

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2016220304022

姓 名 罗悦

(实验) 课程名称 编译技术

理论教师 周尔强

实验教师 周尔强

电子科技大学

实验报告

学生姓名：罗悦 学号：2016220304022 指导教师：周尔强

组长姓名：罗悦 组长学号：2016220304022 组内排序：1

组员姓名：罗悦 组员学号：2016220304022 组内排序：1

实验地点：信软学院楼 304 实验时间：2017. 10. 1

一、 实验室名称：信软 304

二、 实验项目名称：词法分析器

三、 实验学时：4 学时

四、 实验目的：

1. 做一个手动词法分析器，以 `python.py` 作为输入文件，用 C 语言编写程序。

(1) 识别程序中所有的整数、浮点数及字符串常量

(2) 识别程序中所有的符号、标识符及关键字

(3) 将所编写程序命名为 “`man_lex.c`”

2. 学习所提供资料中的简单 `flex` 源程序，试着修改其中的规则及代码，再运行之。

3. 用 `flex` 完成 任务 1，并将所编写 `flex` 源文件命名为 `auto_lex.l`。

五、 实验原理：

手动词法分析器是通过 C 语言实现对所给文档的内容匹配，包括整数、浮点数、字符串常量、符号、标识符和关键字。运用枚举的方法把各个类型分类，并实现最终输出。自动词法分析器的原理是 `Flex` 会自动生成一个“词法分析器”，我们把词法分析器要识别的单词(token)用正则表达式写好，然后作为 `flex` 的输

入文件,输入命令 `flex xxx.l` (`xxx.l` 为输入文件)。flex 经过处理后,就得到一个名字叫 `lex.yy.c` 的 C 源代码。这个 C 源代码文件,就是我们的词法扫描程序/词法分析器。

六、实验内容:

通过对语法熟悉,运用 C 语言巧妙地匹配各个字符,关键字等。并完整代码函数,能运行给出的示例文档,并分析出文档的内容进行匹配。学习并掌握运用 flex,运用 flex 生成词法分析器进行分析给出文档。

七、实验器材 (设备、元器件):

PC 机一台。

八、实验步骤:

(1) 分析给出文档的内容,了解所应该匹配的各个类型的大致内容,并仔细看清楚文档中的细节内容。

(2) 通过 C 语言,运用枚举类型的方法,对各个类型进行分类,包括关键字、字符、整形、浮点型、字符串、标识符等。并完成各个类型的匹配函数。生成手动匹配程序 `man_lex.c`。

(3) 在 Linux 或者 windows 下对 `man_lex.c` 进行 gcc 编译,并运行程序实现对给出文档 `test.scala` 的内容的匹配。

(4) 按照 flex 所要求的格式把 `test.scala` 文档的所需要匹配的内容用正则表达式写好,然后再作为 flex 的输入文件,对生成的 `lex.yy.c` 文件进行 gcc 编译,然后再在 linux 或者 windows 下运行程序给出对 `test.scala` 的内容的匹配。

九、实验结果与分析(含重要数据结果分析或核心代码流程分析)

```
vlu@ubuntu: ~/Desktop
vlu@ubuntu:~$ cd Desktop/
vlu@ubuntu:~/Desktop$ gcc -g man_lex.c -o man
vlu@ubuntu:~/Desktop$ ./man test.scala
```

(1) 对 man_lex.c 文件进行 gcc 编译，并运行。（如上图所示）

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

int charClass;
#define MAX_LEN 10000
char lexeme[MAX_LEN];
char nextChar;
char next2Char;
int lexLen;
int token;
int nextToken;
FILE *inFile;

#define LETTER 0
#define DIGIT 1
#define UNKNOWN 999

enum
{ABSTRACT=258,CASE,CATCH,CLASS,DEF,DO,ELSE,EXTENDS,FALSE1,FINAL,FINALLY
, FOR,FORSOME,IF,IMPLICIT,IMPORT,LAZY,MACRO,MATCH,NEW,NULL1,OBJECT,OVERR
IDE,PACKAGE,PRIVATE,PROTECTED,RETURN,SEALED,SUPER,THIS,THROW,TRAIT,TRY,
TRUE1,TYPE,VAL,VAR,WHILE,WITH,YIELD,ERROR,INT,FLOAT,ID,NOTE1,NOTE2,STRI
NG,SIGN1,SIGN2,SIGN3,SIGN4,EQU,AEQU,LEQU,GEQU,MEQU,KEQU,NEQU};
char
*keywords[]={ "abstract", "case", "catch", "class", "def", "do", "else", "exten
ds", "false", "final", "finally", "for", "forsome", "if", "implicit", "import",
"lazy", "macro", "match", "new", "null", "object", "override", "package", "priv
ate", "protected", "return", "sealed", "super", "this", "throw", "trait", "try"
, "true", "type", "val", "var", "while", "with", "yield", "error", "int", "id", 0}
```

```
;
```

(2) 把各个关键字、字符、浮点型(FLOAT)、字符串(STRING)、整形(INT)等都定义一个相应枚举类型。(如上代码)

```
void addChar()
{
    if(lexLen<=MAX_LEN-2)
    {
        lexeme[lexLen++]=nextChar;
        lexeme[lexLen]=0;
    }
    else
        printf("ERROR:lexeme is too long.\n");
}
//将 nextChar 中的字符放入数组 lexeme 中
```

(3) 将 nextChar 中的字符放入到 lexeme 中。(如上代码)

```
void getChar()
{
    static int firstRun=1;
    if(firstRun)
    {
        nextChar=getc(inFile);
        next2Char=getc(inFile);
        firstRun=0;
    }
    else
    {
        nextChar=next2Char;
        next2Char=getc(inFile);
    }

    if(nextChar==EOF)
    {
        charClass=EOF;
    }
    else
    {
        if(isalpha(nextChar))
            charClass=LETTER;
```

```
        else if(isdigit(nextChar))
            charClass=DIGIT;
        else
            charClass=UNKNOWN;
    }
}
//读取一个字符并放入 nextChar, 并判断字符的类型
```

(4) 读取一个字符并放入 nextChar, 并判断字符的类型。(如上代码)

```
void getNonBlank()
{
    while(isspace(nextChar))
        getChar();
}
//读取空格
```

(5) 读取空格。(如上代码)

```
int checkSymbol(char ch,char nextCh)
{
    switch(ch)
    {
        case ';':
        case '+':
        case '-':
        case '{':
        case '}':
        case '[':
        case ']':
        case '&':
        case '.':
        case '(':
        case ')':
        case '\\':
            addChar();
            nextToken=ch;
            break;
        case '_':
            addChar();
```

```

    nextToken=SIGN1;
    break;
case ':':
    addChar();
    nextToken=SIGN2;
    break;
case '#':
    addChar();
    nextToken=SIGN3;
    break;
case '@':
    addChar();
    nextToken=SIGN4;
    break;
case '=':
    addChar();
    nextToken=ch;
    if(nextCh=='=')
    {
        getChar();
        addChar();
        nextToken=AEQU;
    }
    else if(nextCh=='>')
    {
        getChar();
        addChar();
        nextToken=LEQU;
    }
    break;
case '>':
    addChar();
    nextToken=ch;
    if(nextCh==':')
    {
        getChar();
        addChar();
        nextToken=GEQU;
    }
    break;
case '<':
    addChar();
    nextToken=ch;
    if(nextCh=='-')

```

```

        {
            getChar();
            addChar();
            nextToken=MEQU;
        }
        if(nextCh==':')
        {
            getChar();
            addChar();
            nextToken=KEQU;
        }
        if(nextCh=='%')
        {
            getChar();
            addChar();
            nextToken=NEQU;
        }
        break;
    case EOF:
        addChar();
        nextToken=EOF;
    default:
        printf("ERROR:unknown character '%c'.\n",ch);
        nextToken=ERROR;
    }
    return nextToken;
}

void checkKeywords(char *pword)
{
    int i=0;
    while(keywords[i]!=0)
    {
        char *pkeyword=keywords[i];
        if(strcmp(pword,pkeyword)==0)
        {
            nextToken=258+i;
            return;
        }
        i++;
    }
}
//识别关键字

```



```
vlu@ubuntu: ~/Desktop
val      :      =      =>      <-      <:      <%      >:      #      @
_

*/ >
< 279, object >
< 301, FileMatcher >
< 123, { >
< 282, private >
< 262, def >
< 301, filesHere >
< 61, = >
< 40, ( >
< 277, new >
< 301, java >
< 46, . >
< 301, io >
< 46, . >
< 301, File >
< 40, ( >
< 304, "." >
< 41, ) >
< 41, ) >
< 46, . >
< 301, listFiles >
< 282, private >
< 262, def >
< 301, filesMatching >
< 40, ( >
< 301, matcher >
< 306, : >
< 301, String >
< 311, => >
< 301, Boolean >
< 41, ) >
```

```
vlu@ubuntu: ~/Desktop
< 311, => >
< 301, Boolean >
< 41, ) >
< 61, = >
< 269, for >
< 40, ( >
< 301, file >
< 313, <- >
< 301, filesHere >
< 59, ; >
< 271, if >
< 301, matcher >
< 40, ( >
< 301, file >
< 46, . >
< 301, getName >
< 41, ) >
< 41, ) >
< 297, yield >
< 301, file >
< 262, def >
< 301, filesEnding >
< 40, ( >
< 301, query >
< 306, : >
< 301, String >
< 41, ) >
< 61, = >
< 301, filesMatching >
< 40, ( >
< 305, - >
< 46, . >
< 301, endsWith >
< 40, ( >
< 301, query >
```

```
vlu@ubuntu: ~/Desktop
< 301, endsWith >
< 40, ( >
< 301, query >
< 41, ) >
< 41, ) >
< 262, def >
< 301, filesContaining >
< 40, ( >
< 301, query >
< 306, : >
< 301, String >
< 41, ) >
< 61, = >
< 301, filesMatching >
< 40, ( >
< 305, - >
< 46, . >
< 301, contains >
< 40, ( >
< 301, query >
< 41, ) >
< 41, ) >
< 262, def >
< 301, filesRegex >
< 40, ( >
< 301, query >
< 306, : >
< 301, String >
< 41, ) >
< 61, = >
< 301, filesMatching >
< 40, ( >
< 305, - >
< 46, . >
< 301, matches >
```

(5) 匹配字符并输出他们的 ASCLL 码，和一些特殊符号如“=>”和关键字所对应的枚举类型。(如上代码和截图所示)

```
void notewords()
{
    lexLen=0;
    getNonBlank();
    if(nextChar=='/')
    {
        addChar();
        getChar();
        if(nextChar=='*')
        {
            addChar();
            getChar();
        }
    }
}
```

```

while(nextChar!='*' || next2Char!='/')
{
    addChar();
    getChar();
}
addChar();
getChar();
addChar();
getChar();

                                nextToken=NOTE1;
}

                                //读取注释
else if(nextChar=='/')
{
    while(nextChar!='\n')
    {
        addChar();
        getChar();
    }
    getChar();

                                nextToken=NOTE2;
}

                                //读取“中文注释”
printf("<%6d,  %s  >\n",nextToken,lexeme);
}
}

```

```
vlu@ubuntu: ~/Desktop
vlu@ubuntu:~$ cd Desktop/
vlu@ubuntu:~/Desktop$ gcc -g man_lex.c
vlu@ubuntu:~/Desktop$ ./man test.scala
< 302, /*
* Copyright (C) 2007-2008 Artima, Inc. All rights reserved.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*
* Example code from:
*
* Programming in Scala (First Edition, Version 6)
* by Martin Odersky, Lex Spoon, Bill Venners
*
* http://booksites.artima.com/programming_in_scala
abstract      case      catch      class      def
do            else      extends    false     final
finally      for        forSome   if         implicit
import      lazy      macro     match     new
null         object    override package private
protected   return    sealed   super     this
throw       trait     try      true     type
val         var        while    with      yield
_          :          =        =>      <-      <:      <%      >:      #      @

val          var        while      with      yield
_           :          =        =>      <-      <:      <%      >:      #      @

*/ >

< 303, //这里是中文注释 >
```

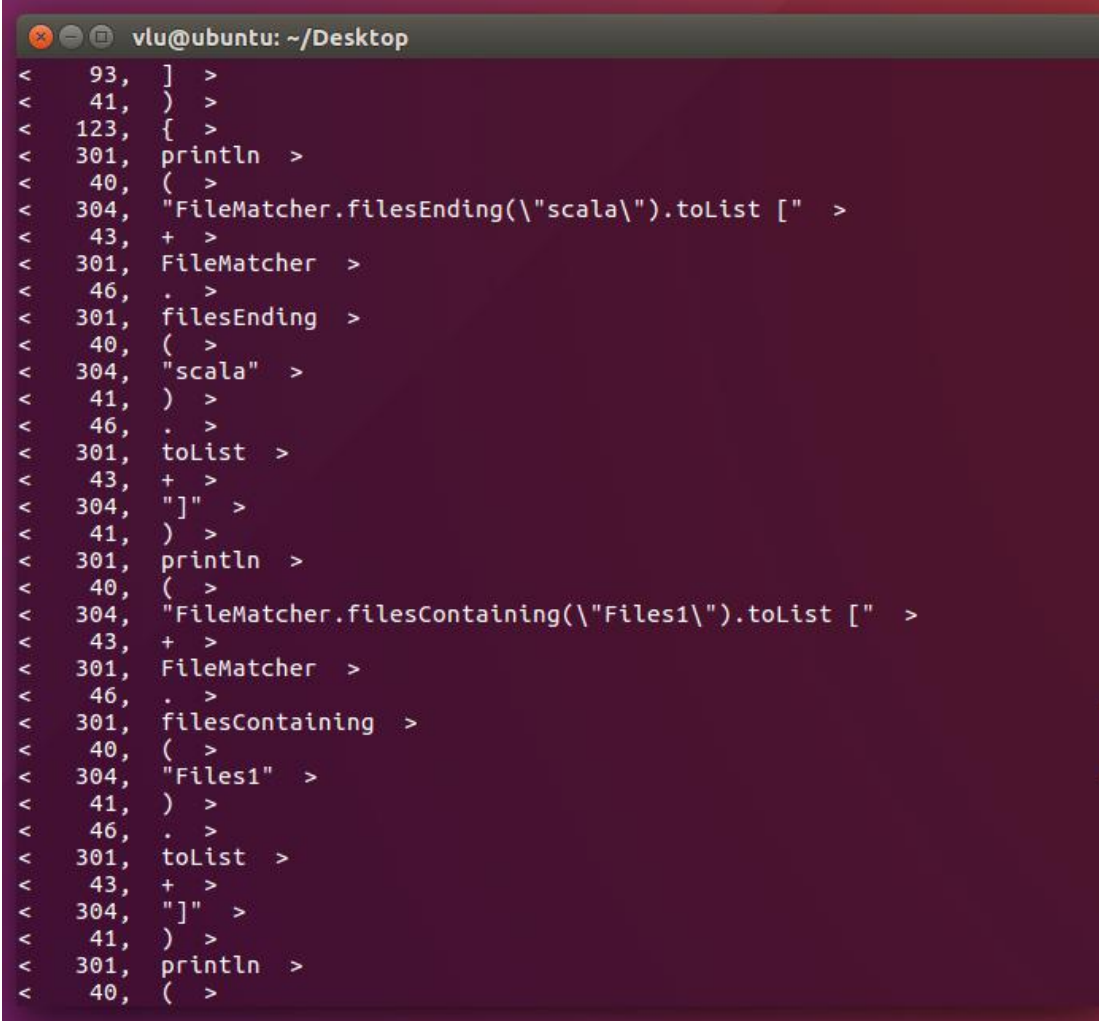
(6) 匹配 test.scala 中的以 “/*” 和 “*/” 表示的注释，并且匹配以 “//” 表示的注释。（如上代码和实验图）

```
void stringwords()
{ lexLen=0;
  if(nextChar=='')
  {
    addChar();
```

```

    getChar();
    while(!(nextChar!='\\'&&next2Char=="")){
        addChar();
        getChar();
    }
    addChar();
    getChar();
    addChar();
    nextToken=STRING;
    printf("<%6d,  %s  >\n",nextToken,lexeme);
    getChar();
}
}

```



The screenshot shows a terminal window with the title 'vlu@ubuntu: ~/Desktop'. It displays the output of a Scala REPL session. The output consists of several lines of text, each preceded by a line number and a prompt character. The text shows the execution of code that uses `FileMatcher` to find files ending with `.scala` and files containing `Files1`. The output is as follows:

```

< 93, ] >
< 41, ) >
< 123, { >
< 301, println >
< 40, ( >
< 304, "FileMatcher.filesEnding(\"scala\").toList [" >
< 43, + >
< 301, FileMatcher >
< 46, . >
< 301, filesEnding >
< 40, ( >
< 304, "scala" >
< 41, ) >
< 46, . >
< 301, toList >
< 43, + >
< 304, "]" >
< 41, ) >
< 301, println >
< 40, ( >
< 304, "FileMatcher.filesContaining(\"Files1\").toList [" >
< 43, + >
< 301, FileMatcher >
< 46, . >
< 301, filesContaining >
< 40, ( >
< 304, "Files1" >
< 41, ) >
< 46, . >
< 301, toList >
< 43, + >
< 304, "]" >
< 41, ) >
< 301, println >
< 40, ( >

```

(7) 匹配 test.scala 中的字符串。(如上代码和实验图所

示)

```
int lexer()
{
    lexLen=0;
    getNonBlank();
    switch(charClass)
    {
        case LETTER:
            addChar();
            getChar();
            while(charClass==LETTER||charClass==DIGIT)
            {
                addChar();
                getChar();
            }
            nextToken=ID;
            checkKeywords(lexeme);
            break;
        case DIGIT:
            addChar();
            getChar();
            if(nextChar=='.')
            {
                addChar();
                getChar();
                while(charClass==DIGIT)
                {
                    addChar();
                    getChar();
                }
                nextToken=FLOAT;
                break;
            }
            while(charClass==DIGIT)
            {
                addChar();
                getChar();
            }
            nextToken=INT;
            break;
        case UNKNOWN:
            checkSymbol(nextChar,next2Char);
            getChar();
    }
```

```

        break;
    case EOF:
        nextToken=EOF;
        lexeme[0]='E';
        lexeme[1]='O';
        lexeme[2]='F';
        lexeme[3]=0;
        break;
    }
    printf("<%6d, %s >\n",nextToken,lexeme);
    return nextToken;
}

```

< 301, Int >

(8) 识别 test.scala 中的读入的下一个字符，如果是整形就返回 Int，如果浮点型就返回 FLOAT，如果是其他类型，就跳入其他函数进行检查。

```

/* recognize tokens for the calculator and print them out */
%{
enum
{ABSTRACT=258,CASE,CATCH,CLASS,DEF,DO,ELSE,EXTENDS,FALSE1,FINAL,FINALLY
, FOR,FORSOME,IF,IMPLICIT,IMPORT,LAZY,MACRO,MATCH,NEW,NULL1,OBJECT,OVERR
IDE,PACKAGE,PRIVATE,PROTECTED,RETURN,SEALED,SUPER,THIS,THROW,TRAIT,TRY,
TRUE1,TYPE,VAL,VAR,WHILE,WITH,YIELD,ERROR,INT,FLOAT,ID,NOTE1,NOTE2,STRIN
G,SIGN1,SIGN2,SIGN3,SIGN4,EQU,AEQU,LEQU,GEQU,MEQU,KEQU,NEQU,OSIGN};

int yylval;
}%

%%

([\n]|[\t]|[ ])* {}
[1-9]*[0-9]* {return INT;}
[1-9]*[0-9]*"."[0-9]* {return FLOAT;}
"\"([^\n^"]|[\\""])*\" {return STRING;}

"/"([^*]*|((\*)+[/]))*\"/ {return NOTE1;}
\\\/[^\n]* {return NOTE2;}
\_ { return SIGN1; }
\: { return SIGN2; }

```



```
\# { return SIGN3; }
\@ { return SIGN4; }
\= { return EQU; }
[=][=] {return AEQU;}
[=][>] { return LEQU; }
[<][-] { return MEQU; }
[<][:] { return KEQU; }
[<][%] { return NEQU; }
[>][:] { return GEQU; }
abstract {return ABSTRACT;}
case {return CASE;}
catch {return CATCH;}
class {return CLASS;}
def {return DEF;}
do {return DO;}
else {return ELSE;}
extends {return EXTENDS;}
false {return FALSE1;}
final {return FINAL;}
for {return FOR;}
finally {return FINALLY;}
forSome {return FORSOME;}
if {return IF;}
implicit {return IMPLICIT;}
import {return IMPORT;}
lazy {return LAZY;}
macro {return MACRO;}
match {return MATCH;}
new {return NEW;}
null {return NULL1;}
object {return OBJECT;}
override {return OVERRIDE;}
package {return PACKAGE;}
private {return PRIVATE;}
protected {return PROTECTED;}
return {return RETURN;}
sealed {return SEALED;}
super {return SUPER;}
this {return THIS;}
throw {return THROW;}
trait {return TRAIT;}
try {return TRY;}
true {return TRUE1;}
type {return TYPE;}
```



```
vlu@ubuntu: ~/Desktop
< 125, } >
< -1, EOF >
vlu@ubuntu:~/Desktop$ flex auto_lex.l
vlu@ubuntu:~/Desktop$ gcc -o scan lex.yy.c
vlu@ubuntu:~/Desktop$ ./scan test.scala
302,/*
* Copyright (C) 2007-2008 Artima, Inc. All rights reserved.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*
* Example code from:
*
* Programming in Scala (First Edition, Version 6)
* by Martin Odersky, Lex Spoon, Bill Venners
*
* http://booksites.artima.com/programming_in_scala
*
abstract    case    catch    class    def
do           else    extends  false   final
finally     for      forSome  if      implicit
import      lazy     macro    match   new
null        object   override package private
protected  return    sealed  super   this
throw       trait    try     true    type
val         var      while   with    yield
```

(匹配注释)

```
throw      trait      try      true      type
val        var        while    with      yield
_          :          =        =>        <-        <:        <%      >:      #        @

*/
279,object
301,FileMatcher
316,{
282,private
262,def
301,filesHere
309,=
316,(
277,new
301,java
300,.
301,io
300,.
301,File
316,(
304,"."
316,)
316,)
300,.
301,listFiles
282,private
262,def
301,filesMatching
316,(
301,matcher
306,:
301,String
311,=>
301,Boolean
```

(匹配字符、关键字)

```
vlu@ubuntu: ~/Desktop
301,nums
306,:
301,List
316,[
301,Int
316,]
316,)
306,:
301,Boolean
309,=
316,{
294,var
301,exists
309,=
266,false
269,for
316,(
301,num
313,<-
301,nums
316,)
271,if
316,(
301,num
316,<
299,0
316,)
301,exists
309,=
291,true
301,exists
316,}
303,///这里是中文注释
279,object
301,Files
```

(匹配字符、关键字、整形、注释)

```
vlu@ubuntu: ~/Desktop
301,println
316,(
304,"FileMatcher.filesEnding(\"scala\").toList ["
316,+
301,FileMatcher
300,.
301,filesEnding
316,(
304,"scala"
316,)
300,.
301,toList
316,+
304,"]"
316,)
301,println
316,(
304,"FileMatcher.filesContaining(\"Files1\").toList ["
316,+
301,FileMatcher
300,.
301,filesContaining
316,(
304,"Files1"
316,)
300,.
301,toList
316,+
304,"]"
316,)
301,println
316,(
304,"FileMatcher.filesRegex(\".*Re.ex.*\").toList ["
316,+
301,FileMatcher
```

(匹配字符串、关键字、字符)

```
vlu@ubuntu: ~/Desktop
309,=
301,x
316,}
262,def
301,minute
309,=
301,m
262,def
301,minute
305,_
309,=
316,(
301,x
306,:
301,Int
316,)
316,{
301,require
316,(
299,0
316,<
309,=
301,x
316,&
316,&
301,x
316,<
299,60
316,)
301,m
309,=
301,x
316,}
316,}
vlu@ubuntu:~/Desktop$
```

（匹配字符、关键字）

（10）用 flex 输入要匹配的文档 test.scala，并用 gcc 编译生成的 lex.yy.c 文件，并运行。得到各个类型的输出。（代码与截图展示如上）

十、总结及心得体会：

运用手动程序来实现对文档的内容匹配时要书写许多的匹配各种类型的函数，并注意指针的溢出问题，并且在类型的匹配函数中要灵活的运用递归，这样可以使函数本身变得简单易懂，并且不容易出错。还有在匹配字符串的时候会出现转移字符“\”，需要运用 while 函数确定循环条件来匹配到正确的字符串。总得

来说，手动识别程序书写较为复杂，容易出错。对于 flex 自动生成的词法分析器来说，就更加方便快捷，只需要写出简单的各个类型的正则表达式，然后按照 flex 格式书写出来，作为 flex 的输入文件，然后进行编译就能运行，这样的话比较简单清晰，而且如果出现了匹配错误也会很清晰的知道哪里有些错误，也方便进行修改。总得来说，flex 方法比手动生成程序更加简单易懂，能更快的实现我们需要的词法分析器。

十一、对本实验过程及方法、手段的改进建议：

对于手动分析函数中，运用了许多的 while 函数，但是这样的判断方式比较复杂，如果能灵活的运用递归调用函数的方法的话，可能会使函数变得更加简单清晰一点。Linux 下的命令行有一个快捷下载 flex 的方法，如果能灵活运用，也会大大提高实验效率。

报告评分：

指导教师签字：