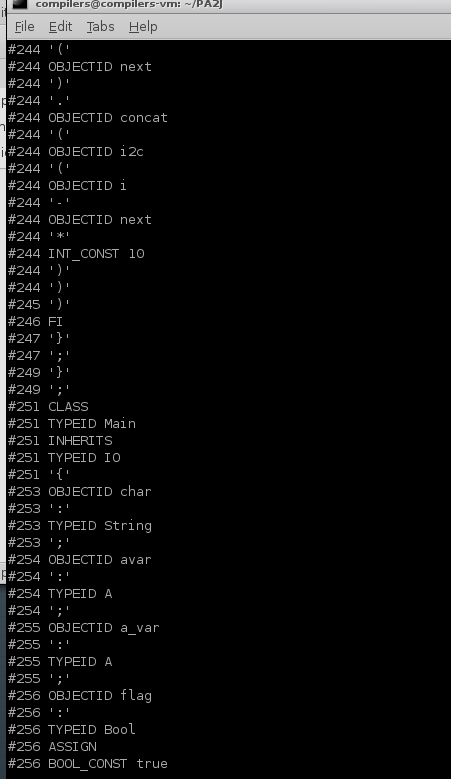
#429 '}'

#429 ';'

**完全符合**









backslash2.cool

"I'm not going to escape this newline

"

"This is also not escaped \\

"

"This is, though \\\

"

"\\"

"It's fine to have this: \n in a string"

#name "backslash2.cool"

#2 ERROR "Unterminated string constant"

#3 ERROR "Unterminated string constant"

#5 ERROR "Unterminated string constant"

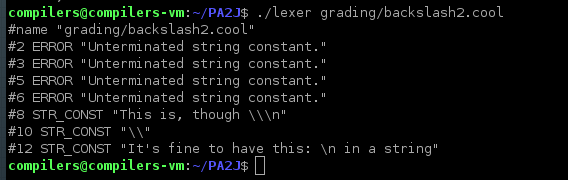
#6 ERROR "Unterminated string constant"

#8 STR\_CONST "This is, though \\\n"

#10 STR\_CONST "\\"

#12 STR\_CONST "It's fine to have this: \n in a string"

**测试完全符合**

**badkeywords.cool**

casee

elsif

form

es;ac

true9

False

fi\_if

inh erits

isS

isv Oid

Loopnew

NOte

ofofofofof

PO8ol

THEN

hile

ifif

IFIF

fif

**#name "badkeywords.cool"**

#1 OBJECTID casee

#2 OBJECTID elsif

#3 OBJECTID form

#4 OBJECTID es

#4 ';'

#4 OBJECTID ac

#5 OBJECTID true9

#6 TYPEID False

#7 OBJECTID fi\_if

#8 OBJECTID inh

#8 OBJECTID erits

#9 OBJECTID isS

#10 OBJECTID isv

#10 TYPEID Oid

#11 TYPEID Loopnew

#12 TYPEID NOte

#13 OBJECTID ofofofofof

#14 TYPEID PO8ol

#15 THEN

#16 OBJECTID hile

#17 OBJECTID ifif

#18 TYPEID IFIF

#19 OBJECTID fif

测试完全符合



**comment\_in\_string.cl.cool**

"string (\* 123 \*) string"

"string \

(\* 123 \*)"

"string (\* \

123 \*)"

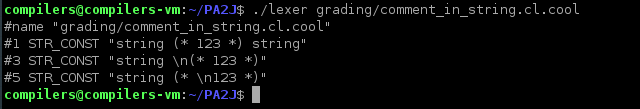
**#name "comment\_in\_string.cl.cool"**

#1 STR\_CONST "string (\* 123 \*) string"

#3 STR\_CONST "string \n(\* 123 \*)"

#5 STR\_CONST "string (\* \n123 \*)"

测试完全符合



**endcomment.cool**

This comment was never begun \*)

**#name "endcomment.cool"**

#1 TYPEID This

#1 OBJECTID comment

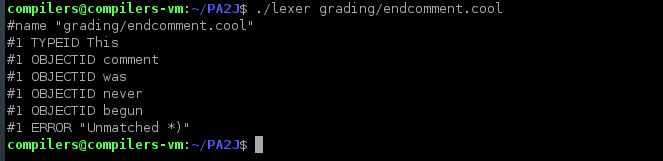
#1 OBJECTID was

#1 OBJECTID never

#1 OBJECTID begun

#1 ERROR "Unmatched \*)"

**测试完全符合**



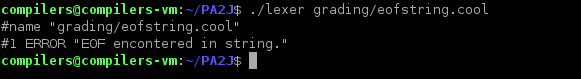
**eofstring.cool**

" there is an EOF coming... wait for it... wait for it…

**#name "eofstring.cool"**

#1 ERROR "EOF in string constant"

**完全符合**

****

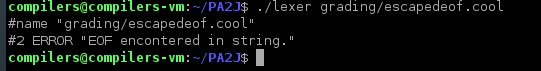
**escapedeof.cool**

"What happens if I escape an EOF? \

**#name "escapedeof.cool"**

#2 ERROR "EOF in string constant"

**完全符合**



**lineno2.cool**

(\* jfkds;ajfkdl;a (\* "

" \*) fdjsk;ajs

\*)

!

fjdk

""

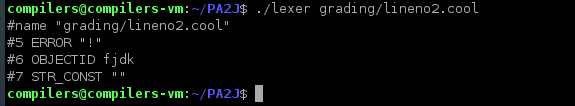
**#name "lineno2.cool"**

#5 ERROR "!"

#6 OBJECTID fjdk

#7 STR\_CONST ""

**测试完全符合**

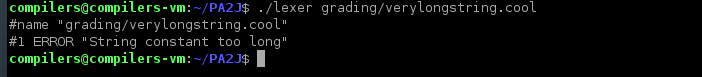
**verylongstring.cool**

"I'm very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very long"

**#name "verylongstring.cool"**

#1 ERROR "String constant too long"

**测试完全符合**

****

**null\_in\_string\_followed\_by\_tokens.cl.cool**

(\* null character in string \*)

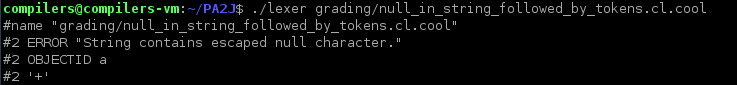
"null character is here -><-" a +

**#name "null\_in\_string\_followed\_by\_tokens.cl.cool"**

#2 ERROR "String contains null character."

#2 OBJECTID a

#2 '+'　ｓ



**6 Conclusion**

At first, I place different rules randomly in different position. For example, I place the identifier rules before the keyword rules, which will make all keyword to be an identifier. Due to those terrible actions, I really messed all things up, and lead to a lots of error when I tried to produce tokens.

Then, when I was testing my lexer, it turned out that it could not handle some cases properly, especially strings. Then I modified the escape characters rules carefully. At last, my lexer seems to work.