Atraya Mukherjee
Raveena Jadhav

1. Methods

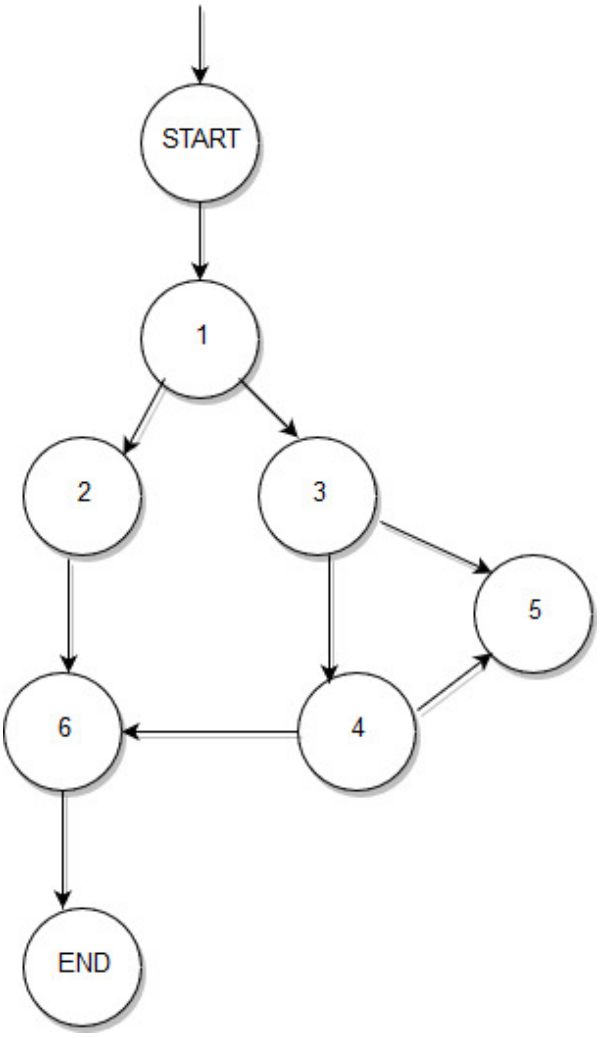
1.BufferedReader open_character_stream(String fname)	2. int get_char(BufferedReader br)
3. char unget_char (int ch,BufferedReader br)	4. BufferedReader open_token_stream(String fname)
5. String get_token(BufferedReader br)	6. static boolean is_token_end(int str_com_id, int res)
7. static int token_type(String tok)	8. void print_token(String tok)
9. static boolean is_comment(String ident)	10. static boolean is_keyword(String str)
11. static boolean is_char_constant(String str)	12. static boolean is_num_constant(String str)
13. static boolean is_str_constant(String str)	14. static boolean is_identifier(String str)
15. static void unget_error(BufferedReader br)	16. static void print_spec_symbol(String str)
17. static boolean is_spec_symbol(char c)	18.public static void main(String[] args) throws IOException

1. Open_character_stream

```
/* NAME:   open_character_stream        */
/* INPUT:      a filename               */
/* OUTPUT:     a BufferedReader */
/* DESCRIPTION: when not given a filename,     */
/*          open stdin,otherwise open      */
/*          the existed file           */
```

```
28      BufferedReader open_character_stream(String fname) {
29          BufferedReader br = null;
30          if (fname == null) {
31              br = new BufferedReader(new InputStreamReader(System.in));
32          } else {
33              try {
34                  FileReader fr = new FileReader(fname);
35                  br = new BufferedReader(fr);
36              } catch (FileNotFoundException e) {
37                  System.out.print("The file " + fname +" doesn't exists\n");
38                  e.printStackTrace();
39              }
40          }
41          return br;
42      }
```

Nodes	Lines
1	28, 29, 30
2	31
3	32, 33, 34
4	35
5	36, 37, 38, 39
6	41, 42



Edge Coverage
[(1,2,6),(1,3,4),(1,3,5), (1,3,4,5)]

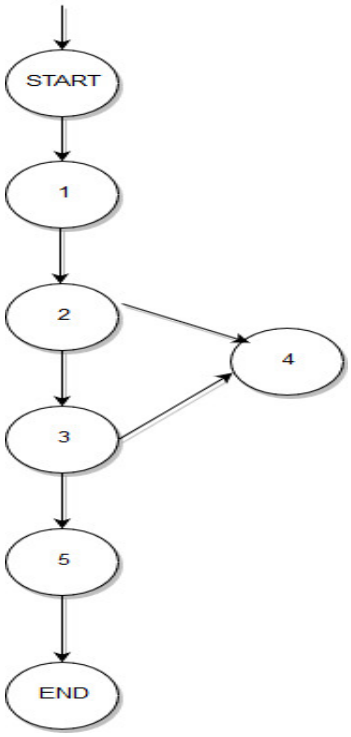
Test data	Test Path
null	1, 2, 6
“”	1, 3 5
“file”	1, 3, 4, 5
“file.txt”	1, 3, 4, 6

2. get_char

/ NAME: get_char */*
/ INPUT: a BufferedReader */*
/ OUTPUT: a character */*

```
49      int get_char(BufferedReader br){
50          int ch = 0;
51          try {
52              br.mark(4);
53              ch= br.read();
54          } catch (IOException e) {
55              e.printStackTrace();
56          }
57          return ch;
58      }
```

Nodes	Lines
1	49, 50, 51
2	52
3	53
4	54, 55, 56
5	57



Edge Coverage
[(1,2,3,5),(1,2,4),(1,2,3,4)]

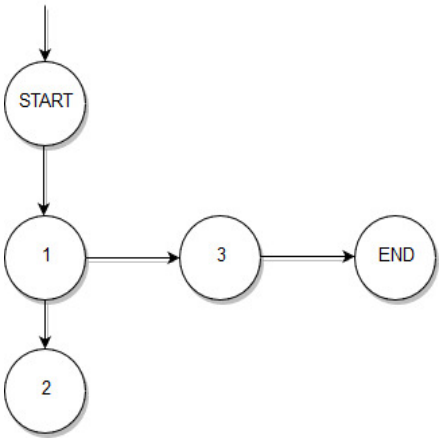
Test data	Test Path
null	1, 2, 3, 5
“”	1, 2,4
br	1, 3, 4

3. unget_char

/ NAME: unget_char */*
/ INPUT: a BufferedReader,a character */*
/ OUTPUT: a character */*
/ DESCRIPTION: move backward */*

```
66      char unget_char (int ch,BufferedReader br) {
67          try {
68              br.reset();
69          } catch (IOException e) {
70              e.printStackTrace();
71          }
72          return 0;
73      }
74
```

Nodes	Lines
1	66, 67,68
2	69, 70, 71
3	72



Edge Coverage
[(1,3),(1,2)]

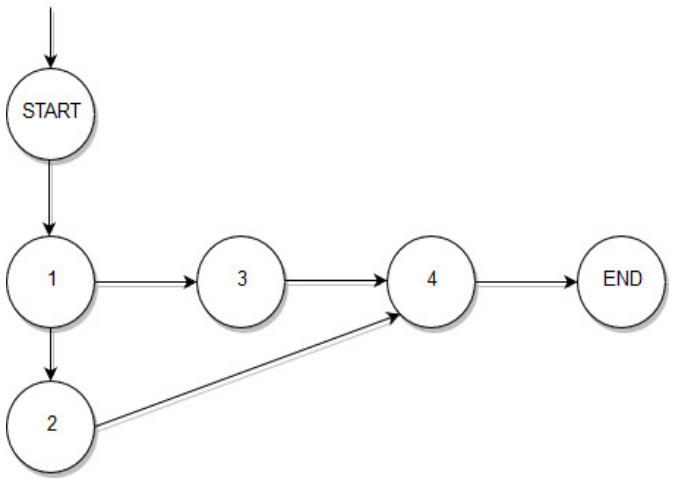
Test data	Test Path
‘a’,br	1, 3
‘a’,null	1, 2

4. open_token_stream

```
/* NAME:       open_token_stream                    */
/* INPUT:      a filename                           */
/* OUTPUT:     a BufferedReader                     */
/* DESCRIPTION: when filename is EMPTY,choice standard  */
/*              input device as input source        */
```

```
82      BufferedReader open_token_stream(String fname)
83      {
84          BufferedReader br;
85          if(fname.equals(null))
86              br=open_character_stream(null);
87          else
88              br=open_character_stream(fname);
89          return br;
90      }
```

Nodes	Lines
1	82, 83, 84
2	85, 86
3	87, 88
4	89, 90



Edge Coverage
[(1,2,4),(1,3,4)]

Test data	Test Path
null	1,2,4
“file.txt”	1,3,4

5. get_token

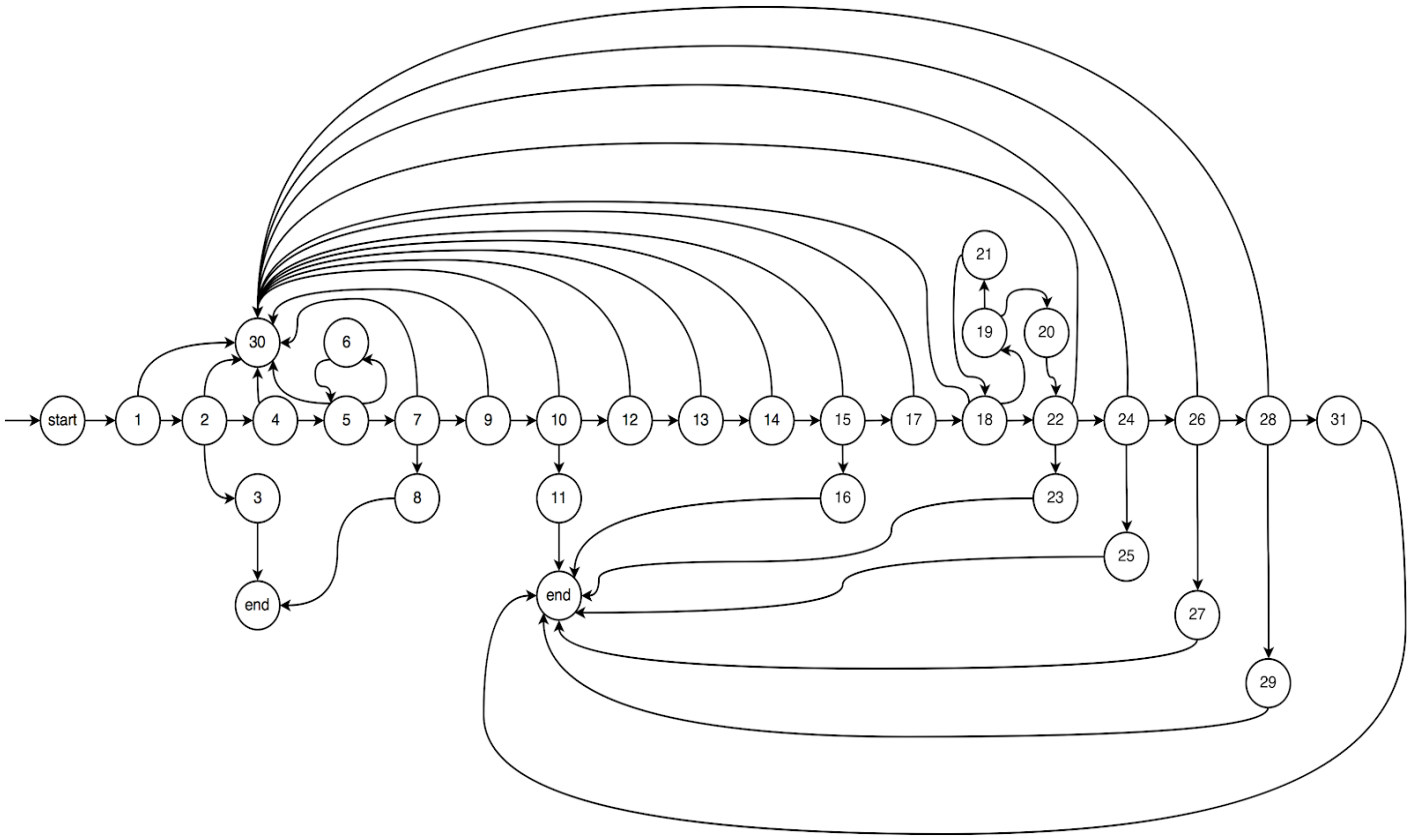
```
/*********************************************************/
/* NAME :  get  token                          */
/* INPUT:  a BufferedReader          */
/* OUTPUT:      a token string                          */
/* DESCRIPTION: according the syntax of tokens,dealing  */
/*              with different case  and get one token  */
  /*********************************************************/
```

```
 99      String get_token(BufferedReader br)
100      {
101          int i=0,j;
102          int id=0;
103          int res = 0;
104          char ch = '\0';
105
106          StringBuilder sb = new StringBuilder();
107
108          try {
109              res = get_char(br);
110              if (res == -1) {
111                  return null;
112              }
113              ch = (char)res;
114              while(ch==' '||ch=='\n' || ch == '\r')      /* strip all blanks until meet characters */
115              {
116                  res = get_char(br);
117                  ch = (char)res;
118              }
119
120              if(res == -1)return null;
121              sb.append(ch);
122              if(is_spec_symbol(ch)==true)return sb.toString();
123              if(ch =='"')id=2;    /* prepare for string */
124              if(ch ==59)id=1;    /* prepare for comment */
125
126              res = get_char(br);
127              if (res == -1) {
128                  unget_char(ch,br);
129                  return sb.toString();
130              }
131              ch = (char)res;
132
133              while (is_token_end(id,res) == false)/* until meet the end character */
134              {
135                  sb.append(ch);
136                  br.mark(4);
137                  res = get_char(br);
138                  if (res == -1) {
139                      break;
140                  }
141                  ch = (char)res;
142              }
143
144              if(res == -1)       /* if end character is eof token    */
145              { unget_char(ch,br);        /* then put back eof on token_stream */
146                  return sb.toString();
147              }
148
149              if(is_spec_symbol(ch)==true)     /* if end character is special_symbol */
150              { unget_char(ch,br);        /* then put back this character       */
151                  return sb.toString();
152              }
153              if(id==1)                   /* if end character is " and is string */
154              {
155                  sb.append(ch);
156                  return sb.toString();
157              }
158              if(id==0 && ch==59)
159                                          /* when not in string or comment,meet ";" */
160              { unget_char(ch,br);        /* then put back this character         */
161                  return sb.toString();
162              }
163          } catch (IOException e) {
164              e.printStackTrace();
165          }
166
167          return sb.toString();                   /* return nomal case token            */
168      }
```


Nodes	Lines
1	99 - 108
2	110
3	111-112

4	113
5	114
6	115-118
7 & 8	120
9	121
10 & 11	122
12	123
13	124
14	125, 126
15	127
16	128, 129
17	131, 132
18	133
19	134-138
20	139
21	141, 142
22	143, 144
23	145, 146
24	148, 149
25	150, 151
26	153
27	154, 155, 156
28	158
29	159, 160, 161
30	163, 164, 165

31	166, 167



Edge Coverage
[(1,30),(1,2,30),(1,2,3),(1,2,4,30), (1,2,4,5,30), (1,2,4,5,7,30), (1,2,4,5,7,8),(1,2,4,5,7,9,30), (1,2,4,5,7,9,10,30), (1,2,4,5,7,9,10,12,30), (1,2,4,5,7,9,10,12,13,30), (1,2,4,5,7,9,10,12,13,14,30), (1,2,4,5,7,9,10,12,13,14,15,30), (1,2,4,5,7,9,10,12,13,14,15,16), (1,2,4,5,7,9,10,12,13,14,15,17,30), (1,2,4,5,7,9,10,12,13,14,15,17,18,30), (1,2,4,5,7,9,10,12,13,14,15,17,18,19,20,22), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,23), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,30), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,25), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,26,30), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,27), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,26,28,30), (1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,26,28,29), (1,2,4,5,6,5,7,9,10,12,13,14,15,17,18,19,21,18,22,24,26,28,31)]

Test data	Test Path
	1,2,3,4,5,6,5,7,9,10,12,13,14,15,17,18,19,21,18,19,20,22,24,26,28,31
	1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,26,28,29
	1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,26,27
	1,2,4,5,7,9,10,12,13,14,15,17,18,22,24,25
	1,2,4,5,7,9,10,12,13,14,15,17,18,22,23
	1,2,4,5,7,9,10,12,13,14,15,16
	1,2,4,5,7,9,10,11
	1,2,4,5,7,8
	1,2,3
	1,30

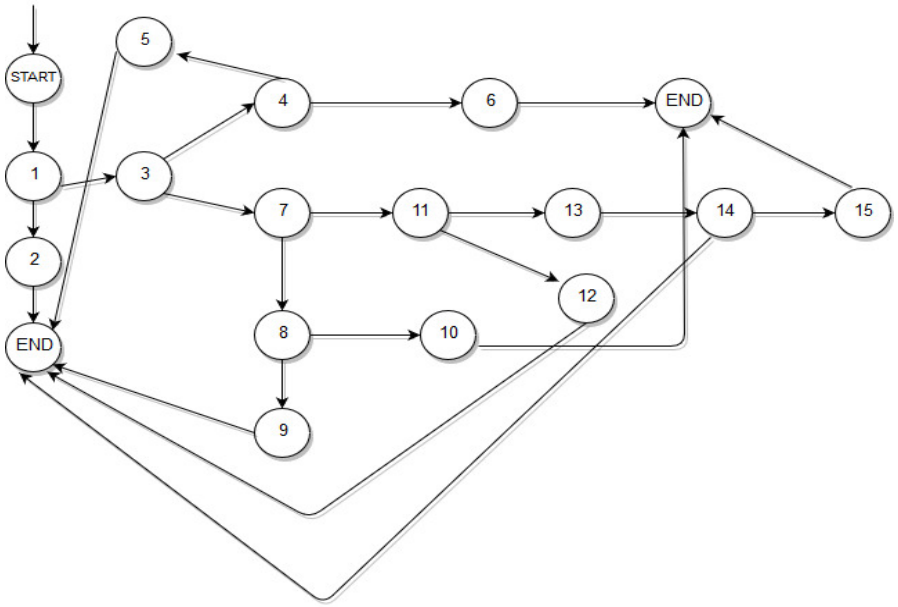
6. is_token_end

```
/*******************************************************/
/* NAME:   is  token  end                    */
/* INPUT:       a character,a token status          */
/* OUTPUT: a BOOLEAN value                   */
  /*******************************************************/
```

```
175 @    static boolean is_token_end(int str_com_id, int res)
176      {
177          if(res==-1)
178              return(true); /* is eof token? */
179          char ch = (char)res;
180          if(str_com_id==1)          /* is string token */
181          {
182              if(ch=='"' | ch=='\n' || ch == '\r')   /* for string until meet another " */
183                  return true;
184              else
185                  return false;
186          }
187
188          if(str_com_id==2)    /* is comment token */
189          {
190              if(ch=='\n' || ch == '\r' || ch=='\t')     /* for comment until meet end of line */
191                  return true;
192              else
193                  return false;
194          }
195
196          if(is_spec_symbol(ch)==true)
197              return true; /* is special_symbol? */
198          if(ch ==' ' || ch=='\n'|| ch=='\r' || ch==59)
199              return true;
200
201          return false;               /* other case,return FALSE */
202      }
```

Nodes	Lines
1	175, 176, 177
2	178
3	179, 180
4	181, 182, 186
5	183
6	184, 185
7	187, 188
8	189, 190
9	191
10	192, 193
11	195, 196
12	197
13	198

14	199
15	200, 201



Edge Coverage
[(1,3,4,6),(1,2),(1,3,4,5),(1,3,7,8,9),(1,3,7,8,10),(1,3,7,11,12),(1,3,7,11,13,14),(1,3,7,11,13,14,16)]

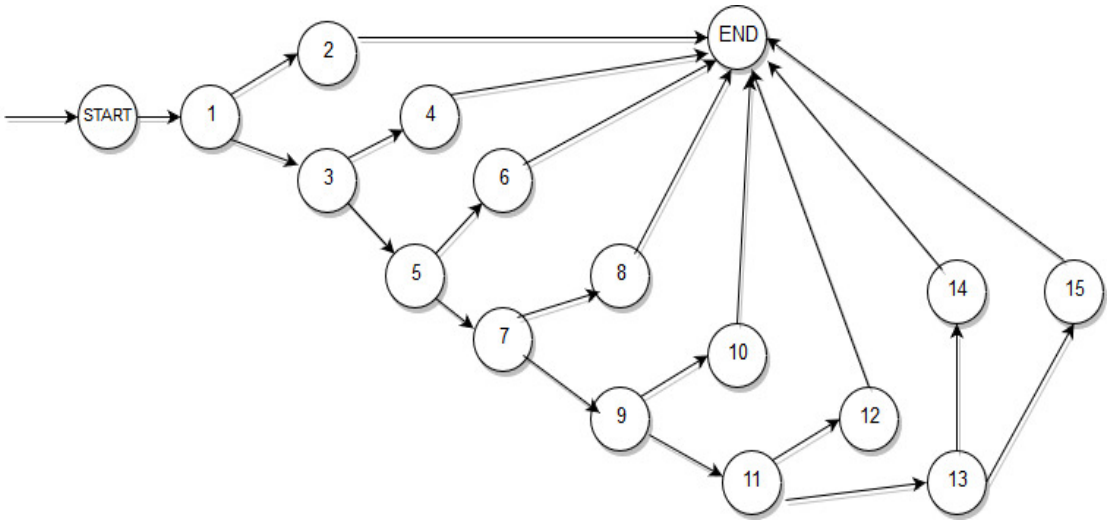
Test data	Test Path
	(1,3,4,6)
	(1,2)
	(1,3,4,5)
	(1,3,7,8,9)
	(1,3,7,8,10)
	(1,3,7,11,12)
	(1,3,7,11,13,14)
	(1,3,7,11,13,14,15)

7. token_type

```
/*****************************************************/
/* NAME :  token  type                    */
/* INPUT:       a token            */
/* OUTPUT:      an integer value                */
/* DESCRIPTION: the integer value is corresponding  */
/*            to the different token type        */
 /*****************************************************/
```

```
211         static int token_type(String tok)
212         {
213             if(is_keyword(tok))return(keyword);
214             if(is_spec_symbol(tok.charAt(0)))return(spec_symbol);
215             if(is_identifier(tok))return(identifier);
216             if(is_num_constant(tok))return(num_constant);
217             if(is_str_constant(tok))return(str_constant);
218             if(is_char_constant(tok))return(char_constant);
219             if(is_comment(tok))return(comment);
220             return(error);                      /* else look as error token */
221         }
```

Nodes	Lines
1	211, 212, 213
2	213
3 and 4	214
5 and 6	215
7 and 8	216
9 and 10	217
11 and 12	218
13 and 14	219
15	220



Edge Coverage
[(1,2),(1,3,4),(1,3,5,6),(1,3,5,7,8),(1,3,5,7,9,10),(1,3,5,7,9,11,12),(1,3,5,7,9,11,13,14),(1,3,5,7,9,11,13,15)]

Test data	Test Path
"and"	(1,2)
"("	(1,3,4)
"is1d"	(1,3,5,6)
"1"	(1,3,5,7,8)
"\"\""	(1,3,5,7,9,10)
"#a"	(1,3,5,7,9,11,12)
"//"	(1,3,5,7,9,11,13,14)
"="	(1,3,5,7,9,11,13,15)

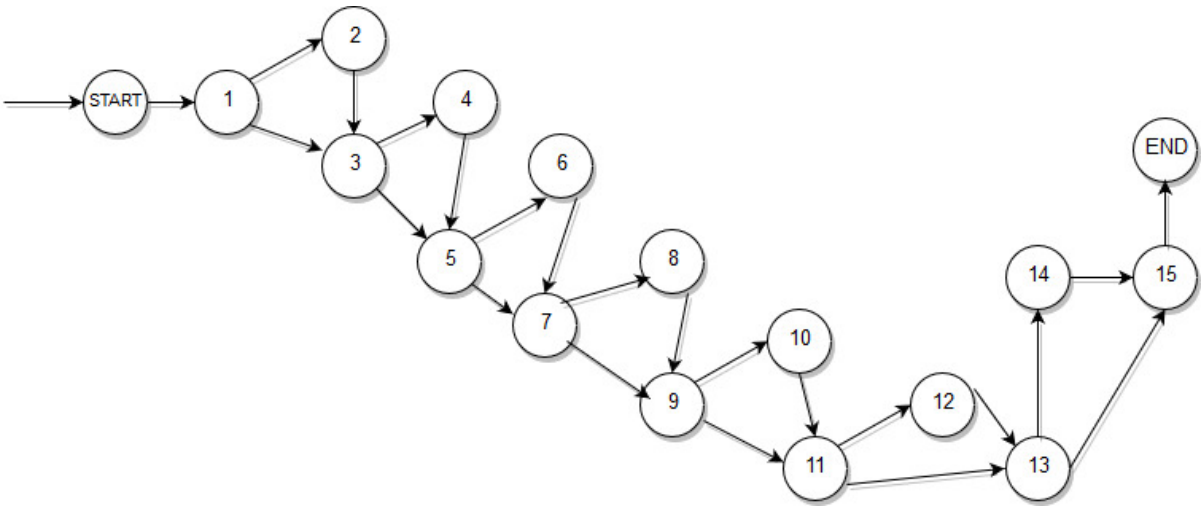
8. print_token

```
/*****************************************************/
/* NAME:  print  token                               */
/* INPUT:  a token                                   */
 /*****************************************************/
```

```
227     void print_token(String tok)
228     { int type;
229         type=token_type(tok);
230         if(type==error)
231         {
232             System.out.print("error,\"" + tok + "\".\n");
233         }
234 
235         if(type==keyword)
236         {
237             System.out.print("keyword,\"" + tok + "\".\n");
238         }
239 
240         if(type==spec_symbol)print_spec_symbol(tok);
241         if(type==identifier)
242         {
243             System.out.print("identifier,\"" + tok + "\".\n");
244         }
245         if(type==num_constant)
246         {
247             System.out.print("numeric," + tok + ".\n");
248         }
249         if(type==str_constant)
250         {
251             System.out.print("string," + tok + ".\n");
252         }
253         if(type==char_constant)
254         {
255             System.out.print("character,\"" + tok.charAt(1) + "\".\n");
256         }
257 
258     }
```

Nodes	Lines
1	227-230
2	231-233
3	234, 235
4	236-238
5	239-240

6	240
7	241
8	242-244
9	245
10	246-248
11	249
12	250-252
13	253
14	254-256
15	257, 258



Edge Coverage
[(1,3,5,7,9,11,13,15),(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)]

Test data	Test Path
"1error"	(1,2,3,5,7,9,11,13,15)
"and"	(1,3,4,5,7,9,11,13)

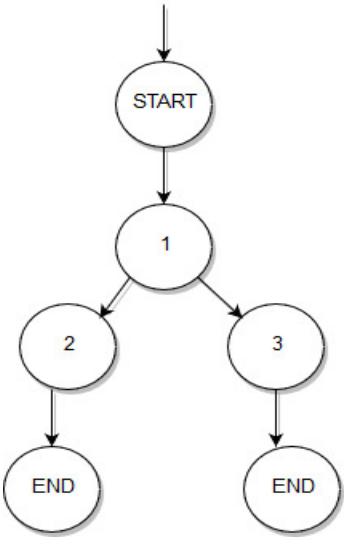
"("	(1,3,5,6,7,9,11,13)
"1"	(1,3,5,7,8,9,11,13)
"identifer"	(1,3,5,7,9,11,13)
"\"String\""	(1,3,5,7,9,10,11,13)
"//comment"	(1,3,5,7,9,11,12,13)
#c	(1,3,5,7,9,11,14,13)

9. is_comment

```
/************************************/
/* NAME:   is_comment          */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
 /************************************/
```

```
268 @       static boolean is_comment(String ident)
269         {
270             if( ident.charAt(0) ==59 )   /* the char is 59   */
271                 return true;
272             else
273                 return false;
274         }
```

Nodes	Lines
1	268-270
2	271
3	272,273



Edge Coverage
[(1,2),(1,3)]

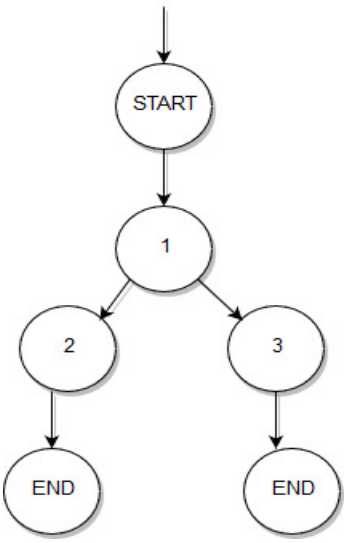
Test data	Test Path
"//"	(1,2)
"-"	(1,3)

10. is _keyword

```
/*************************************/
/* NAME:   is_keyword          */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
 /*************************************/
```

```
281     static boolean is_keyword(String str)
282     {
283         if (str.equals("and") || str.equals("or") || str.equals("if") ||
284                 str.equals("xor")||str.equals("lambda")||str.equals("=>"))
285             return true;
286         else
287             return false;
288     }
```

Nodes	Lines
1	281-284
2	285
3	286-287



Edge Coverage
[(1,2),(1,3)]

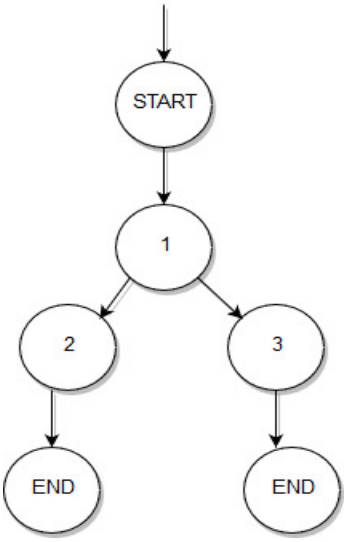
Test data	Test Path
"=>"	(1,2)
"token"	(1,3)

11. is_char _constant

```
/************************************/
/* NAME:  is  char  constant     */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
/************************************/
```

```
297     static boolean is_char_constant(String str)
298     {
299         if (str.length() > 2 && str.charAt(0)=='#' && Character.isLetter(str.charAt(1)))
300             return true;
301         else
302             return false;
303     }
```

Nodes	Lines
1	297-299
2	300
3	301-303



Edge Coverage
[(1,2),(1,3)]

Test data	Test Path

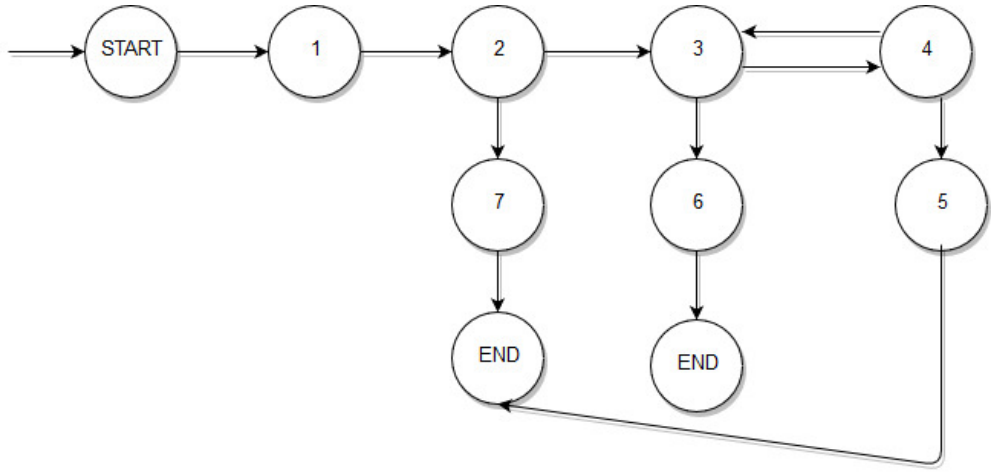
"#a"	(1,2)
"a"	(1,3)

12. is_num_constant

```
/*************************************/
/* NAME:  is_num_constant      */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
/*************************************/
```

```
310      static boolean is_num_constant(String str)
311      {
312          int i=0; // BUG i should be zero
313
314          if ( Character.isDigit(str.charAt(0)))
315          {
316              while ( i <= str.length() && str.charAt(i) != '\0' )   /* until meet token end sign */
317              {
318                  if(Character.isDigit(str.charAt(i))) //BUG
319                      i++;
320                  else
321                      return false;
322              }                          /* end WHILE */
323              return true;
324          }
325          else
326              return false;               /* other return FALSE */
327      }
328
```

Nodes	Lines
1	310-315
2	316-317
3	318
4	319
5	320-321
6	323-324
7	325-327



Edge Coverage
[(1,2,3,4,3,4,5),(1,2,3,6),(1,2,7)]

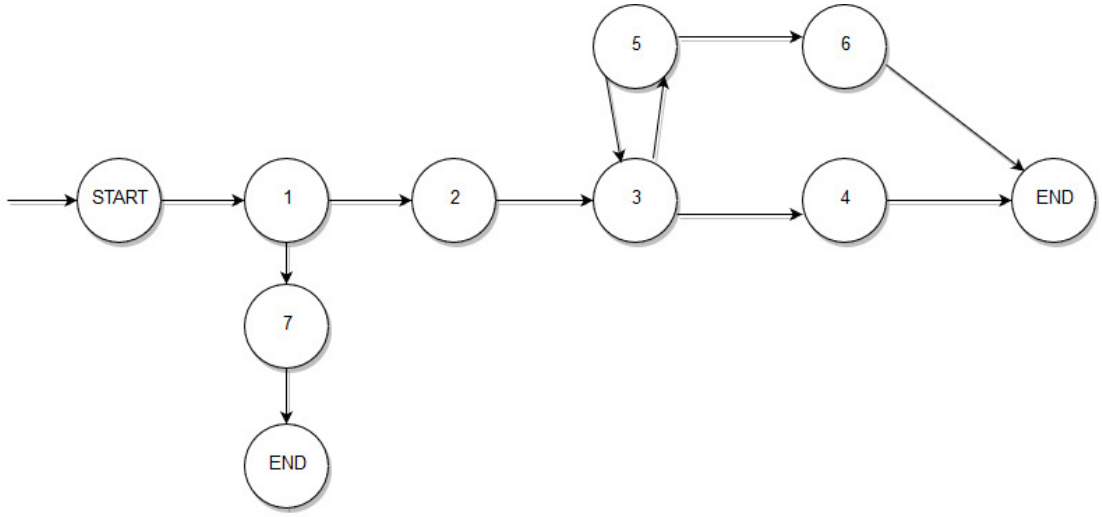
Test data	Test Path
"c"	(1,2,7)
"111111"	(1,2,3,6)
“1c”	(1,2,3,4,3,4,5)

13. Is_str_constant

```
/*************************************/
/* NAME:   is_str_constant      */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
/*************************************/
```

```
334     static boolean is_str_constant(String str)
335     {
336         int i=1;
337
338         if ( str.charAt(0) =='"')
339         { while (i < str.length() && str.charAt(0)!='\0')  /* until meet the token end sign */
340         { if(str.charAt(i)=='"')
341             return true;        /* meet the second '"'          */
342         else
343             i++;
344         }               /* end WHILE */
345             return true;
346         }
347         else
348             return false;       /* other return FALSE */
349     }
```

Nodes	Lines
1	334,335,336,337,338
2	339
3	340
4	341
5	342,343,344
6	345
7	347, 348



Edge Coverage
[(1,2,3,5,3,4),(1,2,3,5,6),(1,7)]

Test data	Test Path
"1"	(1,7)
"\"abc"	(1,2,3,5,6)
"\"\""	(1,2,3,4)

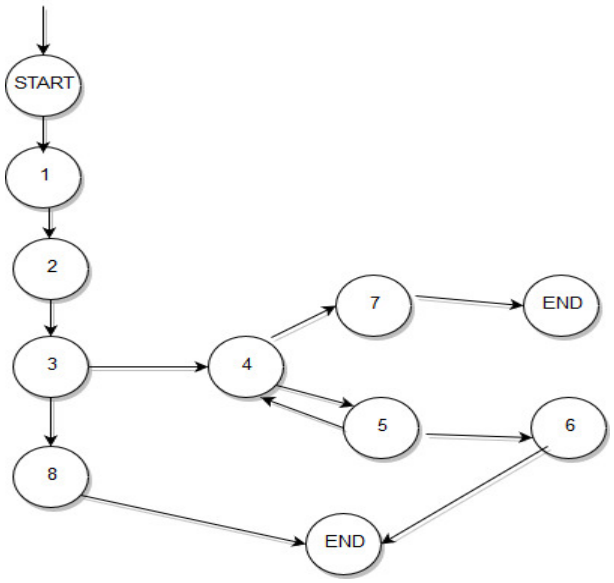
"\"abc\""	(1,2,3,4,5,4)

14. Is_identifier

```
/*************************************/
/* NAME:   is_identifier        */
/* INPUT:  a token */
/* OUTPUT:      a BOOLEAN value      */
/*************************************/
```

```
356      static boolean is_identifier(String str)
357      {
358          int i=0; //bug
359
360          if ( Character.isLetter(str.charAt(0)) )
361          {
362              while(i < str.length() && str.charAt(i) !='\0' )   /* unti meet the end token sign */
363              {
364                  if(Character.isLetter(str.charAt(i)) || Character.isDigit(str.charAt(i)))
365                      i++;
366                  else
367                      return false;
368              }      /* end WHILE */
369              return false;
370          }
371          else
372              return true;
373      }
```

Nodes	Lines
1	356-360
2	361
3	362
4	363, 364
5	365
6	366, 367
7	369
8	371, 372



Edge Coverage
[(1,2,3,4,5,4,5,6),(1,2,3,4,7),(1,2,3,8)]

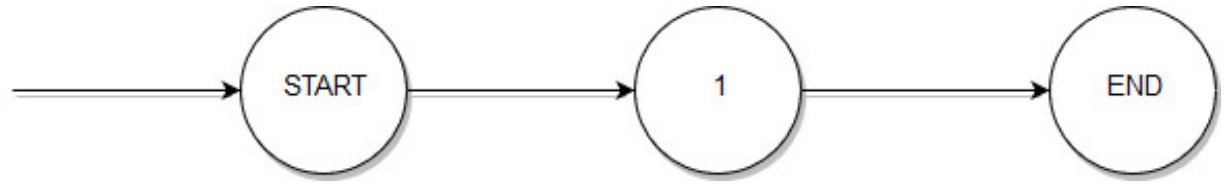
Test data	Test Path
“token”	(1,2,3,4,5,4,5,6)
“is!d”	(1,2,3,4,7)
“1”	(1,2,3,8)

15. Unget_error

```
/*****************************************/
/* NAME:   unget  error              */
/* INPUT:      a BufferedReader */
/* OUTPUT:     print error message       */
/*****************************************/
```

```
380     static void unget_error(BufferedReader br) { System.out.print("It can not get charcter\n"); }
```

Nodes	Lines
1	380



Edge Coverage
[(1)]

Test data	Test Path
br	(1)

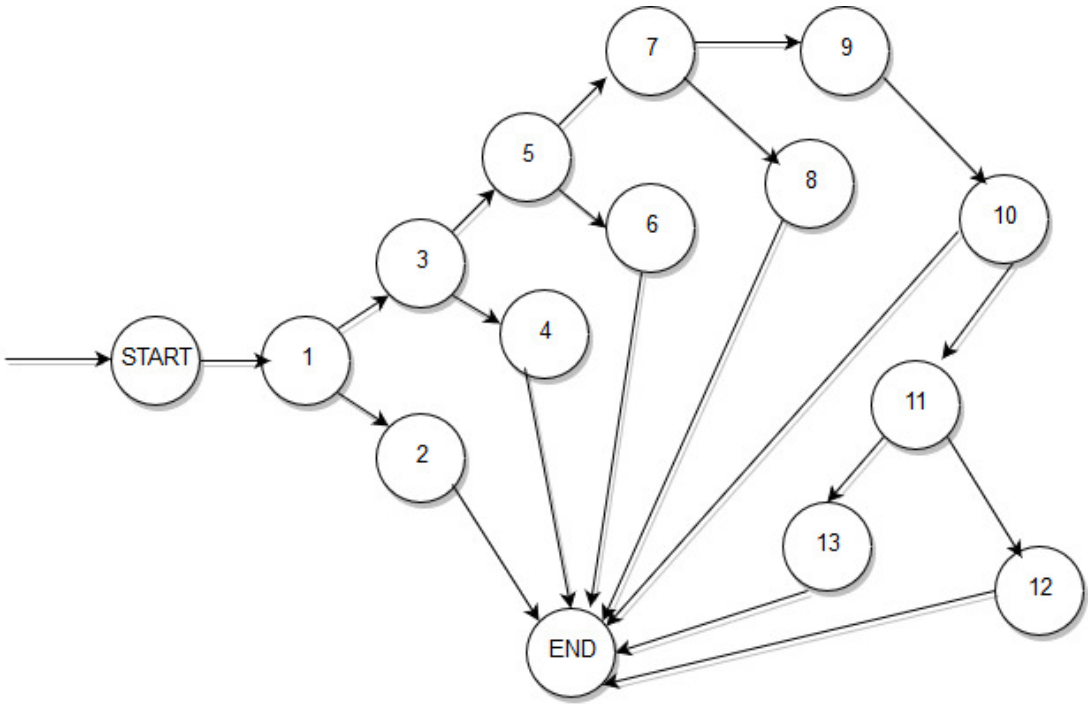
16. print_spec_symbol

```
/*************************************************/
/* NAME:       print_spec_symbol              */
/* INPUT:      a spec_symbol token */
/* OUTPUT :     print out the spec  symbol token  */
/*            according to the form required   */
/*************************************************/
```

```
391     static void print_spec_symbol(String str)
392     {
393         if      (str.equals("{")) // bug fix
394         {
395
396             System.out.print("lparen.\n");
397             return;
398         }
399         if (str.equals(")"))
400         {
401
402             System.out.print("rparen.\n");
403             return;
404         }
405         if (str.equals("["))
406         {
407             System.out.print("lsquare.\n");
408             return;
409         }
410         if (str.equals("]"))
411         {
412
413             System.out.print("rsquare.\n");
414             return;
415         }
416         if (str.equals("'"))
417         {
418             System.out.print("quote.\n");
419             return;
420         }
421         if (str.equals("`"))
422         {
423
424             System.out.print("bquote.\n");
425             return;
426         }
427
428
429     }
```

Nodes	Lines
1	391-393
2	394-397
3	399
4	400-403

5	405
6	406-408
7	410
8	411-414
9	416
10	417-419
11	421
12	422-425
13	427-429



Edge Coverage
[(1,2),(1,3,4),(1,3,5,6),(1,3,5,7,8),[(1,3,5,7,9,10),(1,3,5,7,9,10,11,12),(1,3,5,7,9,10,11,13)]

Test data	Test Path
‘(‘	(1,2)
‘)’	(1,3,4)
‘[’	(1,3,5,6)
‘]’	(1,3,5,7,8)
‘/’’	(1,3,5,7,9,10)
‘`’	(1,3,5,7,9,10,11,12)
‘,’	(1,3,5,7,9,10,11,13)

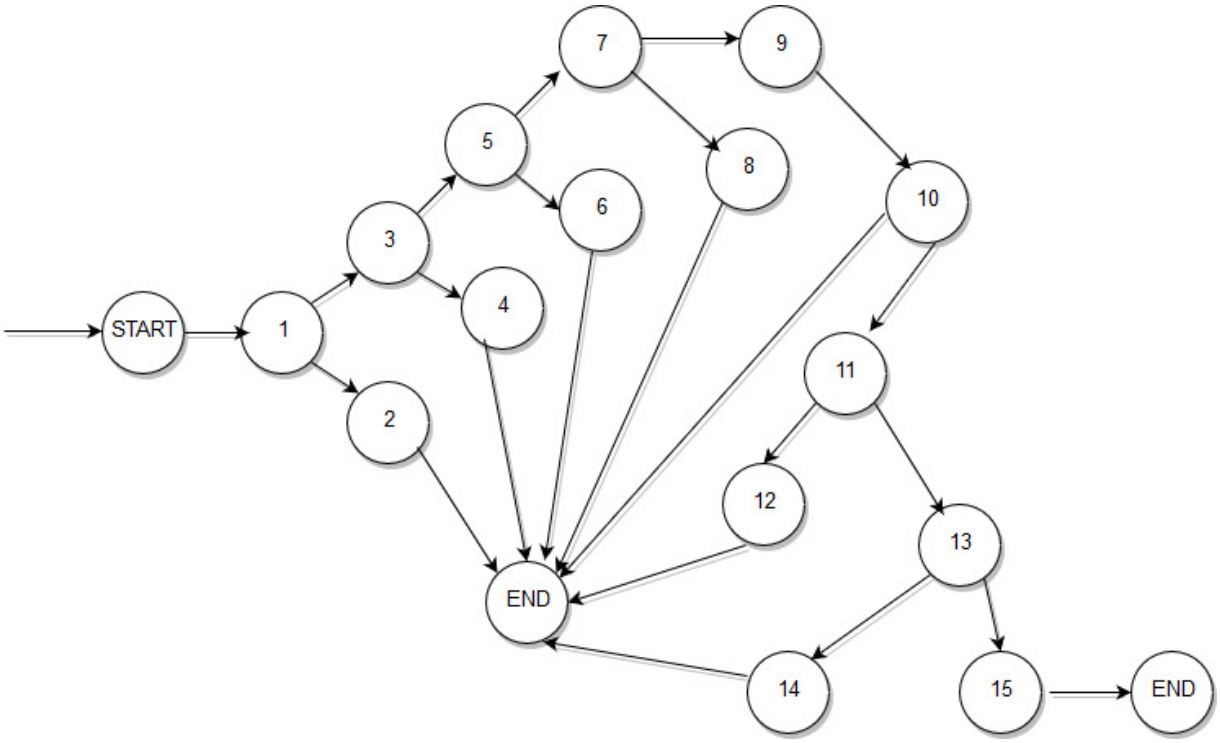
17. is_spec_symbol

```
/************************************/
/* NAME:        is_spec_symbol       */
/* INPUT:       a token */
/* OUTPUT:      a BOOLEAN value      */
  /************************************/
```

```
436 @    static boolean is_spec_symbol(char c)
437      {
438          if (c == '(')
439          {
440              return true;
441          }
442          if (c == ')')
443          {
444              return true;
445          }
446          if (c == '[')
447          {
448              return true;
449          }
450          if (c == ']')
451          {
452              return true;
453          }
454          if (c == '\'')
455          {
456              return true;
457          }
458          if (c == '`')
459          {
460              return true;
461          }
462          if (c == ',')
463          {
464              return true;
465          }
466          return false;     /* others return FALSE */
467      }
```

Nodes	Lines
1	436,437,438
2	439,440

3	442
4	443,444
5	446
6	447,448
7	450
8	451,452
9	454
10	455,456
11	458
12	459, 460
13	462
14	463, 464
15	466



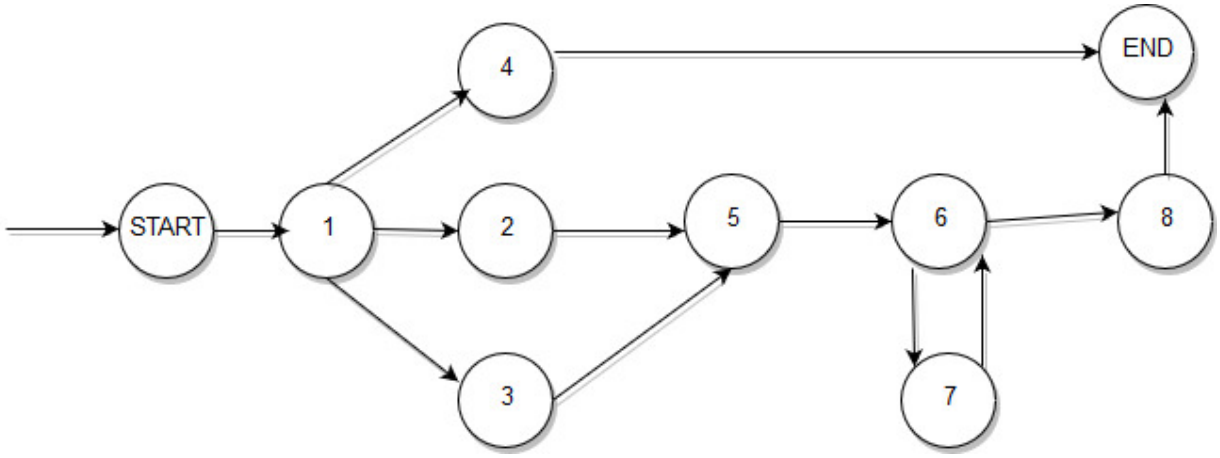
Edge Coverage
[(1,2),(1,3,4),(1,3,5,6),(1,3,5,7,8),(1,3,5,7,9,10),(1,3,5,7,9,10,11,12),(1,3,5,7,9,10,11,13,14),(1,3,5,7,9,10,11,13,15)]

Test data	Test Path
‘(‘	(1,2)
‘)’	(1,3,4)
‘[’	(1,3,5,6)
‘]’	(1,3,5,7,8)
‘/’’	(1,3,5,7,9,10)
‘`’	(1,3,5,7,9,10,11,12)
‘,’	(1,3,5,7,9,10,11,13,14)
‘=’	(1,3,5,7,9,10,11,13,15)

18. main

```
469     public static void main(String[] args) throws IOException {
470         String fname = null;
471         if (args.length == 0) { /* if not given filename,take as '"' */
472             fname = new String();
473         } else if (args.length == 1) {
474             fname = args[0]; //arg[1] is out of bounds
475         } else {
476             System.out.print("Error!,please give the token stream\n");
477             System.exit(0);
478         }
479         Printtokens2 t = new Printtokens2();
480         BufferedReader br = t.open_token_stream(fname); /* open token stream */
481         String tok = t.get_token(br);
482         while (tok != null) {   /* take one token each time until eof */
483             t.print_token(tok);
484             tok = t.get_token(br);
485         }
486 
487         System.exit(0);
488     }
489 }
```

Nodes	Lines
1	(469,470,471)
2	(472)
3	(473,474)
4	(475,477)
5	(479,480,481)
6	(482)
7	(483,484,485)
8	(486,487)



Edge Coverage
[(1,4),(1,2,5,6,7,6,8),(1,3,5,6,8)]

Test data	Test Path
null	(1,3,5,6,8)
1	(1,4,8)
“file.txt”	(1,2,5,6,7,6,8)