CSE-4321 Software Testing Final Project - Test Cases

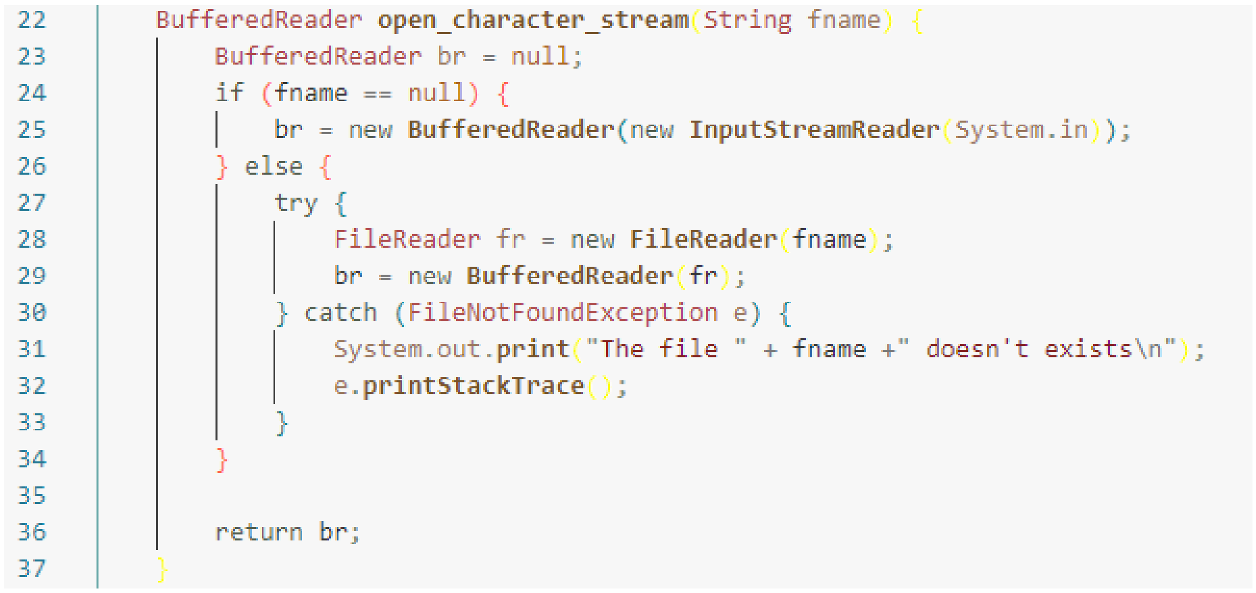
Maximum Effort: Elliot Mai & Benjamin Niccum

December 2, 2024

Contents

1 Unit Tests

ormalsize1.1 Method: openCharacterStream



```
    BufferedReader open_character_stream(String fname) {
        BufferedReader br = null;
        if (fname == null) {
            br = new BufferedReader(new InputStreamReader(System.in));
        } else {
            try {
                FileReader fr = new FileReader(fname);
                br = new BufferedReader(fr);
            } catch (FileNotFoundException e) {
                System.out.print("The file " + fname +" doesn't exists\n");
                e.printStackTrace();
            }
        }

        return br;
    }
```

Figure 1: Code Snippet for openCharacterStream

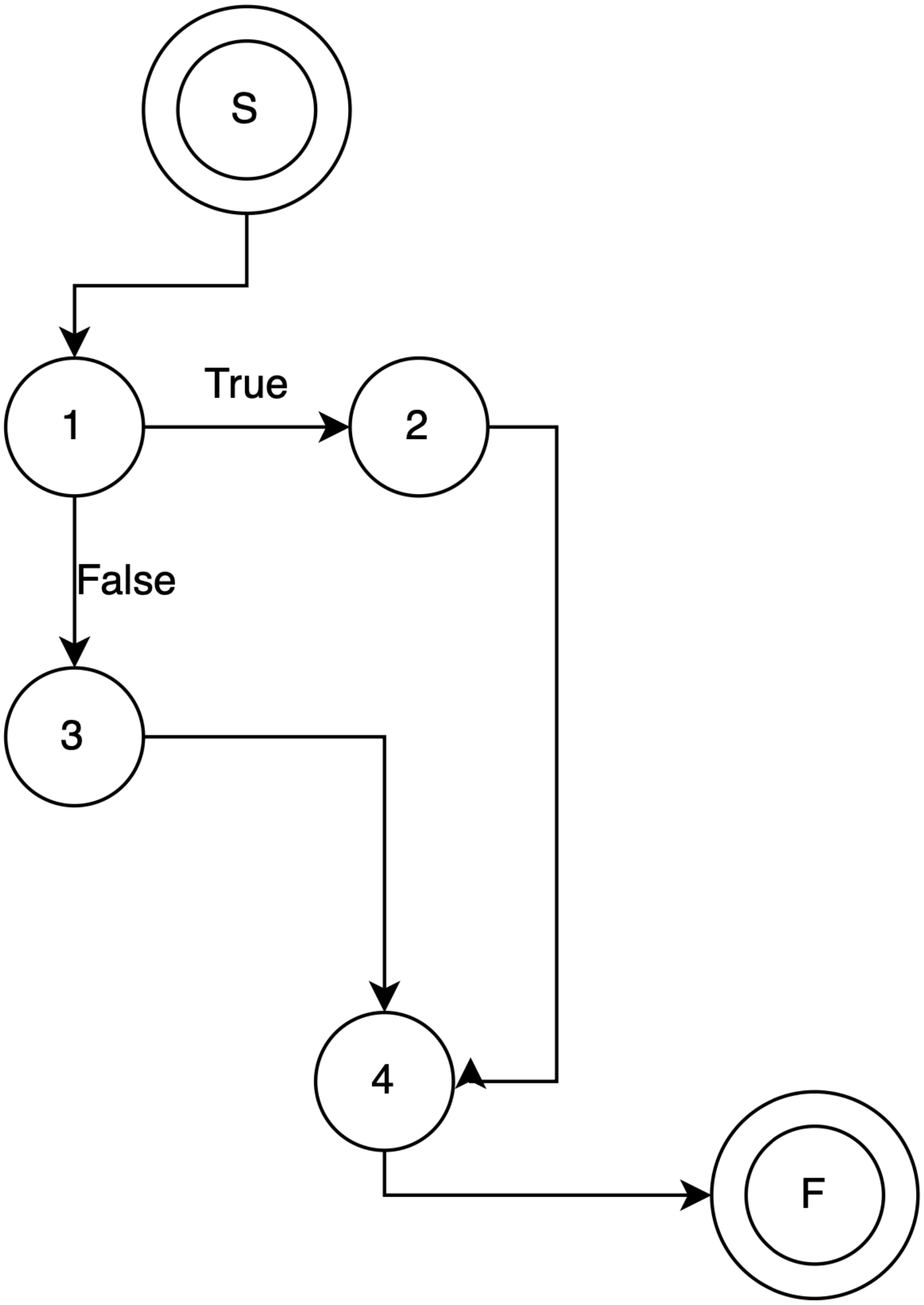
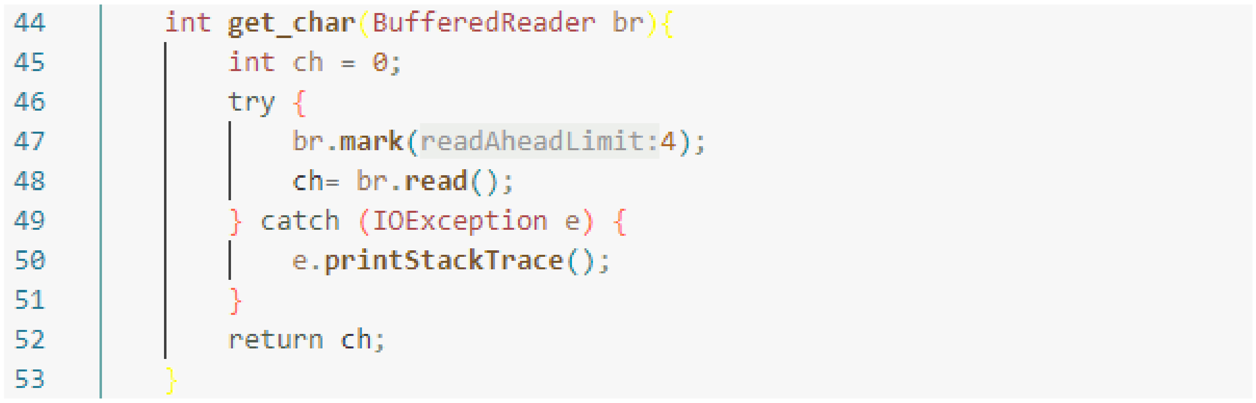


Figure 2: Control Flow Graph for openCharacterStream

Test Path	**Test Data**	**Expected Output**
1, 2, 4	`null`	`notNull`
1, 3, 4	`file.txt`	`True`

Table 1: Test Cases for `openCharacterStream`

ormalsize**1.2 Method: getChar**



```java
    int get_char(BufferedReader br){
        int ch = 0;
        try {
            br.mark(readAheadLimit:4);
            ch= br.read();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return ch;
    }
```

Figure 3: Code Snippet for getChar

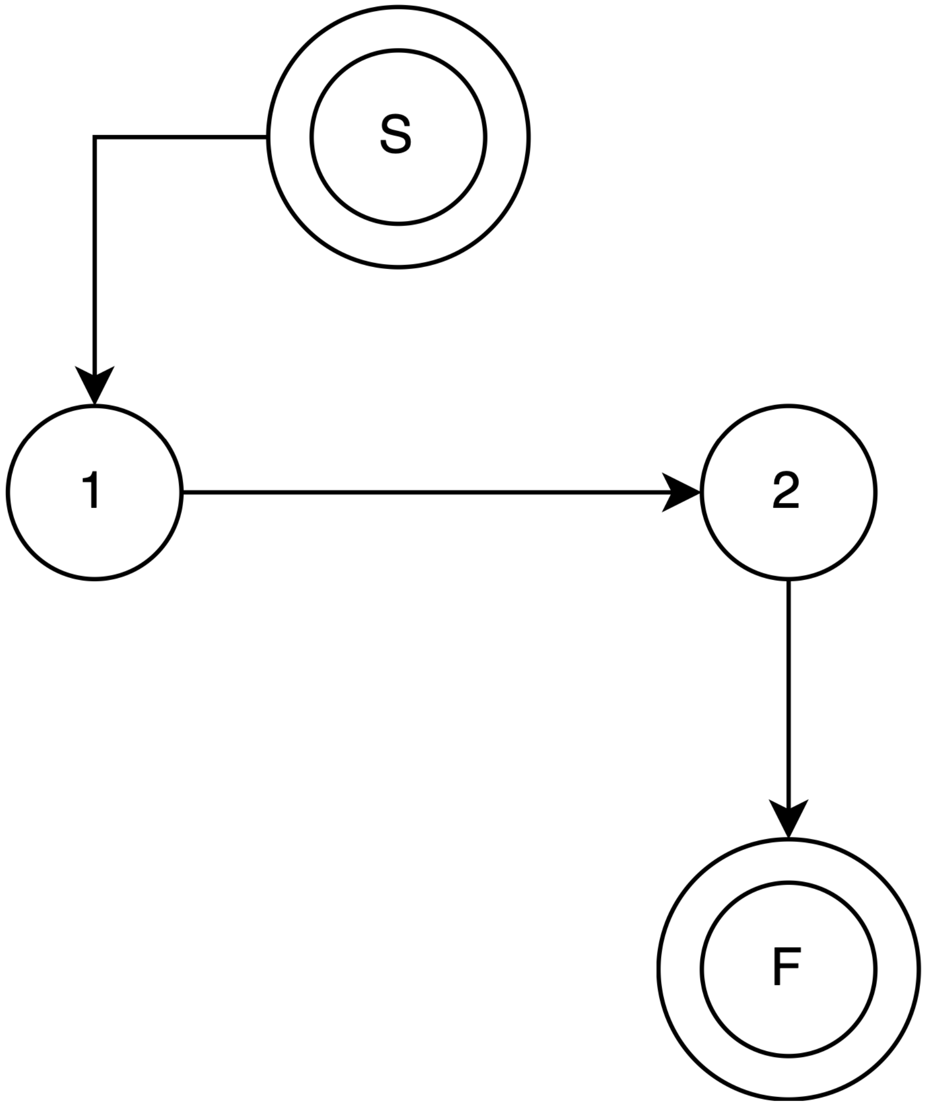
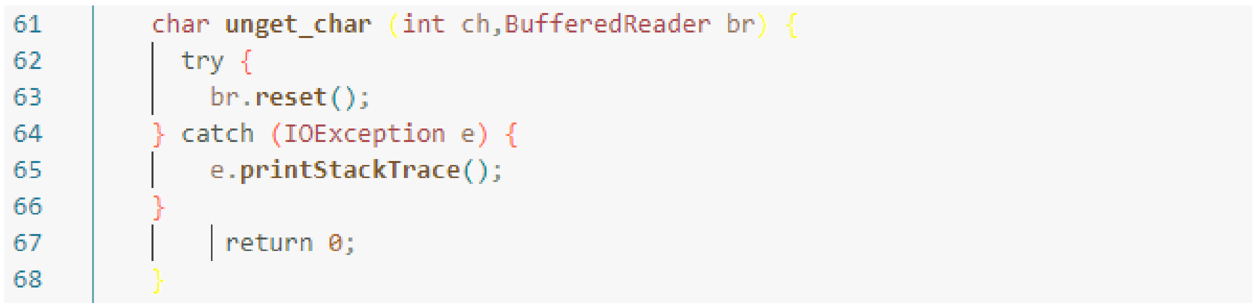


Figure 4: Control Flow Graph for getChar

Test Path	**Test Data**	**Expected Output**
1, 2	`abc`	`a`

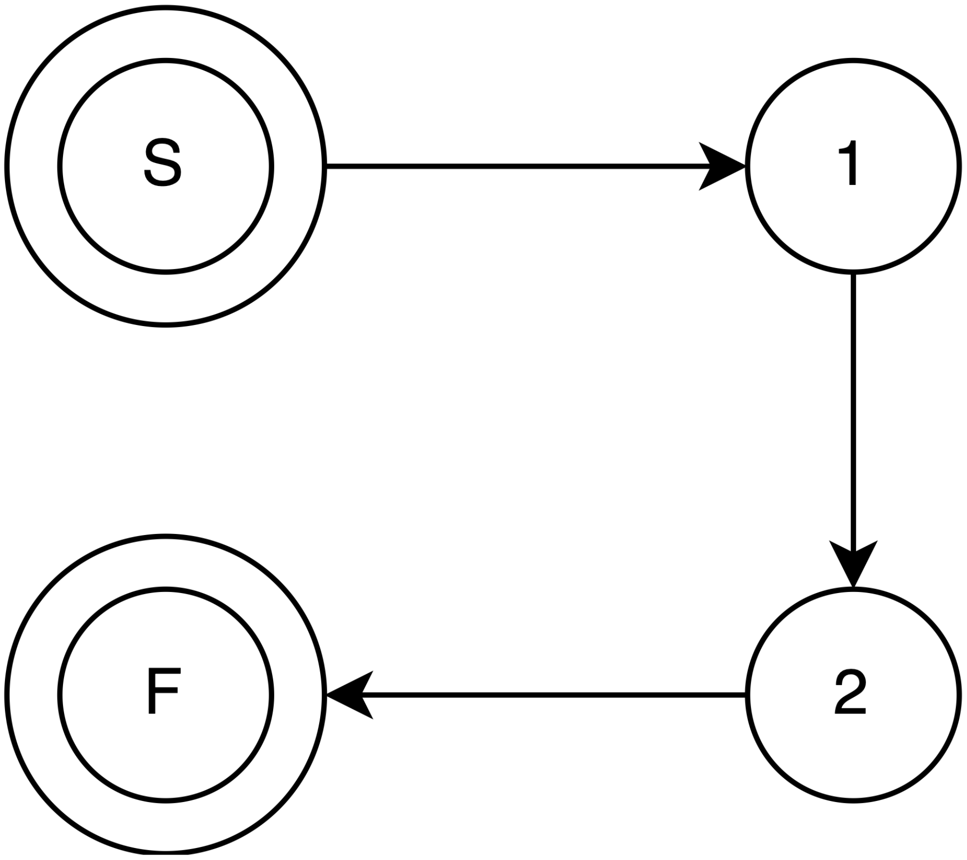
Table 2: Test Cases for `getChar`

ormalsize**1.3 Method: ungetChar**



```
char unget_char (int ch,BufferedReader br) {
  try {
    br.reset();
} catch (IOException e) {
    e.printStackTrace();
}
     return 0;
}
```

Figure 5: Code Snippet for ungetChar



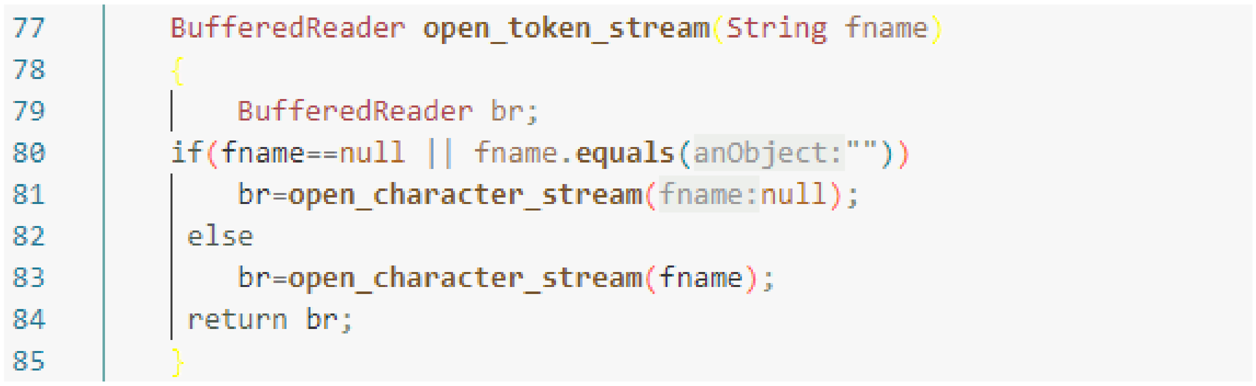
$$\begin{tikzcd}
S \arrow[r] & 1 \arrow[d] \\
F & 2 \arrow[l]
\end{tikzcd}$$

Figure 6: Control Flow Graph for ungetChar

Test Path	Test Data	Expected Output
1, 2	`abc`	`0`

Table 3: Test Cases for `ungetChar`

ormalsize**1.4 Method: openTokenStream**



```
    BufferedReader open_token_stream(String fname)
    {
        BufferedReader br;
    if(fname==null || fname.equals(anObject:""))
        br=open_character_stream(fname:null);
     else
        br=open_character_stream(fname);
     return br;
    }
```

Figure 7: Code Snippet for openTokenStream

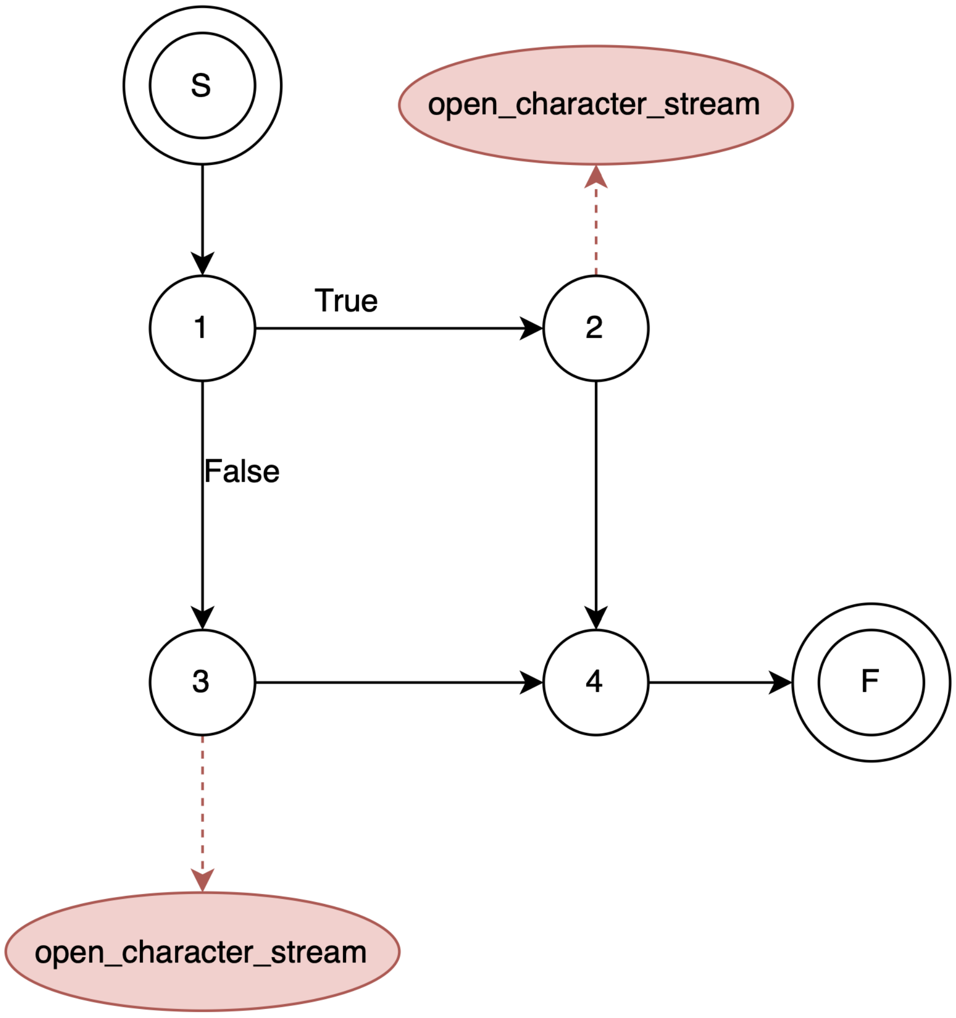
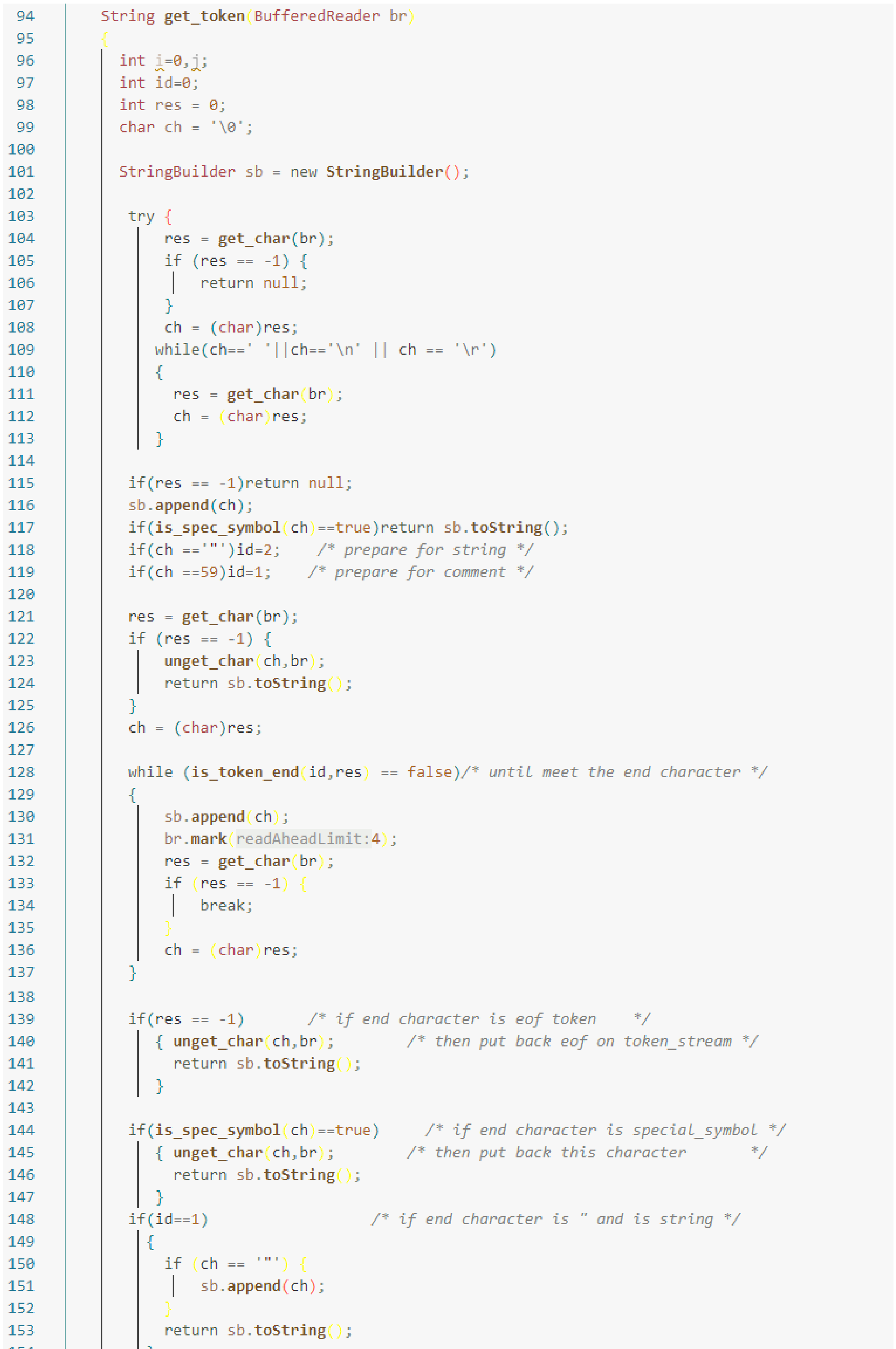


Figure 8: Control Flow Graph for openTokenStream

Test Path	**Test Data**	**Expected Output**
1, 2, 4	`test.txt`	`notNull`
1, 3, 4	`null`	`Null`

Table 4: Test Cases for `openTokenStream`

ormalsize1.5 Method: getToken



```
String get_token(BufferedReader br)
{
  int i=0,j;
  int id=0;
  int res = 0;
  char ch = '\0';

  StringBuilder sb = new StringBuilder();

   try {
       res = get_char(br);
       if (res == -1) {
           return null;
       }
       ch = (char)res;
      while(ch==' '||ch=='\n' || ch == '\r')
      {
        res = get_char(br);
        ch = (char)res;
      }

   if(res == -1)return null;
   sb.append(ch);
   if(is_spec_symbol(ch)==true)return sb.toString();
   if(ch =='"')id=2;    /* prepare for string */
   if(ch ==59)id=1;    /* prepare for comment */

   res = get_char(br);
   if (res == -1) {
       unget_char(ch,br);
       return sb.toString();
   }
   ch = (char)res;

   while (is_token_end(id,res) == false)/* until meet the end character */
   {
       sb.append(ch);
       br.mark(readAheadLimit:4);
       res = get_char(br);
       if (res == -1) {
           break;
       }
       ch = (char)res;
   }

   if(res == -1)       /* if end character is eof token    */
      { unget_char(ch,br);        /* then put back eof on token_stream */
        return sb.toString();
      }

   if(is_spec_symbol(ch)==true)      /* if end character is special_symbol */
      { unget_char(ch,br);        /* then put back this character       */
        return sb.toString();
      }
   if(id==1)                 /* if end character is " and is string */
     {
       if (ch == '"') {
           sb.append(ch);
       }
       return sb.toString();
```

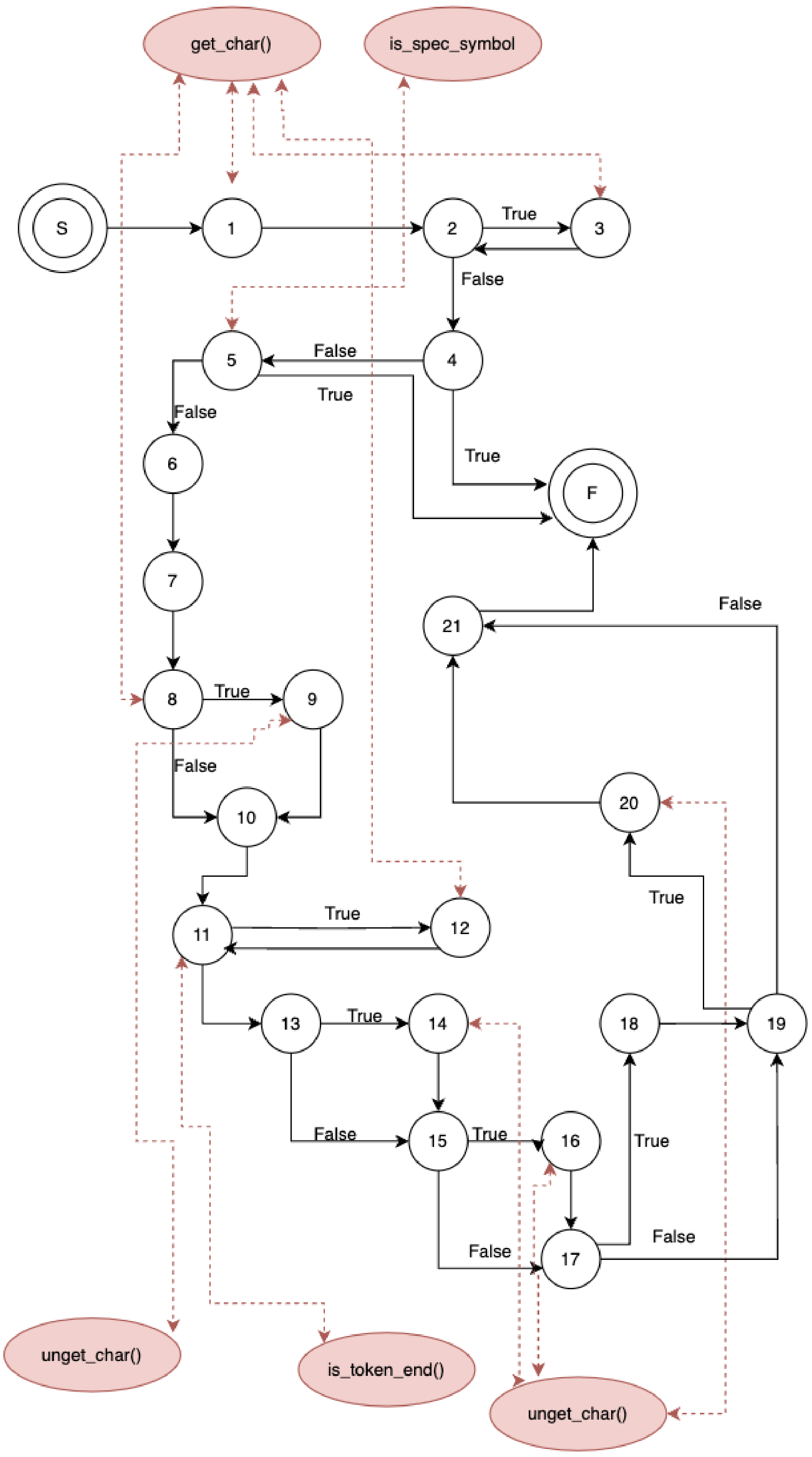
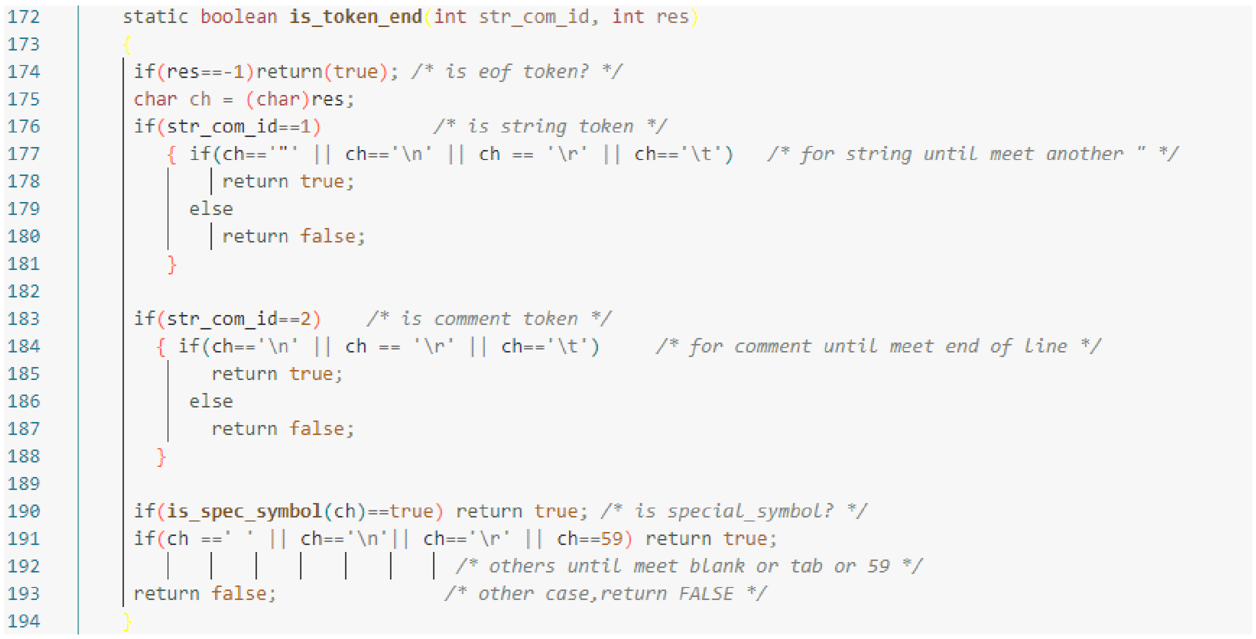


Figure 10: Control Flow Graph for getToken

Test Path	**Test Data**	**Expected Output**
1	`""`	`Null`
1, 2, 3, 2, 4	`"\n\n"`	`Null`
1, 2, 4, 5	`"("`	`"("`
1, 2, 4, 5, 6, 7, 8, 9	`"a"`	`"a"`
1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 11, 13	`"abc"`	`"abc"`
1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15	`"abc)"`	`"abc"`
1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17	`""abc""`	`""abc""`
1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19	`";abc;"`	`";abc;"`
1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21	`"hello"`	`"hello"`

Table 5: Test Cases for `getToken`

ormalsize1.6 Method: isTokenEnd



```
static boolean is_token_end(int str_com_id, int res)
{
 if(res==-1)return(true); /* is eof token? */
 char ch = (char)res;
 if(str_com_id==1)          /* is string token */
    { if(ch=='"' || ch=='\n' || ch == '\r' || ch=='\t')   /* for string until meet another " */
        return true;
      else
        return false;
    }

 if(str_com_id==2)    /* is comment token */
   { if(ch=='\n' || ch == '\r' || ch=='\t')     /* for comment until meet end of line */
        return true;
      else
        return false;
   }

 if(is_spec_symbol(ch)==true) return true; /* is special_symbol? */
 if(ch ==' ' || ch=='\n'|| ch=='\r' || ch==59) return true;
                              /* others until meet blank or tab or 59 */
 return false;                /* other case,return FALSE */
}
```

Figure 11: Code Snippet for isTokenEnd

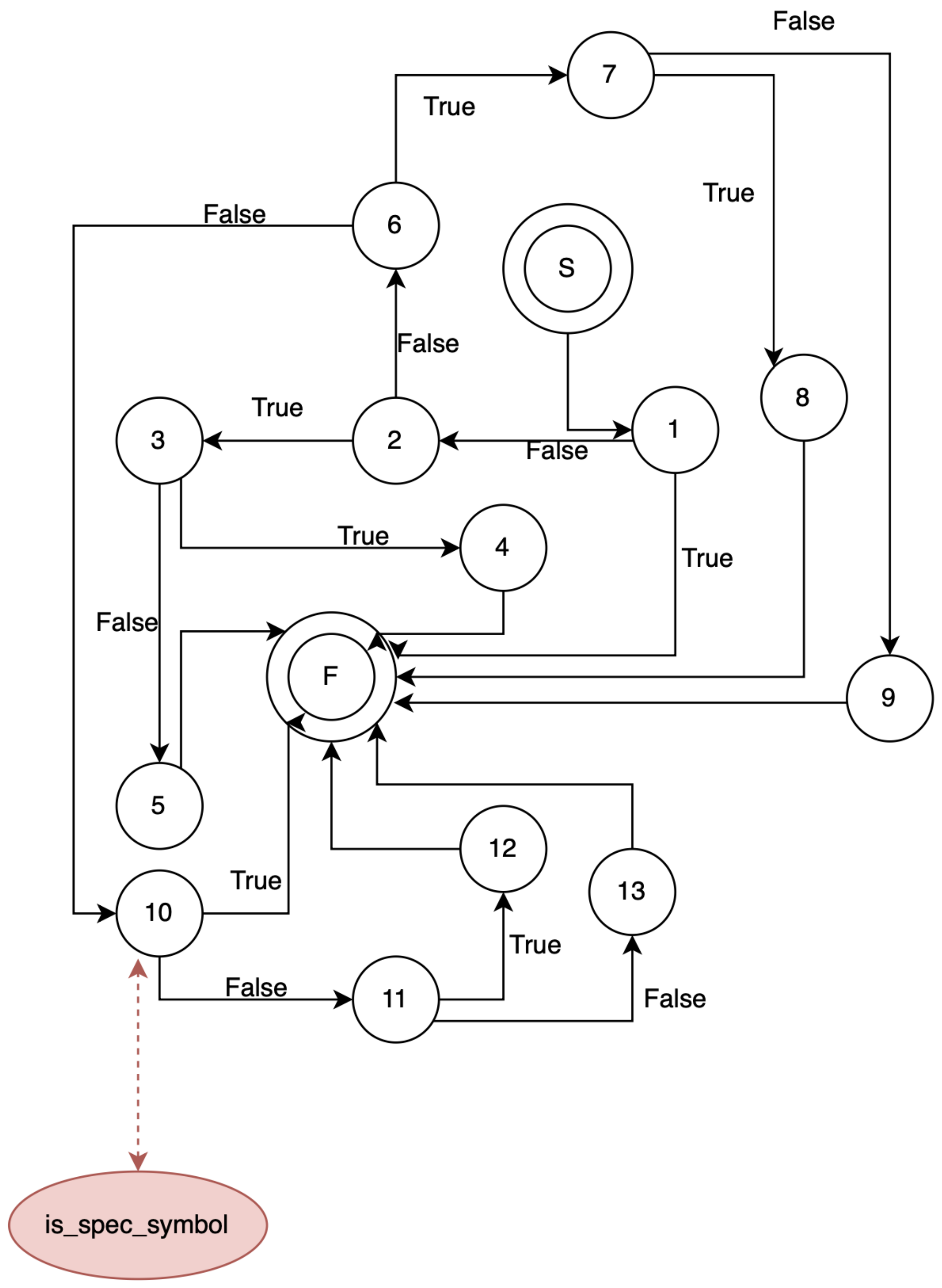


Figure 12: Control Flow Graph for isTokenEnd

Test Path	Test Data	Expected Output
1	`0, -1`	`true`
1, 2, 3, 4	`1, 34`	`true`
1, 2, 3, 5	`1, 97`	`false`
1, 2, 6, 7, 9	`2, 97`	`false`
1, 2, 6, 7, 8	`2, 10`	`true`
1, 2, 6, 10	`0, 40`	`true`
1, 2, 6, 11, 12	`0, 32`	`true`
1, 2, 6, 10, 11, 13	`0, 97`	`false`

Table 6: Test Cases for `isTokenEnd`

ormalsize**1.7 Method: tokenType**

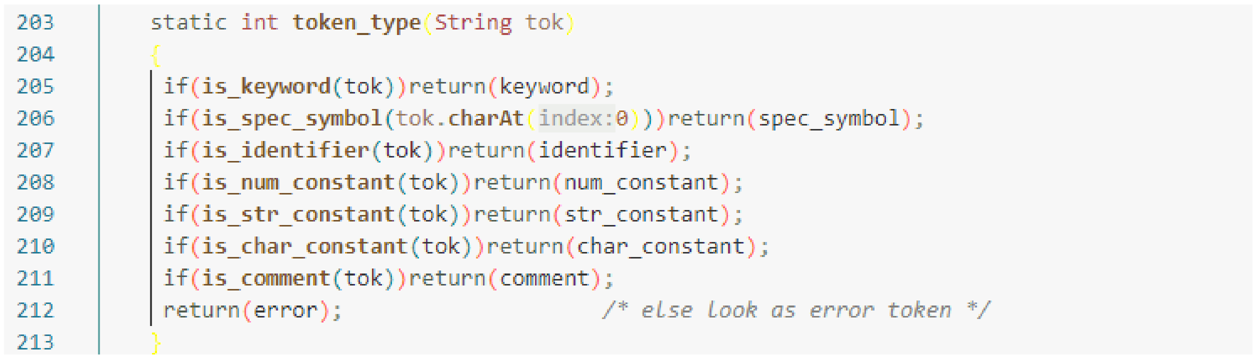

```
static int token_type(String tok)
{
 if(is_keyword(tok))return(keyword);
 if(is_spec_symbol(tok.charAt( index:0)))return(spec_symbol);
 if(is_identifier(tok))return(identifier);
 if(is_num_constant(tok))return(num_constant);
 if(is_str_constant(tok))return(str_constant);
 if(is_char_constant(tok))return(char_constant);
 if(is_comment(tok))return(comment);
 return(error);                  /* else look as error token */
}
```

Figure 13: Code Snippet for tokenType

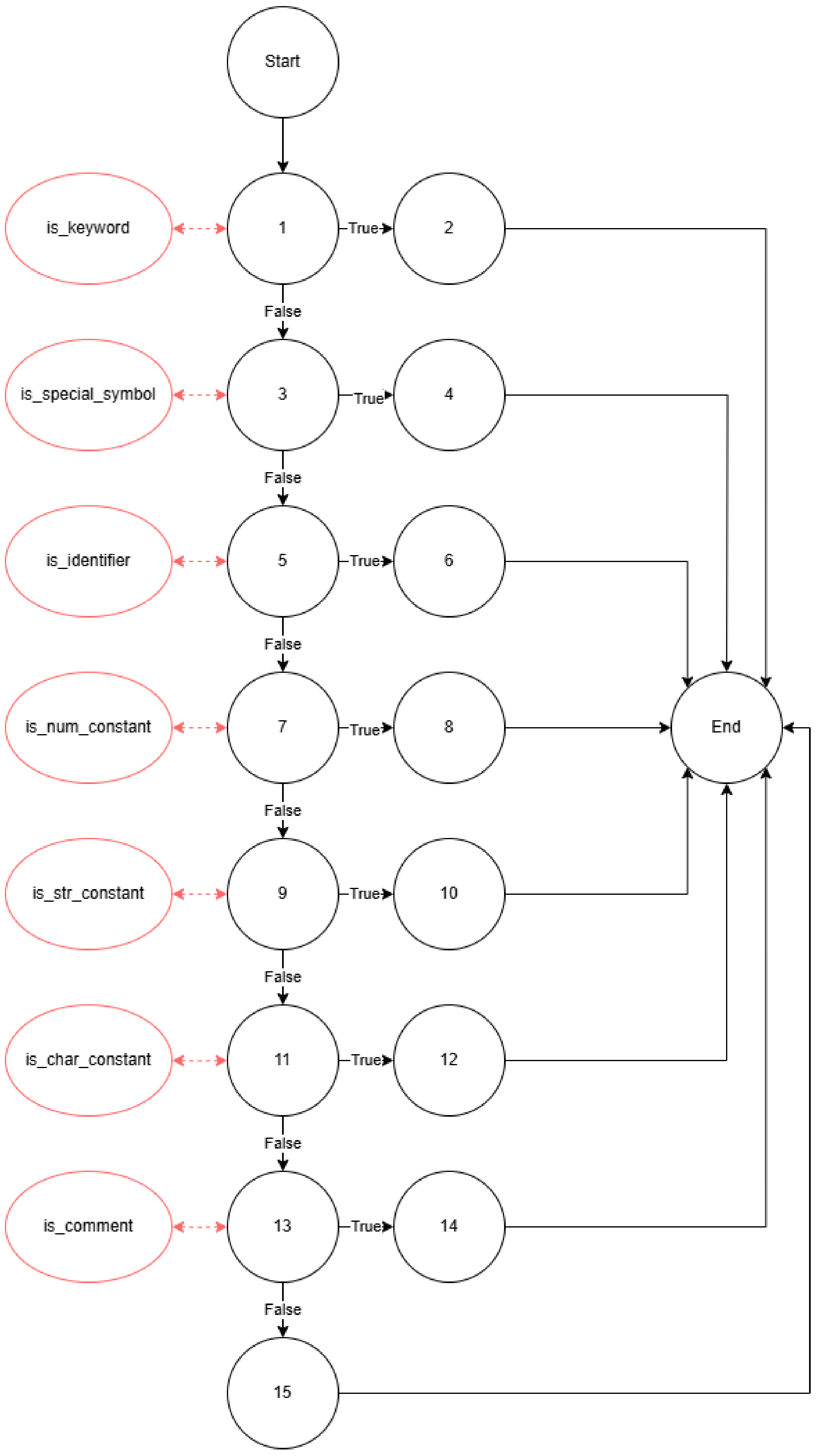
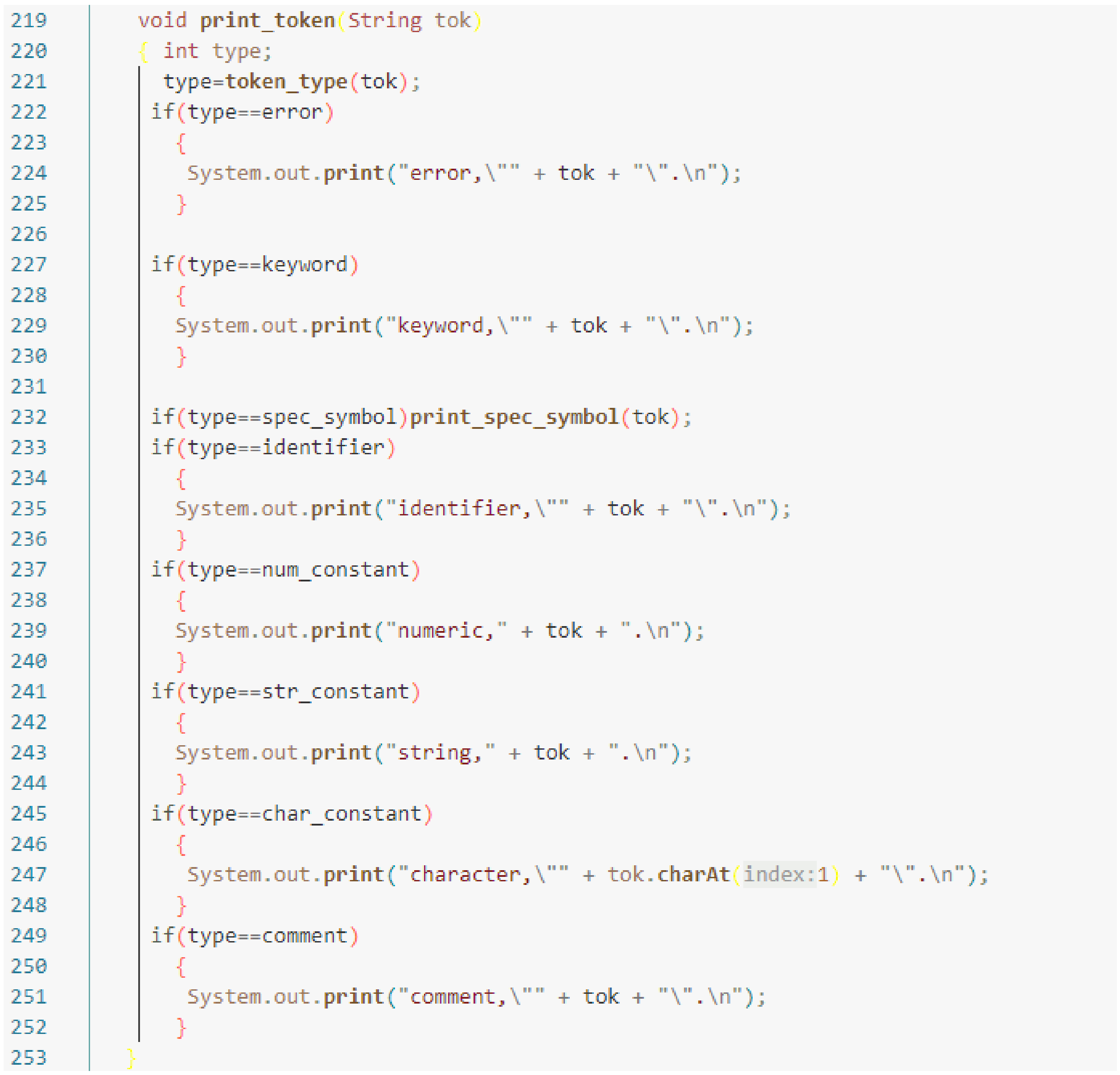


Figure 14: Control Flow Graph for tokenType

Test Path	**Test Data**	**Expected Output**
1, 2	`"and"`	`1`
1, 2	`"or"`	`1`
1, 2	`"if"`	`1`
1, 2	`"xor"`	`1`
1, 2	`"lambda"`	`1`
1, 2	`"=>"`	`1`
1, 3, 4	`"("`	`2`
1, 3, 4	`")"`	`2`
1, 3, 4	`"["`	`2`
1, 3, 4	`"]"`	`2`
1, 3, 4	`"’"`	`2`
1, 3, 4	`"‘"`	`2`
1, 3, 4	`","`	`2`
1, 3, 5, 6	`"variableName"`	`3`
1, 3, 5, 6	`"a"`	`3`
1, 3, 5, 6	`"aa"`	`3`
1, 3, 5, 6	`"a1"`	`3`
1, 3, 5, 6	`"a2"`	`3`
1, 3, 5, 7, 8	`"123"`	`41`
1, 3, 5, 7, 8	`"1"`	`41`
1, 3, 5, 7, 8	`"321"`	`41`
1, 3, 5, 7, 9, 10	`""Hello""`	`42`
1, 3, 5, 7, 9, 10	`""asd""`	`42`
1, 3, 5, 7, 9, 10	`""123""`	`42`
1, 3, 5, 7, 9, 11, 13, 14	`";comment"`	`5`
1, 3, 5, 7, 9, 11, 12	`"#a"`	`43`
1, 3, 5, 7, 9, 11, 12	`"#b"`	`43`
1, 3, 5, 7, 9, 11, 13, 15	`"*^&"`	`0`

Table 7: Test Cases for `tokenType`

ormalsize1.8 Method: printToken



```
void print_token(String tok)
{ int type;
  type=token_type(tok);
 if(type==error)
   {
    System.out.print("error,\"" + tok + "\".\n");
   }

 if(type==keyword)
   {
   System.out.print("keyword,\"" + tok + "\".\n");
   }

 if(type==spec_symbol)print_spec_symbol(tok);
 if(type==identifier)
   {
   System.out.print("identifier,\"" + tok + "\".\n");
   }
 if(type==num_constant)
   {
   System.out.print("numeric," + tok + ".\n");
   }
 if(type==str_constant)
   {
   System.out.print("string," + tok + ".\n");
   }
 if(type==char_constant)
   {
    System.out.print("character,\"" + tok.charAt(index:1) + "\".\n");
   }
 if(type==comment)
   {
    System.out.print("comment,\"" + tok + "\".\n");
   }
}
```

Figure 15: Code Snippet for printToken

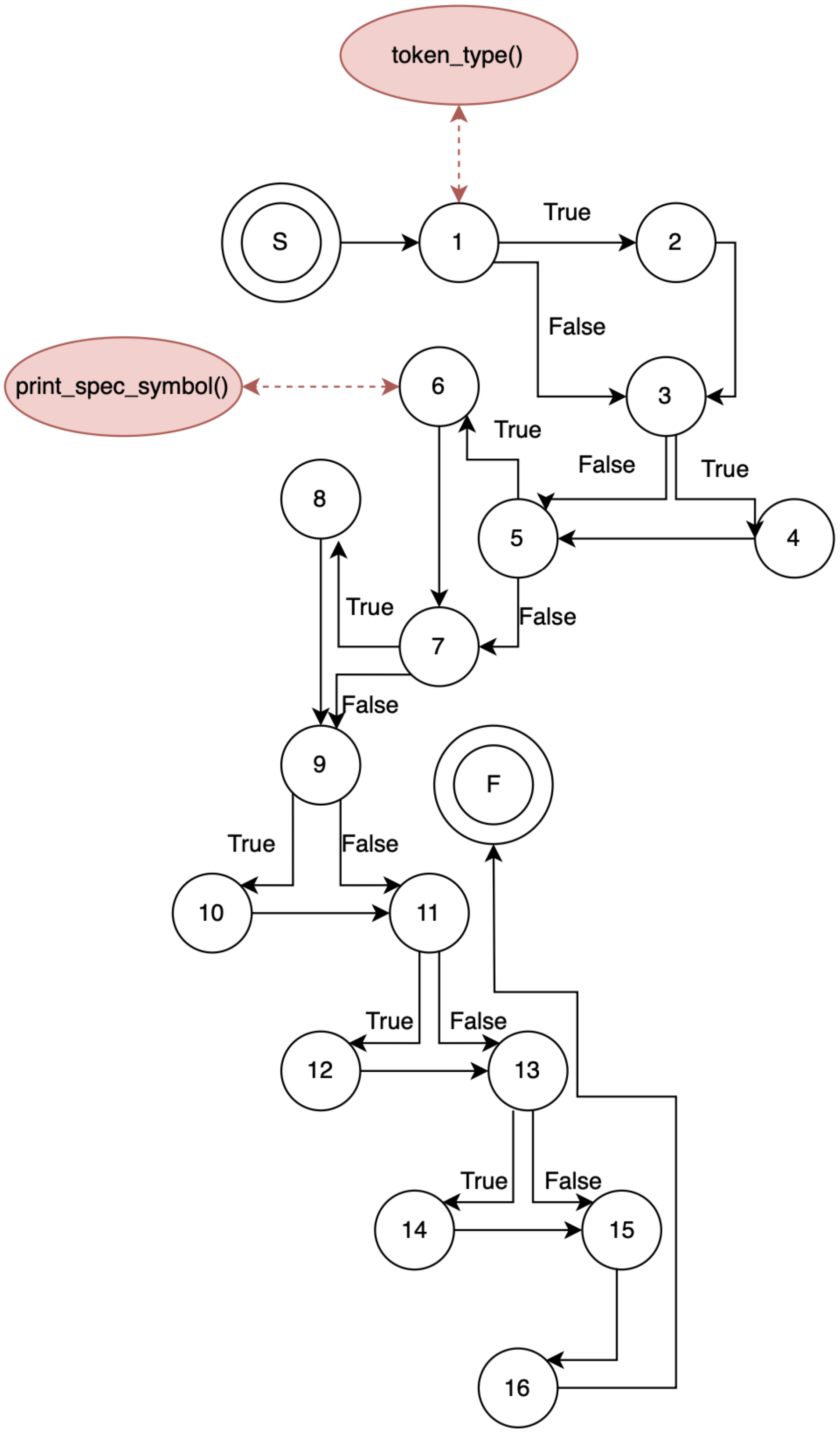
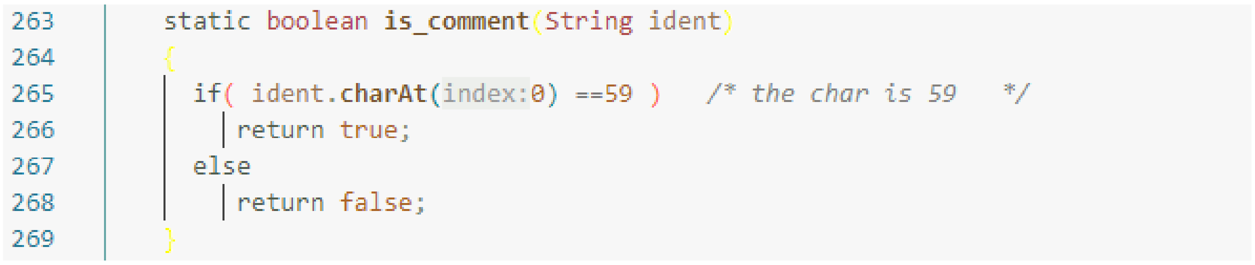


Figure 16: Control Flow Graph for printToken

Test Path	Test Data	Expected Output
1, 2, 3, 5, 7, 9, 11, 13, 15	`"*^&"`	`"error,"*^&"."`
1, 3, 4, 5, 7, 9, 11, 13, 15	`"if"`	`"keyword,"if"."`
1, 3, 5, 6, 7, 9, 11, 13, 15	`"("`	`"lparen."`
1, 3, 5, 7, 8, 9, 11, 13, 15	`"variableName"`	`identifier,"variableName".`
1, 3, 5, 7, 9, 10, 11, 13, 15	`"123"`	`numeric, 123.`
1, 3, 5, 7, 9, 11, 12, 13, 15	`""Hello""`	`"string, "Hello"."`
1, 3, 5, 7, 9, 11, 13, 14, 15	`"#a"`	`"character, "a"."`
1, 3, 5, 7, 9, 11, 13, 15, 16	`";comment"`	`"comment, ";comment"."`

Table 8: Test Cases for `printToken`

ormalsize1.9 Method: isComment



```
static boolean is_comment(String ident)
{
    if( ident.charAt(index:0) ==59 )   /* the char is 59   */
        return true;
    else
        return false;
}
```

Figure 17: Code Snippet for isComment

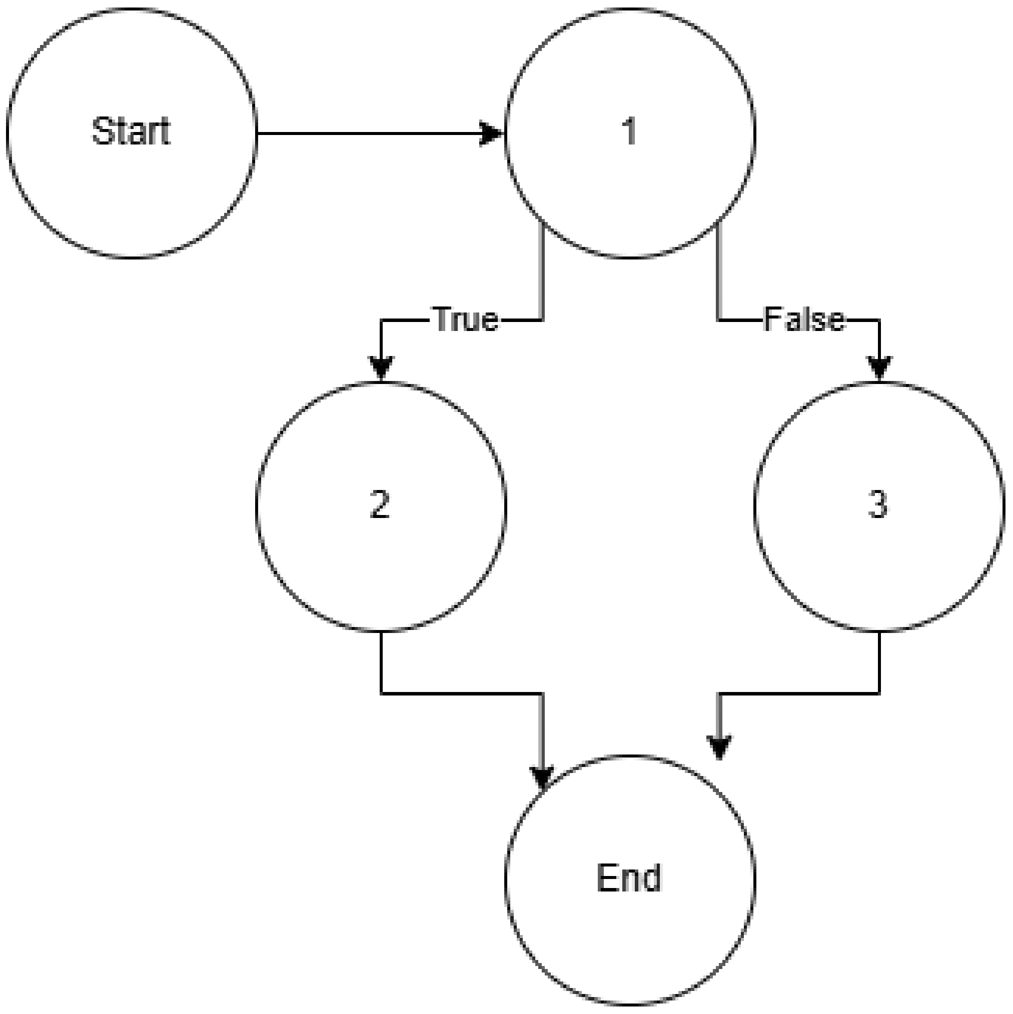
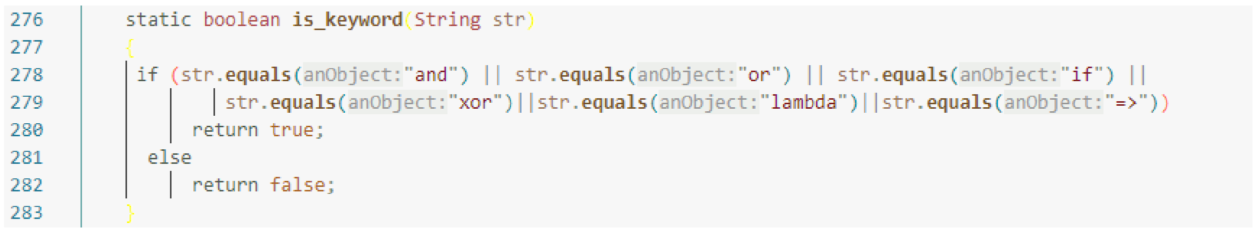


Figure 18: Control Flow Graph for isComment

Test Path	**Test Data**	**Expected Output**
1, 2	`";"`	`true`
1, 3	`"x"`	`false`

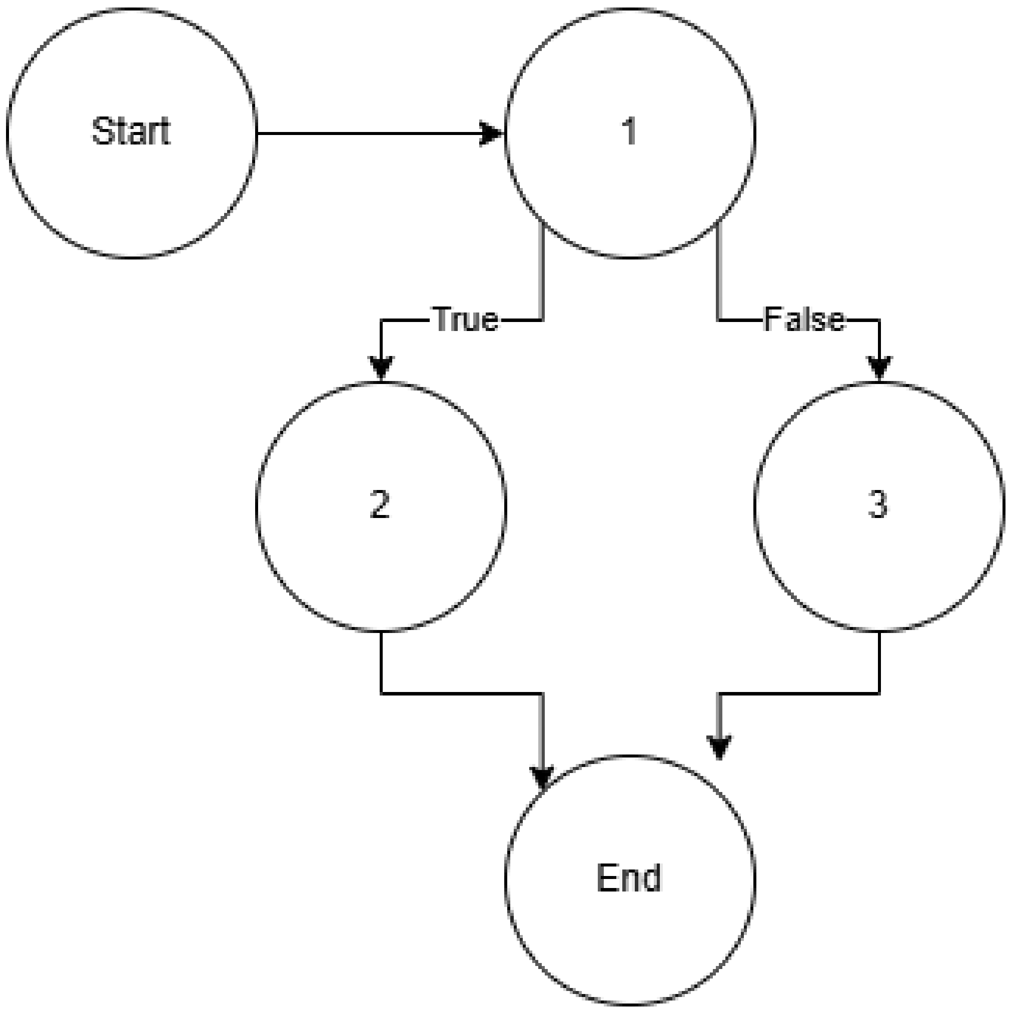
Table 9: Test Cases for `isComment`

ormalsize1.10 Method: isKeyword



```
static boolean is_keyword(String str)
{
if (str.equals(anObject:"and") || str.equals(anObject:"or") || str.equals(anObject:"if") ||
        str.equals(anObject:"xor")||str.equals(anObject:"lambda")||str.equals(anObject:"=>"))
    return true;
 else
    return false;
}
```

Figure 19: Code Snippet for isKeyword



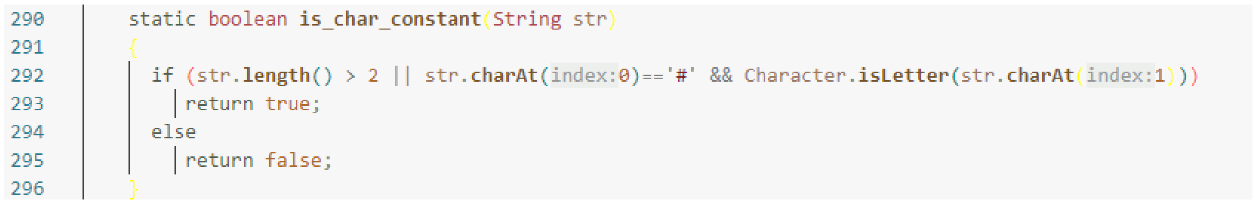
$$\begin{tikzcd}
\text{Start} \arrow[r] & 1 \arrow[dl, "\text{True}"'] \arrow[dr, "\text{False}"] & \\
2 \arrow[dr] & & 3 \arrow[dl] \\
& \text{End} &
\end{tikzcd}$$

Figure 20: Control Flow Graph for isKeyword

Test Path	Test Data	Expected Output
1, 2	`"and"`	`true`
1, 3	`"hello"`	`false`

Table 10: Test Cases for `isKeyword`

ormalsize**1.11 Method: isCharConstant**



```
static boolean is_char_constant(String str)
{
  if (str.length() > 2 || str.charAt(index:0)=='#' && Character.isLetter(str.charAt(index:1)))
    return true;
  else
    return false;
}
```

Figure 21: Code Snippet for isCharConstant

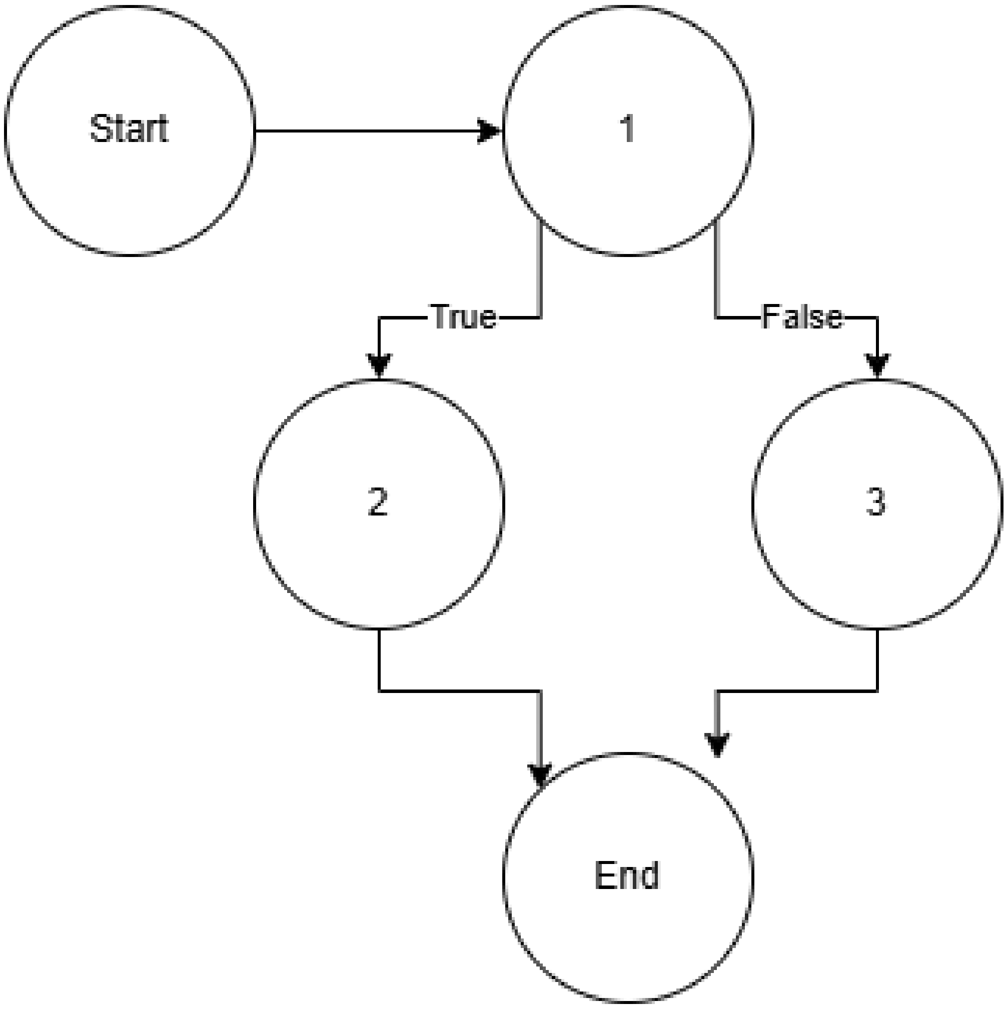
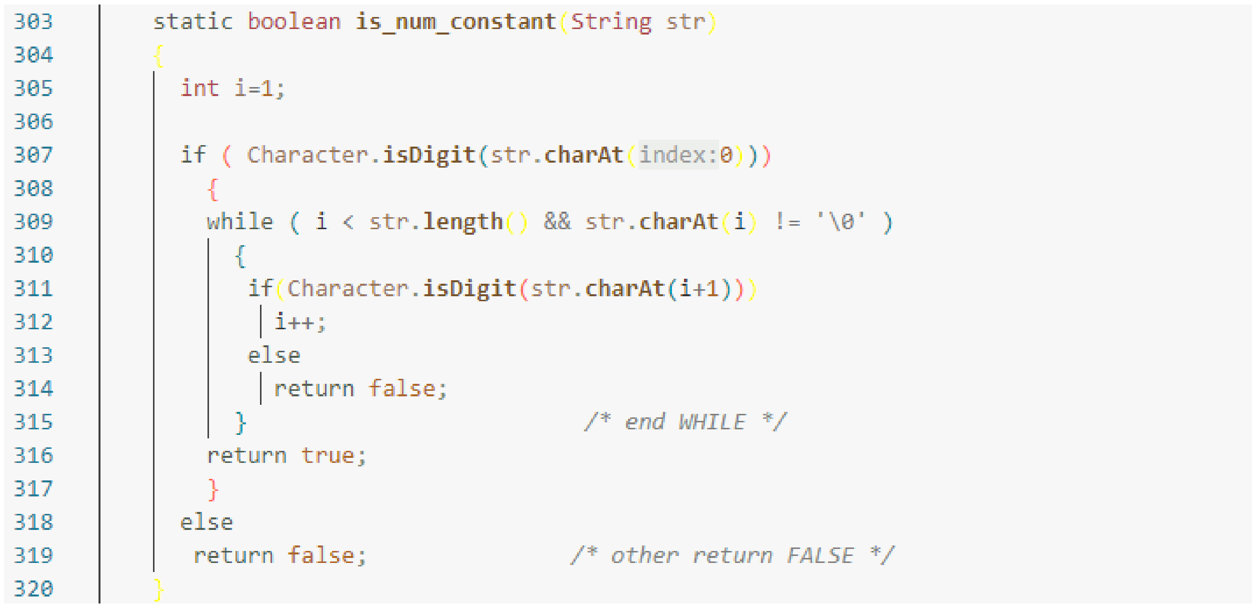


Figure 22: Control Flow Graph for isCharConstant

Test Path	Test Data	Expected Output
1, 2	`"#a"`	`true`
1, 3	`"abc"`	`false`

Table 11: Test Cases for `isCharConstant`

ormalsize1.12 Method: isNumConstant



```
static boolean is_num_constant(String str)
{
  int i=1;

  if ( Character.isDigit(str.charAt(index:0)))
    {
    while ( i < str.length() && str.charAt(i) != '\0' )
      {
       if(Character.isDigit(str.charAt(i+1)))
         i++;
       else
         return false;
      }                          /* end WHILE */
    return true;
    }
  else
   return false;               /* other return FALSE */
}
```

Figure 23: Code Snippet for isNumConstant

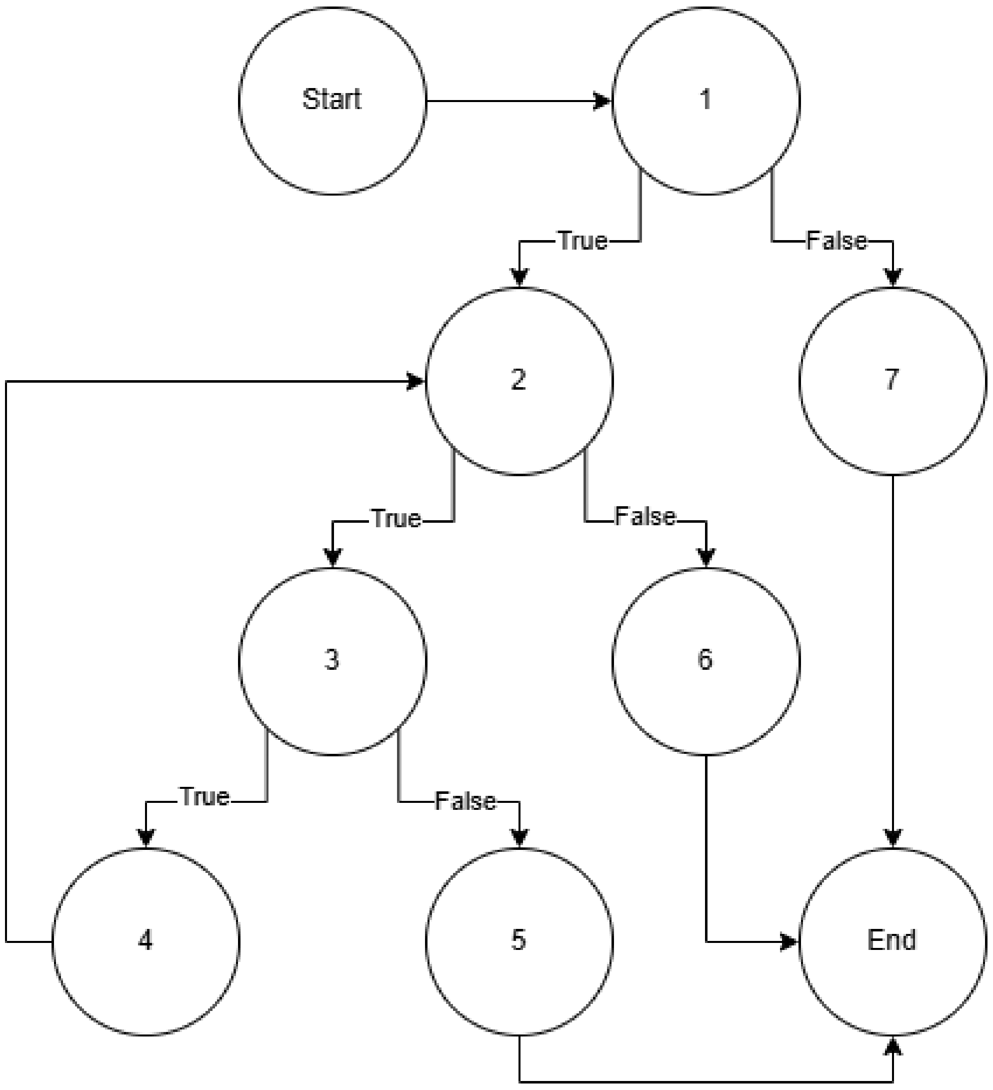
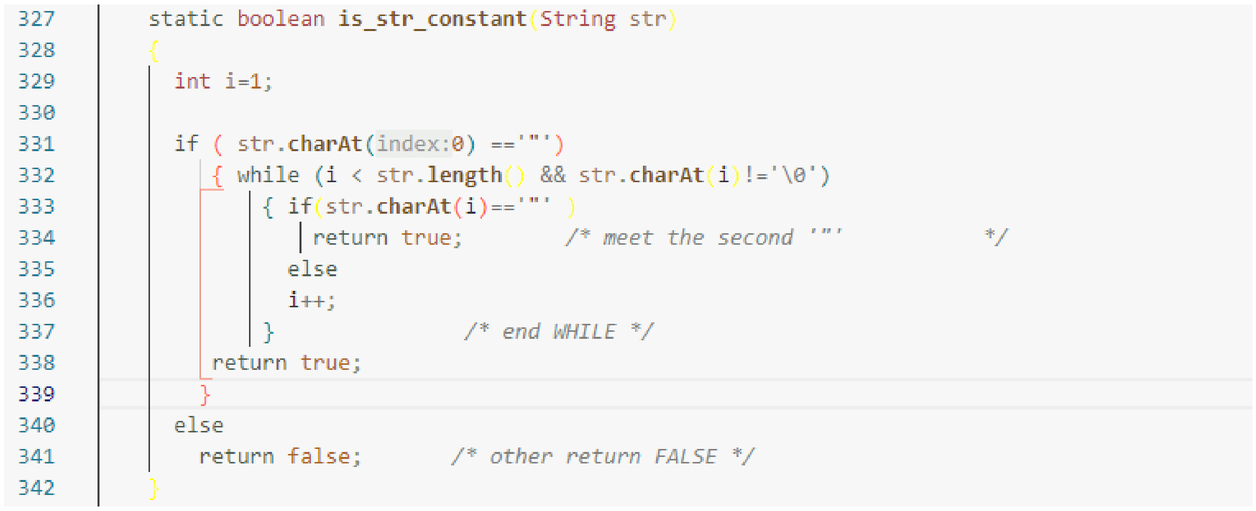


Figure 24: Control Flow Graph for isNumConstant

Test Path	**Test Data**	**Expected Output**
1, 7	`str = "abc"`	`false`
1, 2, 3, 4, 2, 6	`"123"`	`true`
1, 2, 3, 5	`"12a"`	`false`
1, 2, 6	`"1"`	`false`

Table 12: Test Cases for `isNumConstant`

ormalsize**1.13 Method: isStrConstant**



```
static boolean is_str_constant(String str)
{
  int i=1;

  if ( str.charAt(index:0) =='"')
     { while (i < str.length() && str.charAt(i)!='\0')
         { if(str.charAt(i)=='"' )
             return true;        /* meet the second '"'           */
           else
           i++;
         }               /* end WHILE */
     return true;
    }
  else
    return false;       /* other return FALSE */
}
```

Figure 25: Code Snippet for isStrConstant

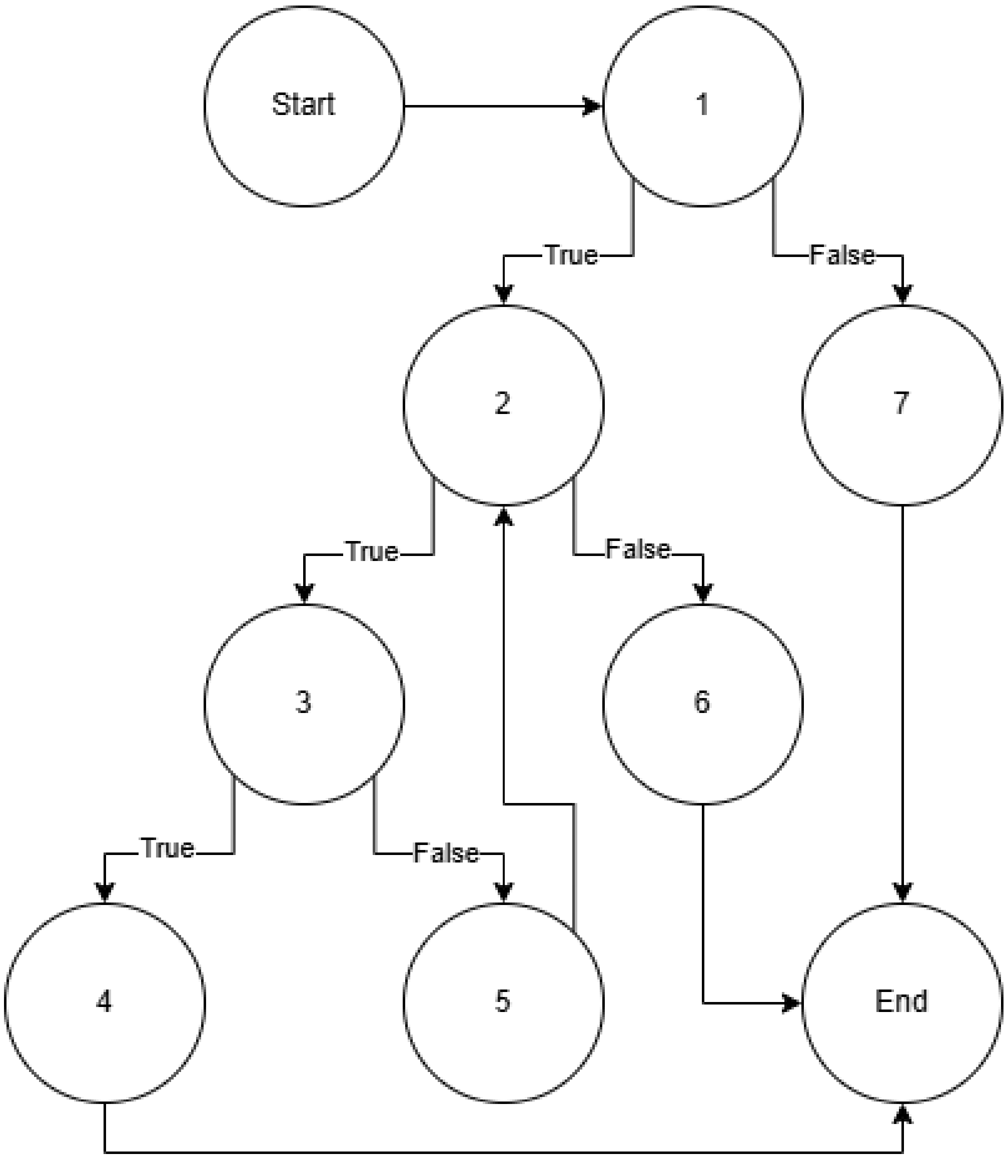
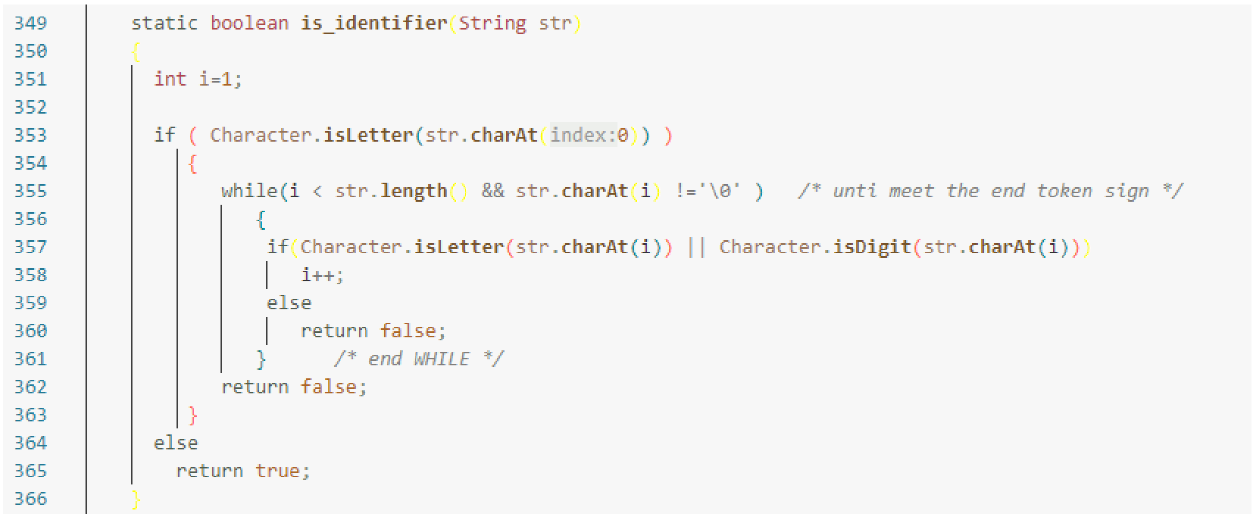


Figure 26: Control Flow Graph for isStrConstant

Test Path	**Test Data**	**Expected Output**
1, 7	`"abc"`	`false`
1, 2, 3, 4	`""""`	`false`
1, 2, 3, 5, 2, 6	`""a""`	`true`
1, 2, 6	`"""`	`false`

Table 13: Test Cases for `isStrConstant`

ormalsize**1.14 Method: isIdentifier**



```
static boolean is_identifier(String str)
{
  int i=1;

  if ( Character.isLetter(str.charAt(index:0)) )
    {
       while(i < str.length() && str.charAt(i) !='\0' )   /* unti meet the end token sign */
          {
           if(Character.isLetter(str.charAt(i)) || Character.isDigit(str.charAt(i)))
              i++;
           else
              return false;
          }      /* end WHILE */
       return false;
    }
  else
    return true;
}
```

Figure 27: Code Snippet for isIdentifier

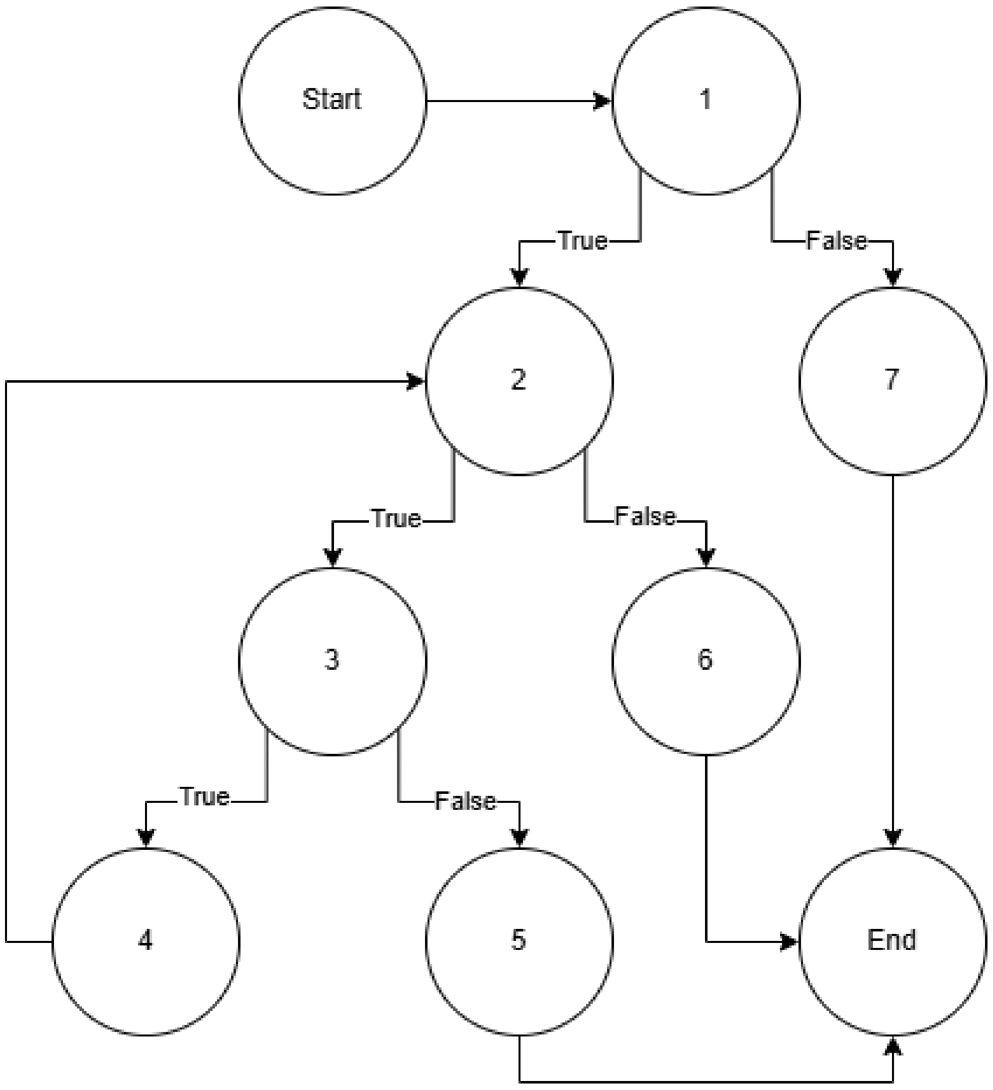
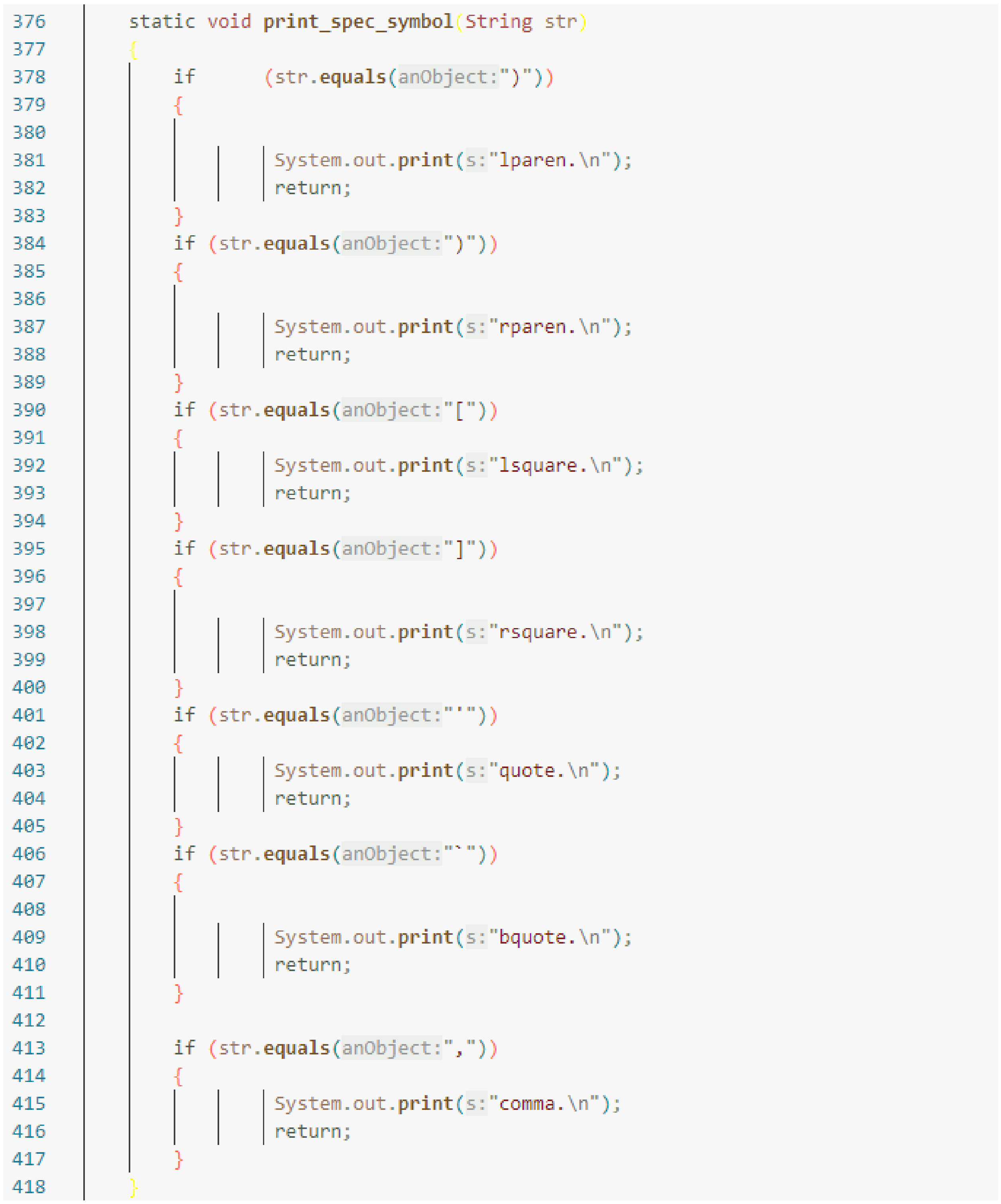


Figure 28: Control Flow Graph for isIdentifier

Test Path	**Test Data**	**Expected Output**
1, 7	`"1"`	`false`
1, 2, 3, 4, 2, 6	`"a1"`	`true`
1, 2, 3, 5	`"output"`	`true`
1, 2, 6	`"1output"`	`false`

Table 14: Test Cases for `isIdentifier`

ormalsize1.15 Method: printSpecSymbol



```java
static void print_spec_symbol(String str)
{
    if      (str.equals(anObject:"("))
    {

            System.out.print(s:"lparen.\n");
            return;
    }
    if (str.equals(anObject:")"))
    {

            System.out.print(s:"rparen.\n");
            return;
    }
    if (str.equals(anObject:"["))
    {
            System.out.print(s:"lsquare.\n");
            return;
    }
    if (str.equals(anObject:"]"))
    {

            System.out.print(s:"rsquare.\n");
            return;
    }
    if (str.equals(anObject:"'"))
    {
            System.out.print(s:"quote.\n");
            return;
    }
    if (str.equals(anObject:"`"))
    {

            System.out.print(s:"bquote.\n");
            return;
    }

    if (str.equals(anObject:","))
    {
            System.out.print(s:"comma.\n");
            return;
    }
}
```

Figure 29: Code Snippet for printSpecSymbol

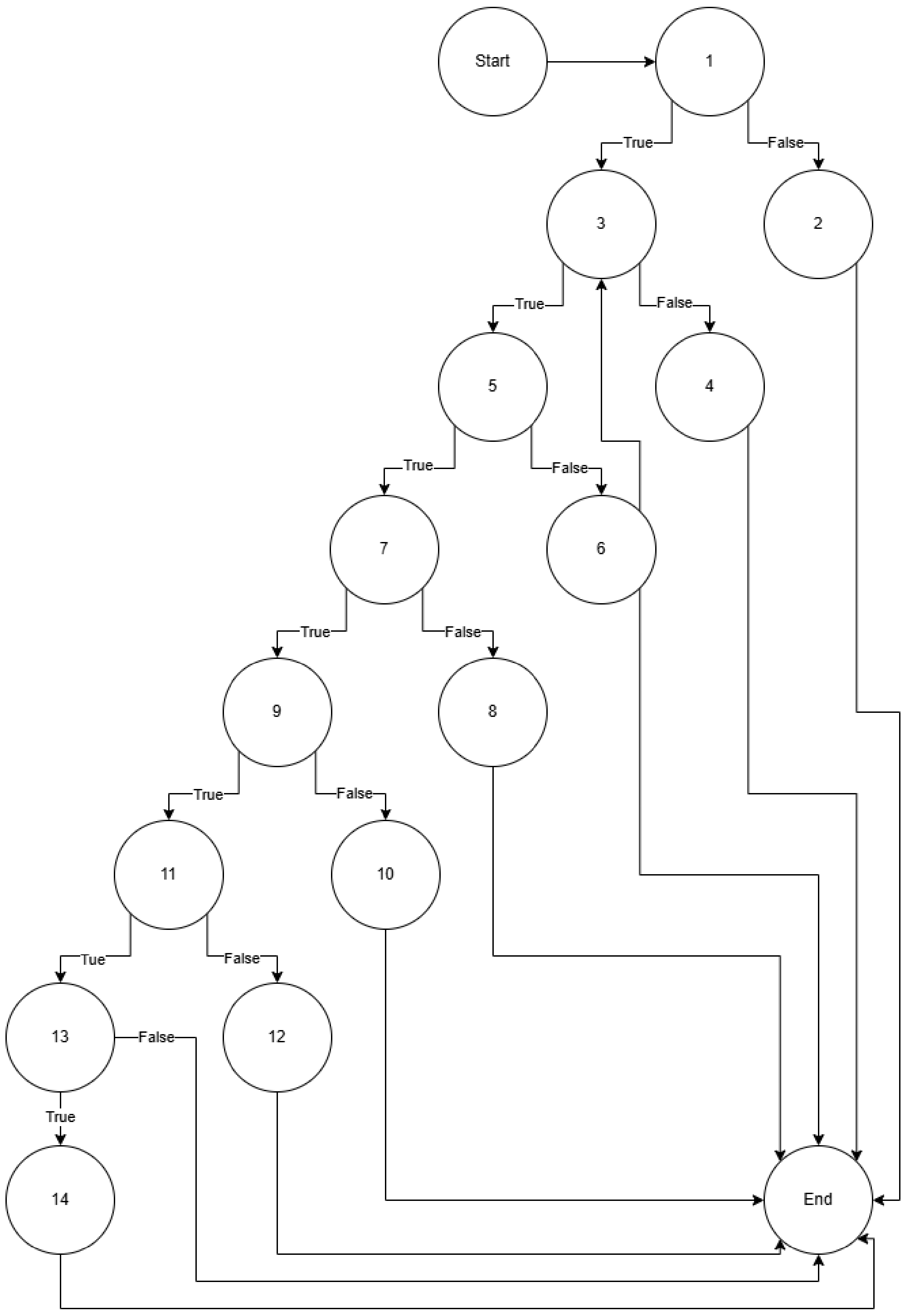
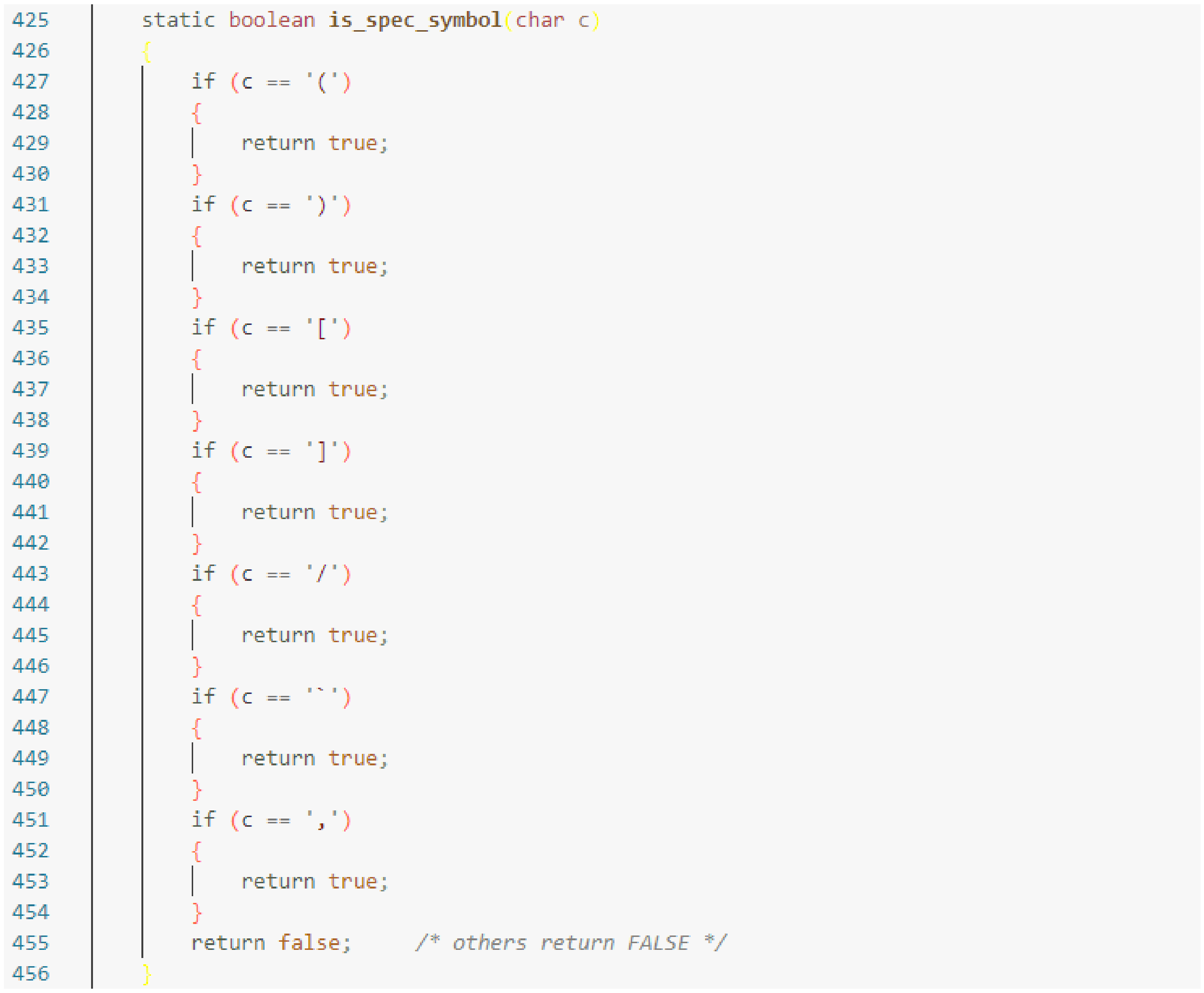


Figure 30: Control Flow Graph for printSpecSymbol

Test Path	Test Data	Expected Output
1, 2	")"	"lparen."
1, 3, 4	"("	"rparen."
1, 3, 5, 6	"["	"lsquare."
1, 3, 5, 7, 8	"]"	"rsquare."
1, 3, 5, 7, 9, 10	"’"	"quote."
1, 3, 5, 7, 9, 11, 12	"‘"	"bquote."
1, 3, 5, 7, 9, 11, 13	","	"comma."
1, 3, 5, 7, 9, 11, 13, 14	"unknown"	error

Table 15: Test Cases for `printSpecSymbol`

ormalsize**1.16 Method: isSpecSymbol**



```
static boolean is_spec_symbol(char c)
{
    if (c == '(')
    {
        return true;
    }
    if (c == ')')
    {
        return true;
    }
    if (c == '[')
    {
        return true;
    }
    if (c == ']')
    {
        return true;
    }
    if (c == '/')
    {
        return true;
    }
    if (c == '`')
    {
        return true;
    }
    if (c == ',')
    {
        return true;
    }
    return false;     /* others return FALSE */
}
```

Figure 31: Code Snippet for isSpecSymbol

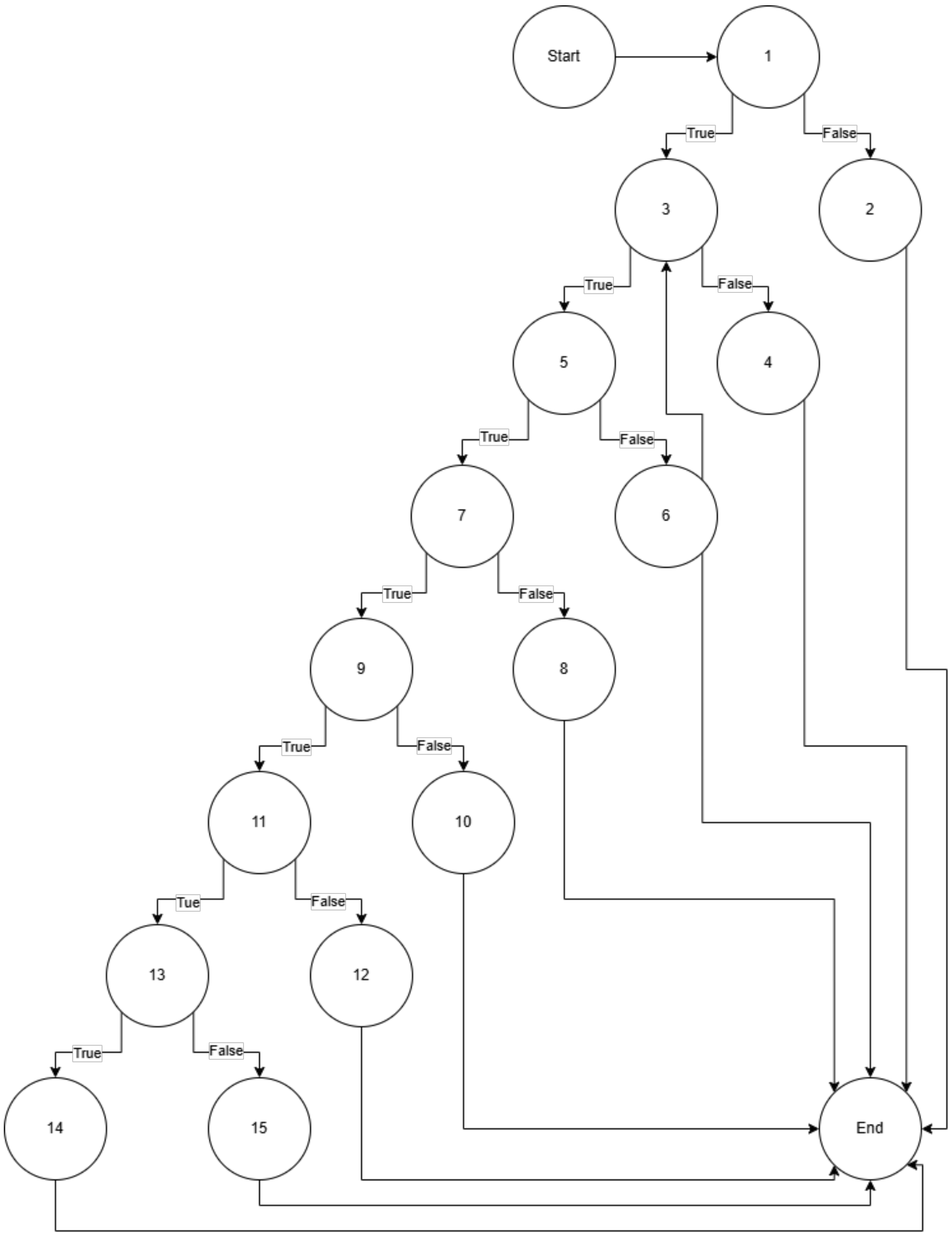


Figure 32: Control Flow Graph for isSpecSymbol

Test Path	**Test Data**	**Expected Output**
1, 2	`c = ’(’`	`true`
1, 3, 4	`c = ’)’`	`true`
1, 3, 5, 6	`c = ’[’`	`true`
1, 3, 5, 7, 8	`c = ’]’`	`true`
1, 3, 5, 7, 9, 10	`c = ’’’`	`true`
1, 3, 5, 7, 9, 11, 12	`c = ’‘’`	`true`
1, 3, 5, 7, 9, 11, 13, 14	`c = ’,’`	`true`
1, 3, 5, 7, 9, 11, 13, 15	`c = ’x’`	`false`

Table 16: Test Cases for `isSpecSymbol`

2 Program Tests

Test Case 1

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,3 \rightarrow$ *openCharacterStream*$[1,3,4]],8 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "file.txt", EOF
- **Output:** ""

Test Case 2

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,2 \rightarrow$ *openCharacterStream*$[1,2,4]],8 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "", EOF
- **Output:** ""

Test Case 3

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,3 \rightarrow$ *openCharacterStream*$[1,3,4]],8 \rightarrow$ *getToken*$[1,5 \rightarrow$ *isSpecSymbol*$[1],15],9,10 \rightarrow$ *printToken*$[1,6 \rightarrow$ *printSpecSymbol*$[1,3]],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "file.txt", "("
- **Output:** "lparen."

Test Case 4

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,3 \rightarrow$ *openCharacterStream*$[1,3,4]],8 \rightarrow$ *getToken*$[1,6 \rightarrow$ *isKeyword*$[1],9 \rightarrow$ *ungetChar*$[1],11 \rightarrow$ *isTokenEnd*$[1,10 \rightarrow$ *isSpecSymbol*$[1],11,12],15],9,10 \rightarrow$ *printToken*$[1,2 \rightarrow$ *tokenType*$[1,7],3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "file.txt", "if lambda"
- **Output:** "keyword,"if".
keyword,"lambda"."

Test Case 5

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,2 \rightarrow$ *openCharacterStream*$[1,2,4]],8 \rightarrow$ *getToken*$[1,6 \rightarrow$ *isIdentifier*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,2 \rightarrow$ *tokenType*$[1,3],3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "", ""abc""
- **Output:** "identifier,"abc"."

Test Case 6

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,3 \rightarrow$ *openCharacterStream*$[1,3,4]],8 \rightarrow$ *getToken*$[1,5 \rightarrow$ *isSpecSymbol*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,6 \rightarrow$ *printSpecSymbol*$[1,3]],11 \rightarrow$ *getToken*$[1,5 \rightarrow$ *isSpecSymbol*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "file.txt", "()"
- **Output:** "lparen.
rparen."

Test Case 7

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,3 \rightarrow$ *openCharacterStream*$[1,3,4]],8 \rightarrow$ *getToken*$[1,6 \rightarrow$ *isStrConstant*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,2 \rightarrow$ *tokenType*$[1,5],3],11 \rightarrow$ *getToken*$[1,6 \rightarrow$ *isNumConstant*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,2 \rightarrow$ *tokenType*$[1,4],3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$
- **Input:** fname = "file.txt", ""HelloWorld" 123"
- **Output:** "string,"HelloWorld".
numeric,123."

Test Case 8

- **Path:** main[1, 2, 6, 7 $\rightarrow$ *openTokenStream*$[1,2 \rightarrow$ *openCharacterStream*$[1,2,4]],8 \rightarrow$ *getToken*$[1,6 \rightarrow$ *isNumConstant*$[1],9 \rightarrow$ *ungetChar*$[1],15],9,10 \rightarrow$ *printToken*$[1,2 \rightarrow$ *tokenType*$[1,4],3],11 \rightarrow$ *getToken*$[1,2,9 \rightarrow$ *ungetChar*$[1],15],9,12]$

- **Input:** fname = "", "123"
- **Output:** "numeric,123."

Test Case 9

- **Path:** main[1, 2, 6, 7 $\rightarrow$ $openTokenStream[1,2 \rightarrow openCharacterStream[1,2,4]], 8 \rightarrow getToken[1,10 \rightarrow isComment[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7], 3], 11 \rightarrow getToken[1,2,9 \rightarrow ungetChar[1], 15], 9, 12]$
- **Input:** fname = "", ";comment"
- **Output:** "comment, ";comment"."

Test Case 10 Test Case 10

- **Path:** main[1, 2, 6, 7 $\rightarrow$ $openTokenStream[1,2 \rightarrow openCharacterStream[1,2,4]], 8 \rightarrow getToken[1,10 \rightarrow isCharConstant[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,6], 3], 11 \rightarrow getToken[1,2,9 \rightarrow ungetChar[1], 15], 9, 12]$
- **Input:** fname = "", "#a"
- **Output:** "character,"a"."

Test Case 11

- **Path:** main[1, 2, 6, 7
$\rightarrow openTokenStream[1,2 \rightarrow openCharacterStream[1,2,4]], 8 \rightarrow getToken[1,6 \rightarrow isKeyword[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]], 3], 11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]], 11 \rightarrow getToken[1,6 \rightarrow isIdentifier[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,3 \rightarrow isIdentifier[1]], 3], 11 \rightarrow getToken[1,6 \rightarrow isNumConstant[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,4 \rightarrow isNumConstant[1]], 3], 11 \rightarrow getToken[1,6 \rightarrow isStrConstant[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,5 \rightarrow isStrConstant[1]], 3], 11 \rightarrow getToken[1,10 \rightarrow isComment[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isComment[1]], 3], 11 \rightarrow getToken[1,6 \rightarrow isKeyword[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]], 3], 11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1,6 \rightarrow$
printSpecSymbol[1, 3]], 11, 12]
- **Input:** fname = "",
"if (x 123 "HelloWorld"
;comment
lambda)"
- **Output:**"keyword,"if".
lparen.
identifier,"x".
numeric,123.
string,"HelloWorld".
comment,";comment".
keyword,"lambda".
rparen."

Test Case 12

- **Path:** main[1, 2, 6, 7
$\rightarrow openTokenStream[1,2 \rightarrow openCharacterStream[1,2,4]], 8 \rightarrow getToken[1,6 \rightarrow isKeyword[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]], 3], 11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]], 11 \rightarrow getToken[1,6 \rightarrow isKeyword[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]], 3], 11 \rightarrow getToken[1,6 \rightarrow isIdentifier[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,3 \rightarrow isIdentifier[1]], 3], 11 \rightarrow getToken[1,6 \rightarrow isError[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1,2 \rightarrow tokenType[1,0], 3], 11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]], 11, 12]$

- **Input:** fname = ””, ”and‘and
  j
  112A)”
- **Output:** ”keyword,”and”.
  bquote.
  keyword,”and”.
  identifier,”j”.
  error,”112A”.
  rparen.”

Test Case 13

- **Path:** main[1, 2, 6, 7
  $\rightarrow openTokenStream[1,2 \rightarrow openCharacterStream[1,2,4]],8 \rightarrow getToken[1,10 \rightarrow$
  $isComment[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow$
  $isComment[1]],3],11 \rightarrow getToken[1,6 \rightarrow isKeyword[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow$
  $printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]],3],11 \rightarrow getToken[1,5 \rightarrow$
  $isSpecSymbol[1],15],9,10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow$
  $getToken[1,6 \rightarrow isIdentifier[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow$
  $tokenType[1,3 \rightarrow isIdentifier[1]],3],11 \rightarrow getToken[1,6 \rightarrow isNumConstant[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,4 \rightarrow$
  $isNumConstant[1]],3],11 \rightarrow getToken[1,6 \rightarrow isStrConstant[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,5 \rightarrow$
  $isStrConstant[1]],3],11 \rightarrow getToken[1,10 \rightarrow isComment[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow$
  $isComment[1]],3],11 \rightarrow getToken[1,6 \rightarrow isKeyword[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow$
  $printToken[1,2 \rightarrow tokenType[1,7 \rightarrow isKeyword[1]],3],11 \rightarrow getToken[1,5 \rightarrow$
  $isSpecSymbol[1],15],9,10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow$
  $getToken[1,6 \rightarrow isCharConstant[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow$
  $tokenType[1,6 \rightarrow isCharConstant[1]],3],11 \rightarrow getToken[1,6 \rightarrow isStrConstant[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,5 \rightarrow$
  $isStrConstant[1]],3],11 \rightarrow getToken[1,6 \rightarrow isNumConstant[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,4 \rightarrow$
  $isNumConstant[1]],3],11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1],15],9,10 \rightarrow$
  $printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow getToken[1,6 \rightarrow isIdentifier[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,3 \rightarrow$
  $isIdentifier[1]],3],11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1],15],9,10 \rightarrow$
  $printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow getToken[1,10 \rightarrow isComment[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow$
  $isComment[1]],3],11 \rightarrow getToken[1,6 \rightarrow isError[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow$
  $printToken[1,2 \rightarrow tokenType[1,0],3],11 \rightarrow getToken[1,6 \rightarrow isError[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,0],3],11 \rightarrow getToken[1,6 \rightarrow$
  $isKeyword[1],9 \rightarrow ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow$
  $isKeyword[1]],3],11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1],15],9,10 \rightarrow$
  $printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow getToken[1,5 \rightarrow$
  $isSpecSymbol[1],15],9,10 \rightarrow printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow$
  $getToken[1,5 \rightarrow isSpecSymbol[1],15],9,10 \rightarrow printToken[1,6 \rightarrow$
  $printSpecSymbol[1,3]],11 \rightarrow getToken[1,5 \rightarrow isSpecSymbol[1],15],9,10 \rightarrow$
  $printToken[1,6 \rightarrow printSpecSymbol[1,3]],11 \rightarrow getToken[1,10 \rightarrow isComment[1],9 \rightarrow$
  $ungetChar[1],15],9,10 \rightarrow printToken[1,2 \rightarrow tokenType[1,7 \rightarrow$
  $isComment[1]],3],11,12]$
- **Input:** fname = ””, ”; this is a comment
  if (x 123 ”hello” ;another comment
  lambda)
  #c ”string with spaces” 456 ‘ quote , comma ;comment
  unknown_token
  123abc

"unclosed_string
and [ ] ( ) ;end_comment"

- **Output:** "comment,"; this is a comment".
keyword,"if".
lparen.
identifier,"x".
numeric,123.
string,"hello".
comment,";another comment".
keyword,"lambda".
rparen.
character,'c'.
string,"string with spaces".
numeric,456.
bquote.
identifier,"quote".
comma.
identifier,"comma".
comment,";comment".
error,"unknown_token".
error,"123abc".
error,""unclosed_string".
keyword,"and".
lsquare.
rsquare.
lparen.
rparen.
comment,";end_comment"."

Test Case 14

- **Path:** main[1, 2, 6, 7 $\rightarrow openTokenStream[1, 2 \rightarrow openCharacterStream[1, 2, 4]], 8 \rightarrow getToken[1, 6 \rightarrow isIdentifier[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1, 2 \rightarrow tokenType[1, 3 \rightarrow isIdentifier[1]], 3], 11 \rightarrow getToken[1, 5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1, 6 \rightarrow printSpecSymbol[1, 3]], 11 \rightarrow getToken[1, 5 \rightarrow isSpecSymbol[1], 15], 9, 10 \rightarrow printToken[1, 6 \rightarrow printSpecSymbol[1, 3]], 11 \rightarrow getToken[1, 6 \rightarrow isStrConstant[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1, 2 \rightarrow tokenType[1, 5 \rightarrow isStrConstant[1]], 3], 11 \rightarrow getToken[1, 6 \rightarrow isError[1], 9 \rightarrow ungetChar[1], 15], 9, 10 \rightarrow printToken[1, 2 \rightarrow tokenType[1, 0], 3], 11, 12]$
- **Input:** fname = "", ""string" ;
' "extr"a""
- **Output:** "identifier,"string".
comment,";".
quote.
string,"extr".
error,"a"."