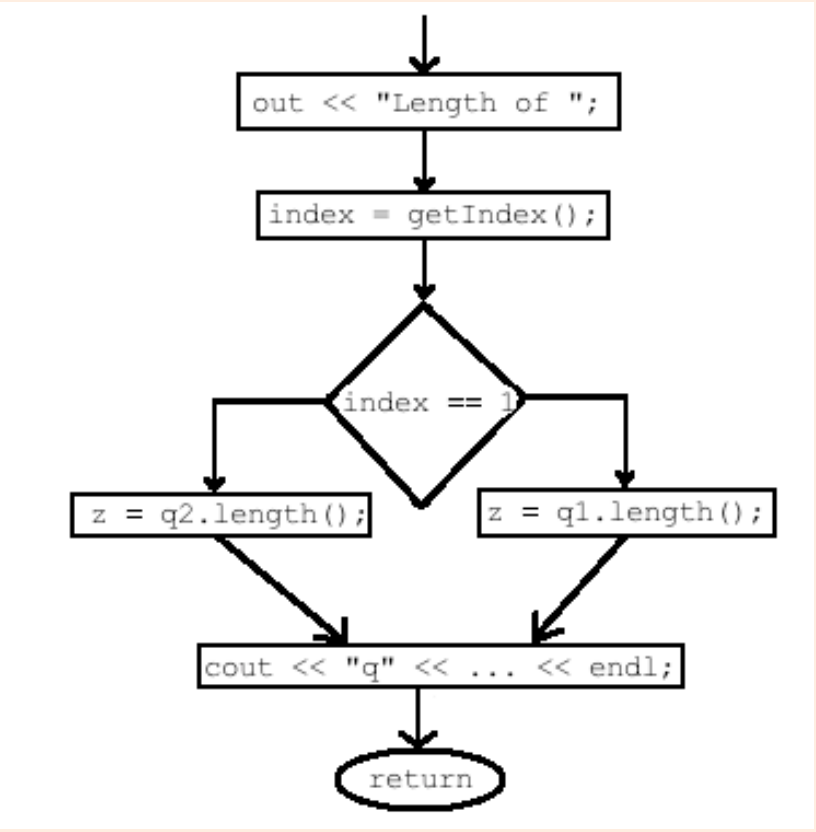
Code-Based Testing

Testing the *doLength* Operation

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Systematic Testing



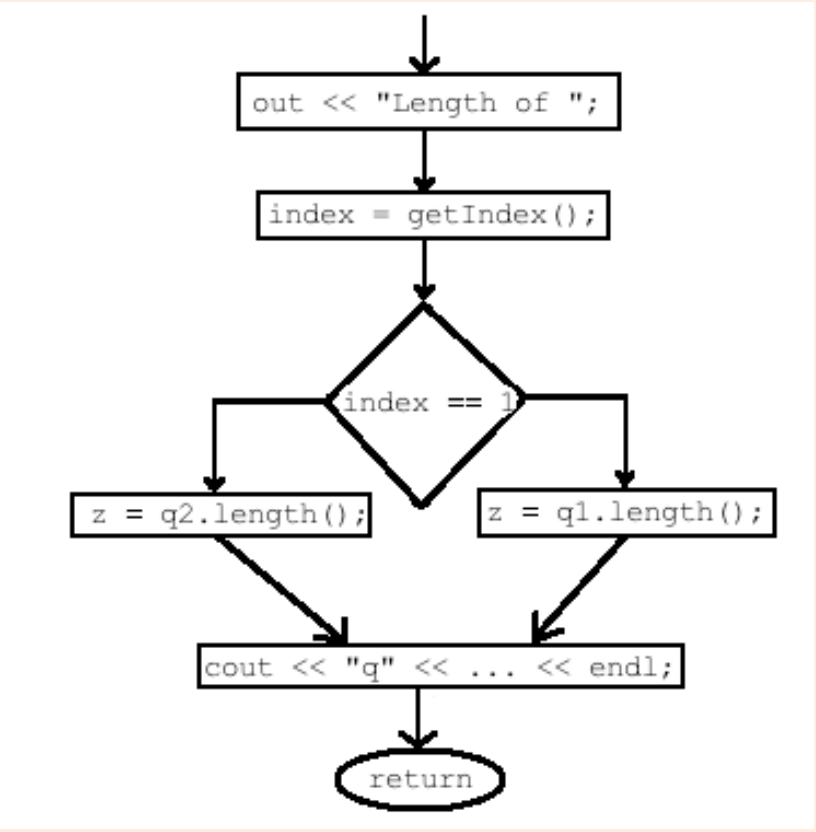
Code-based systematic testing:

- Relies on inspection of the code
- Develop test cases that systematically exercise various parts of the code
- Three types of testing that we will look at:
  1. Statement Coverage
  2. Branch Coverage
  3. Path Coverage

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

A Test Case



A *test case* comprises two parts determined before the test is executed

1. A legal input
2. An allowable result

doLength test case #1:

- *Legal input*: index = 1, q1 = < >, q2 = <33>
- *Allowable result*: 0

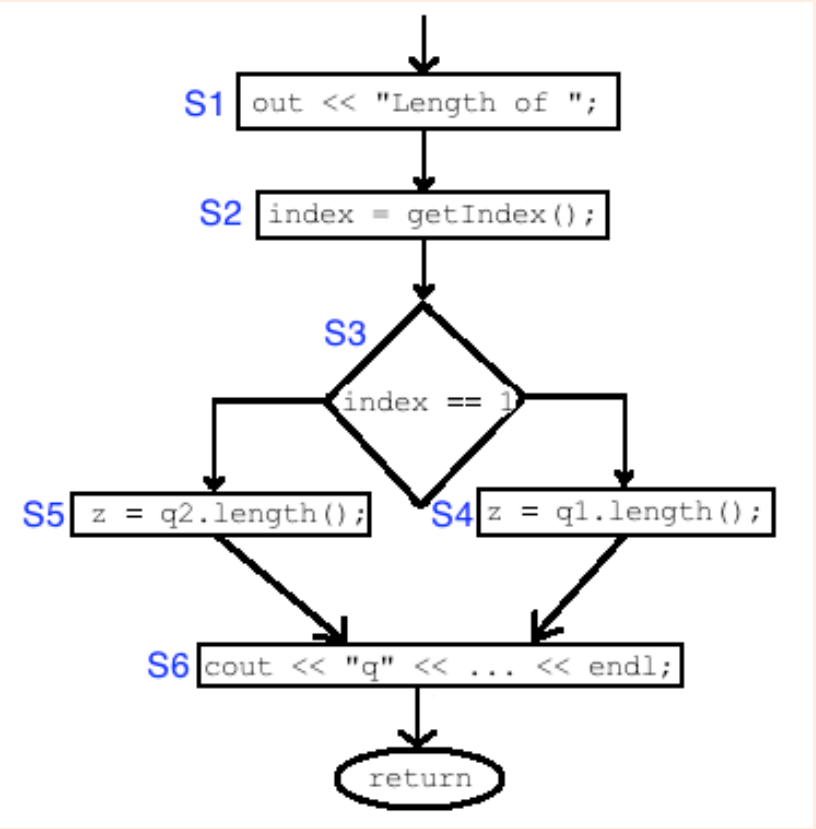
doLength test case #2:

- *Legal input*: index = 2, q1 = < >, q2 = <33>
- *Allowable result*: 1

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Statement Coverage



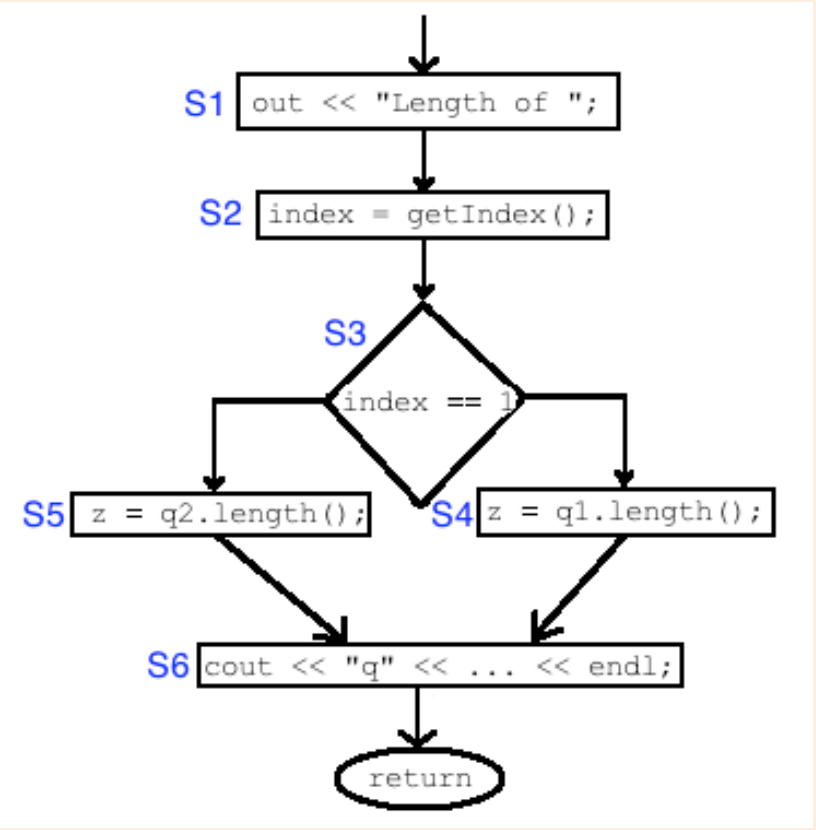
Statement Coverage Testing

- All statements in the code under test will be exercised by a set of test cases
- *doLength* (below) has statements **S1** – **S6**

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";                          // S1
   index = getIndex();                            // S2
   if (index == 1) {                              // S3
      z = q1.length();                            // S4
   } else {
      z = q2.length();                            // S5
   }  // end if
   cout << "|q" << index << "| = " << z << endl; // S6
}  // doLength
```

Statement Coverage



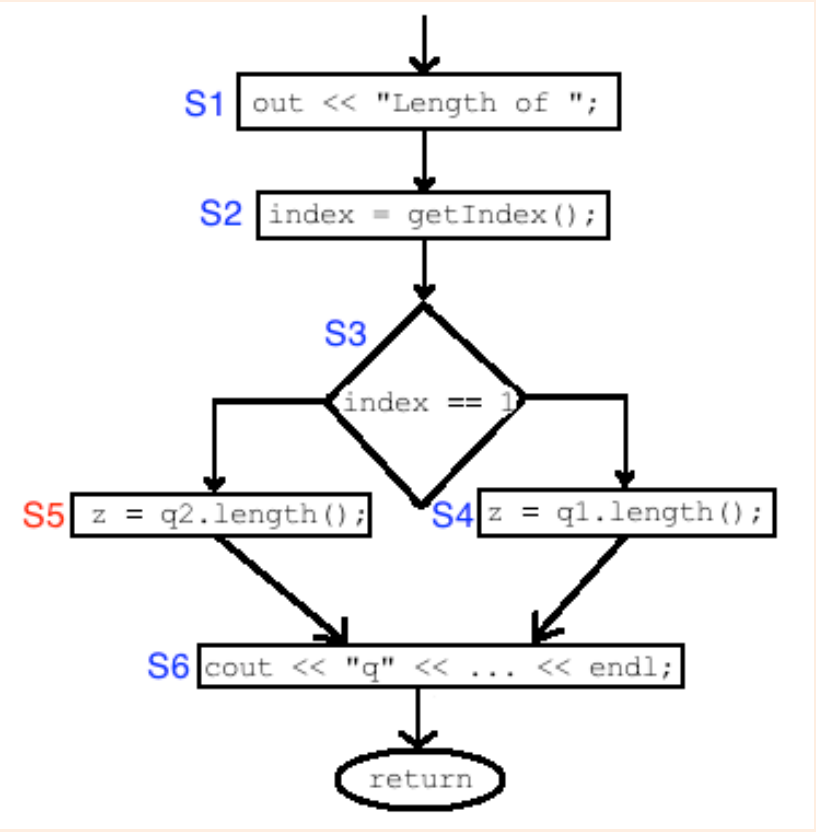
Statement Coverage Testing

- To achieve statement coverage for *doLength* requires 2 test cases

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";                          // S1
   index = getIndex();                            // S2
   if (index == 1) {                              // S3
      z = q1.length();                            // S4
   } else {
      z = q2.length();                            // S5
   }  // end if
   cout << "|q" << index << "| = " << z << endl; // S6
}  // doLength
```

Statement Coverage



Statement Coverage Testing

- Test Case #1:

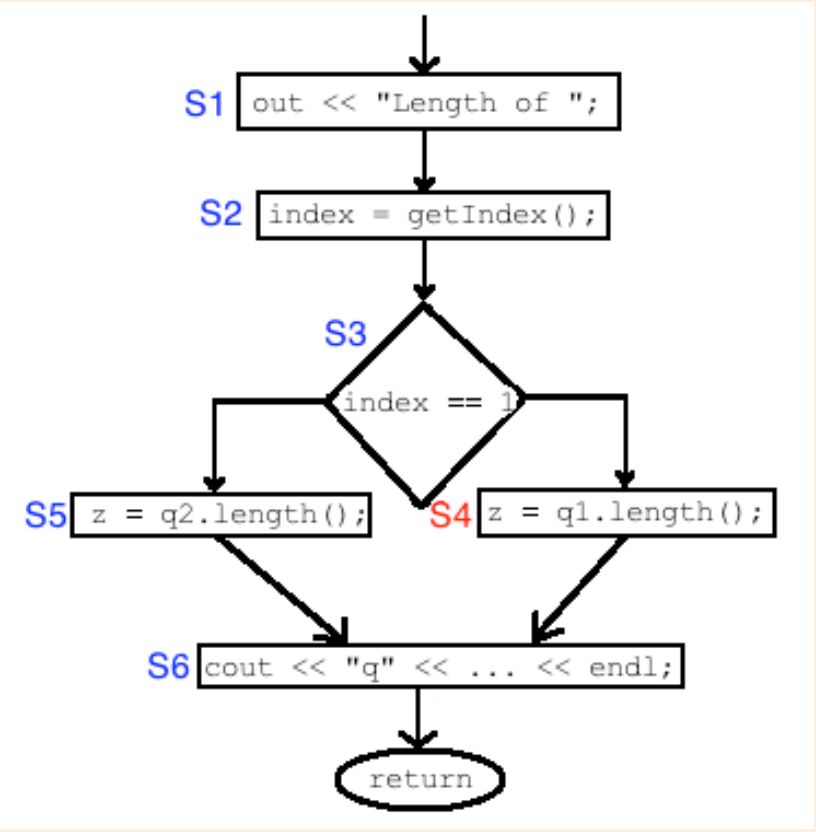
  Where index = 1
  covers S1, S2, S3, S4, S6

  Statement **S5** is not covered by this test case

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";                          // S1
   index = getIndex();                            // S2
   if (index == 1) {                              // S3
      z = q1.length();                            // S4
   } else {
      z = q2.length();                            // S5
   }  // end if
   cout << "|q" << index << "| = " << z << endl; // S6
}  // doLength
```

Statement Coverage



Statement Coverage Testing

- Test Case #2:

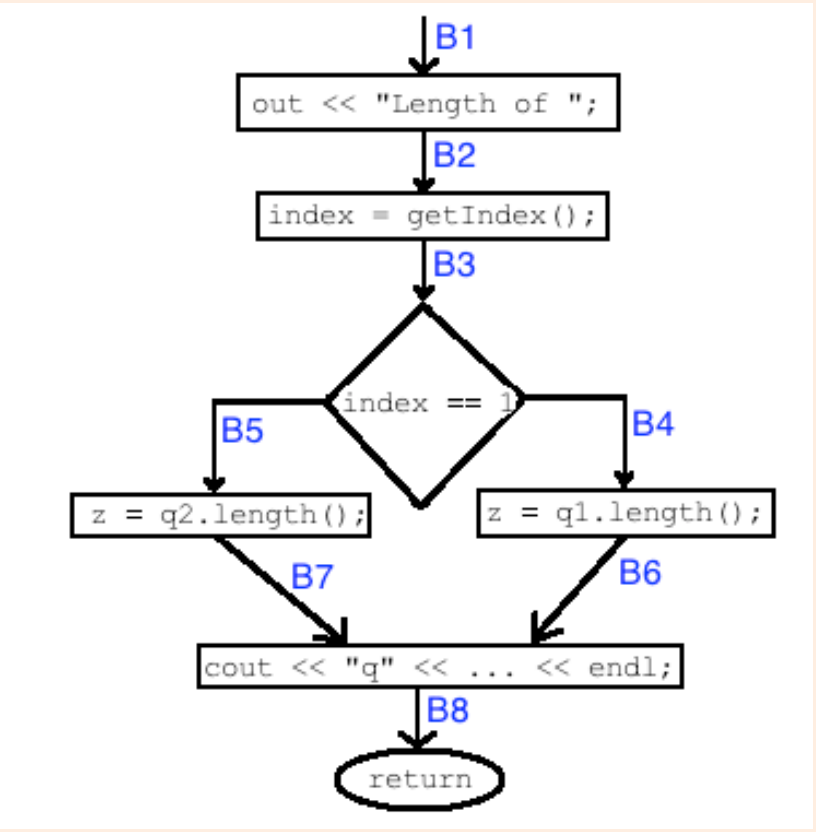
  Where index ≠ 1
  covers S1, S2, S3, S5, S6

  Statement **S4** is not covered by this test case

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";                          // S1
   index = getIndex();                            // S2
   if (index == 1) {                              // S3
      z = q1.length();                            // S4
   } else {
      z = q2.length();                            // S5
   }  // end if
   cout << "|q" << index << "| = " << z << endl; // S6
}  // doLength
```

Branch Coverage



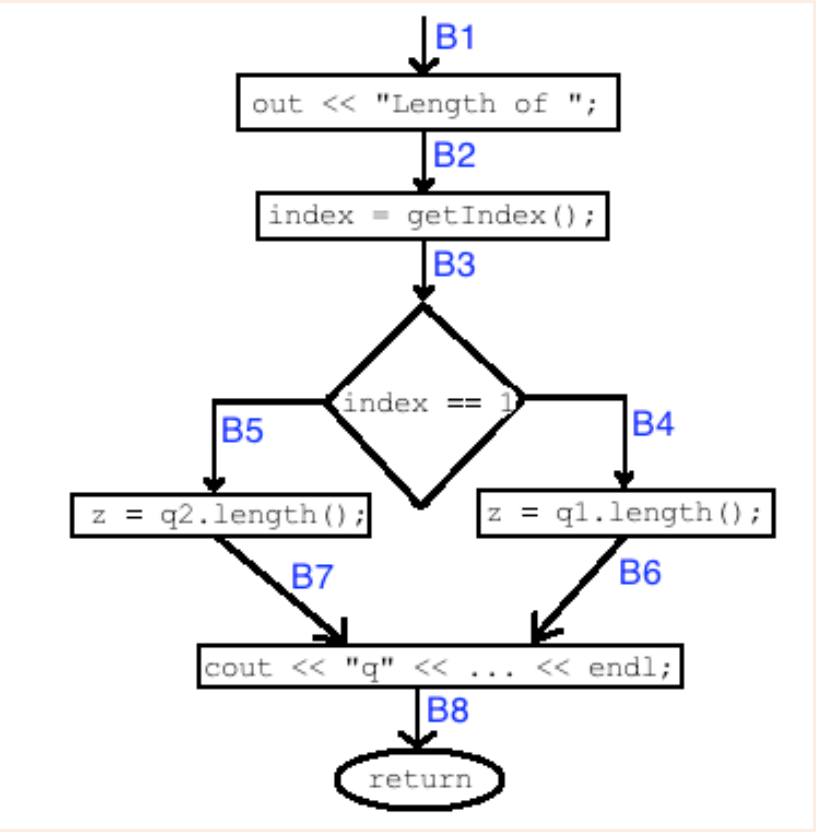
Branch Coverage Testing

- All branches in the code under test will be traversed by a set of test cases
- *doLength* (diagramed to the left) has branches **B1** – **B8**

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Branch Coverage



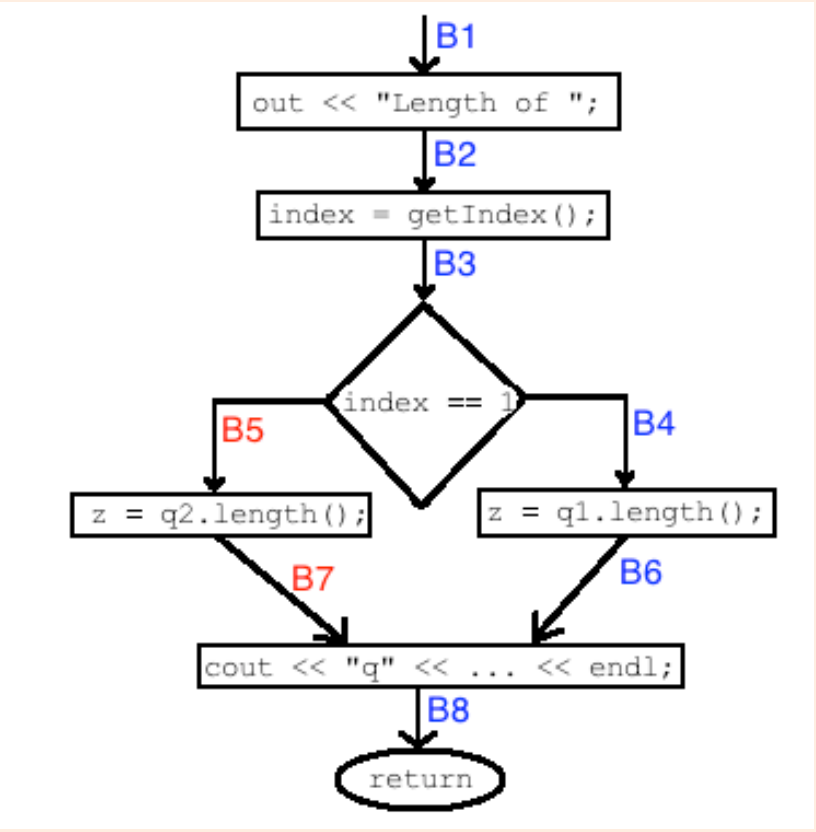
Branch Coverage Testing

- To achieve branch coverage for *doLength* requires 2 test cases

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Branch Coverage



Branch Coverage Testing

- Test Case #1:

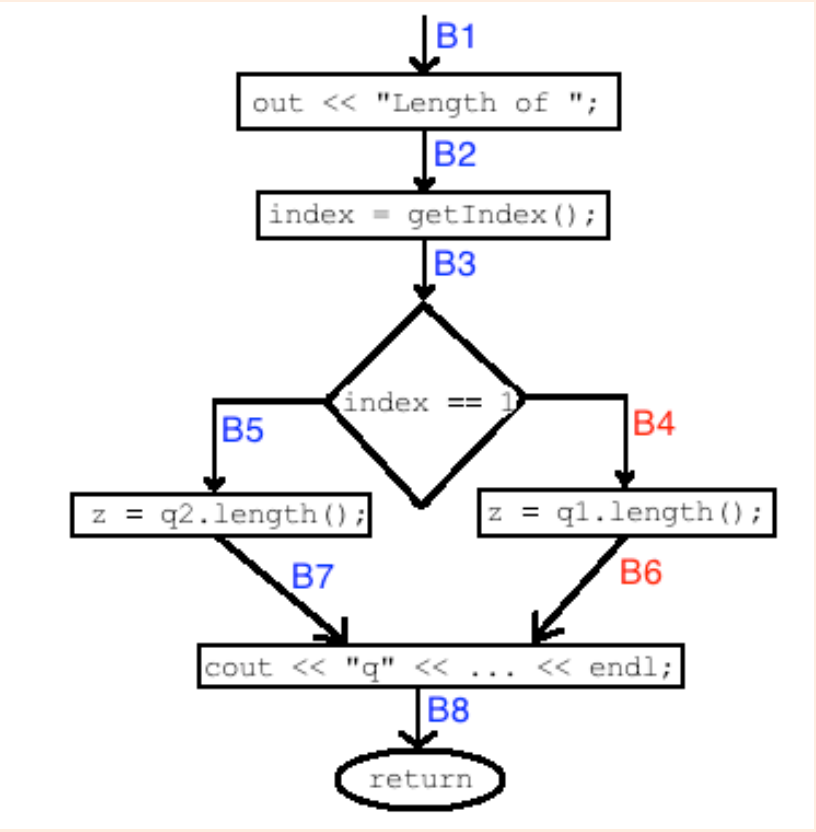
  Where index = 1
  covers B1, B2, B3, B4, B6, B8

  Branches **B5** and **B7** are not covered by this test case

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Branch Coverage



Branch Coverage Testing

- Test Case #2:

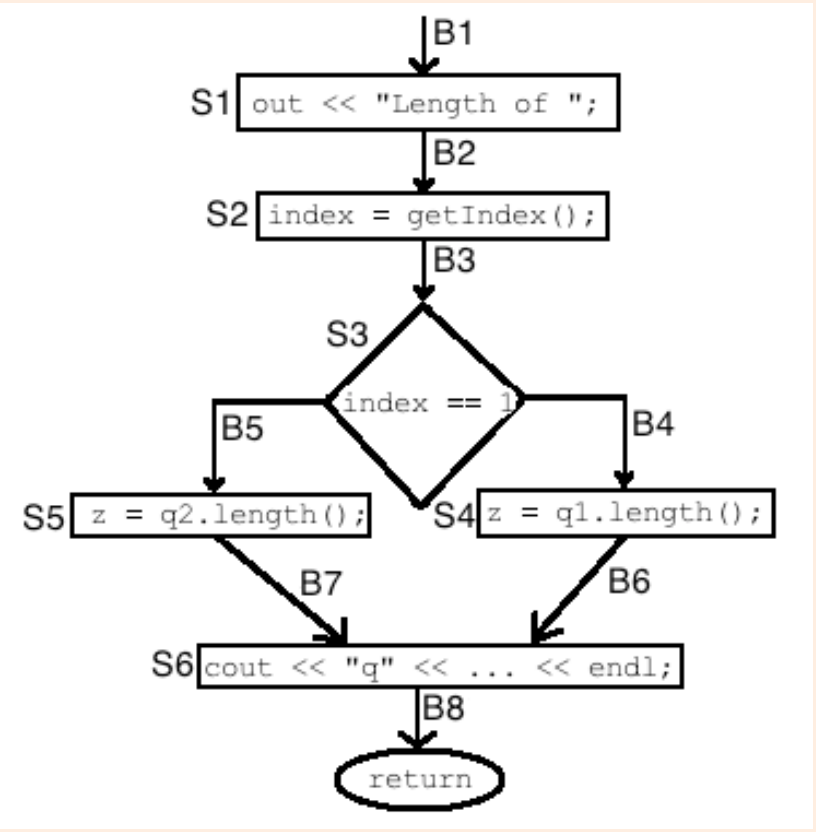
  Where index ≠ 1
  covers B1, B2, B3, B5, B7, B8

  Branches **B4** and **B6** are not covered by this test case

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Branch Coverage



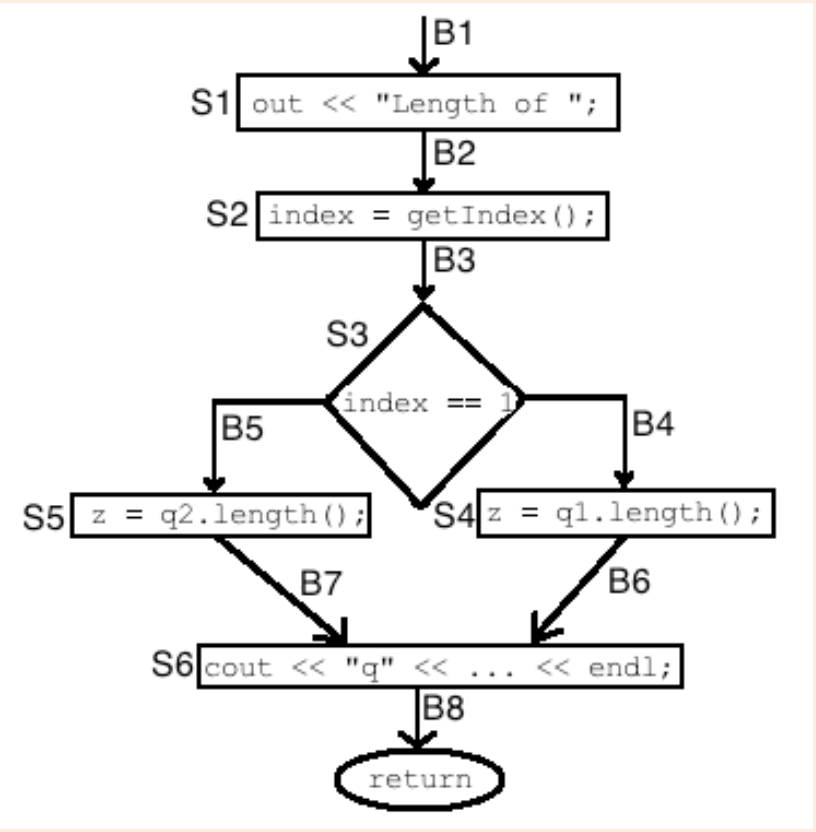
Branch Coverage ≠ Statement Coverage

- For *doLength* it turns out that a set of test cases to achieve statement coverage also achieves branch coverage
- This is a coincidence
- It is not always the case that statement coverage will also achieve branch coverage
- Often additional test cases are required to achieve branch coverage

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage

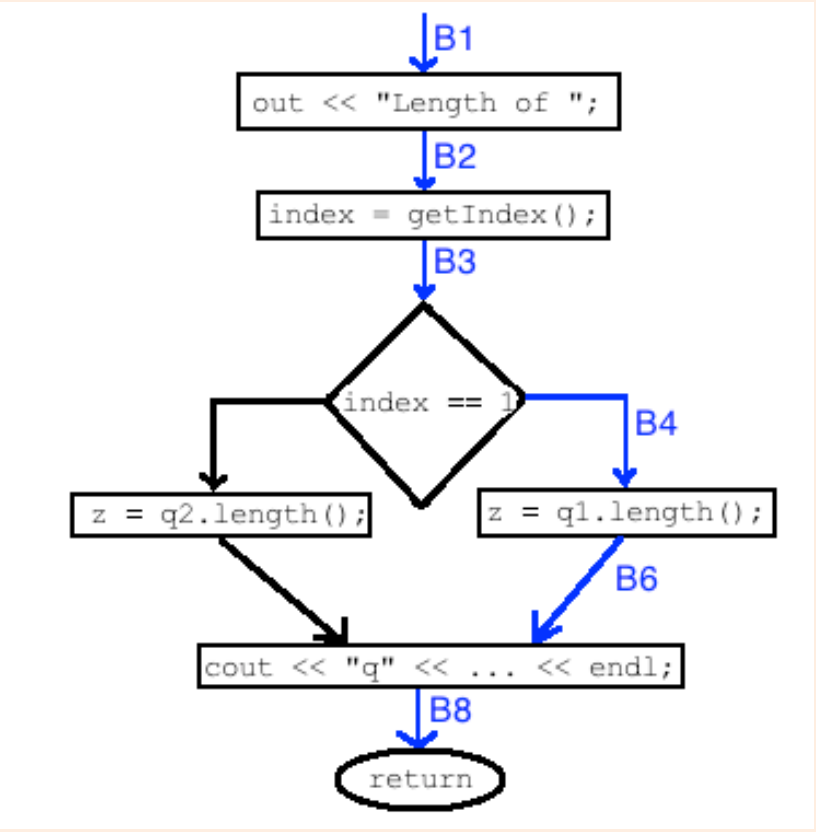


A *path* comprises all the branches taken on one particular input

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage



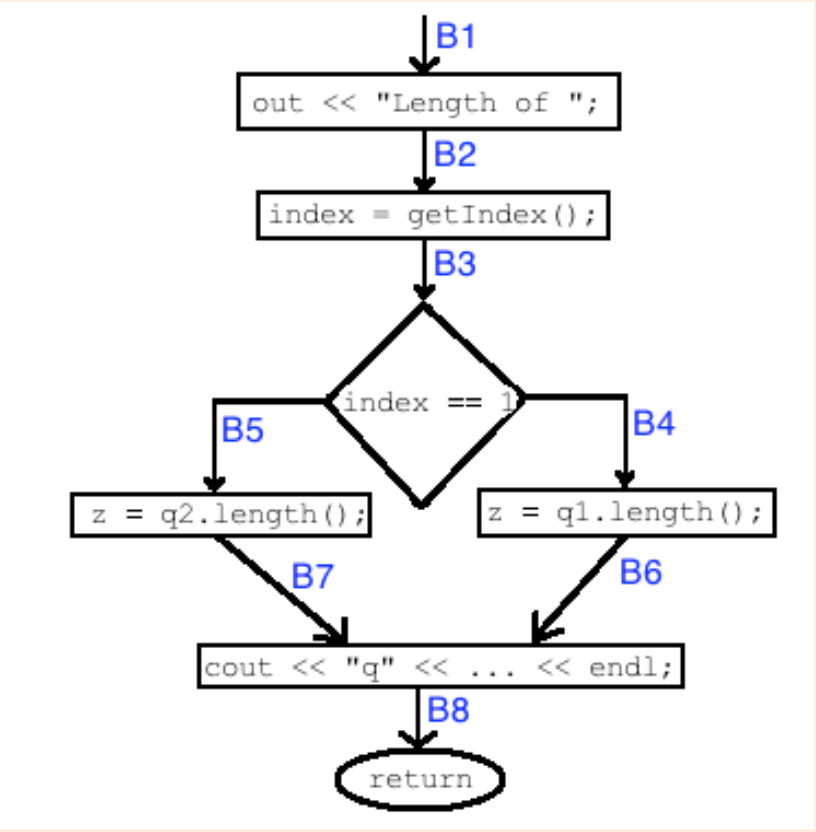
For example, the path when index = 1 is:

B1, B2, B3, B4, B6, B8

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage



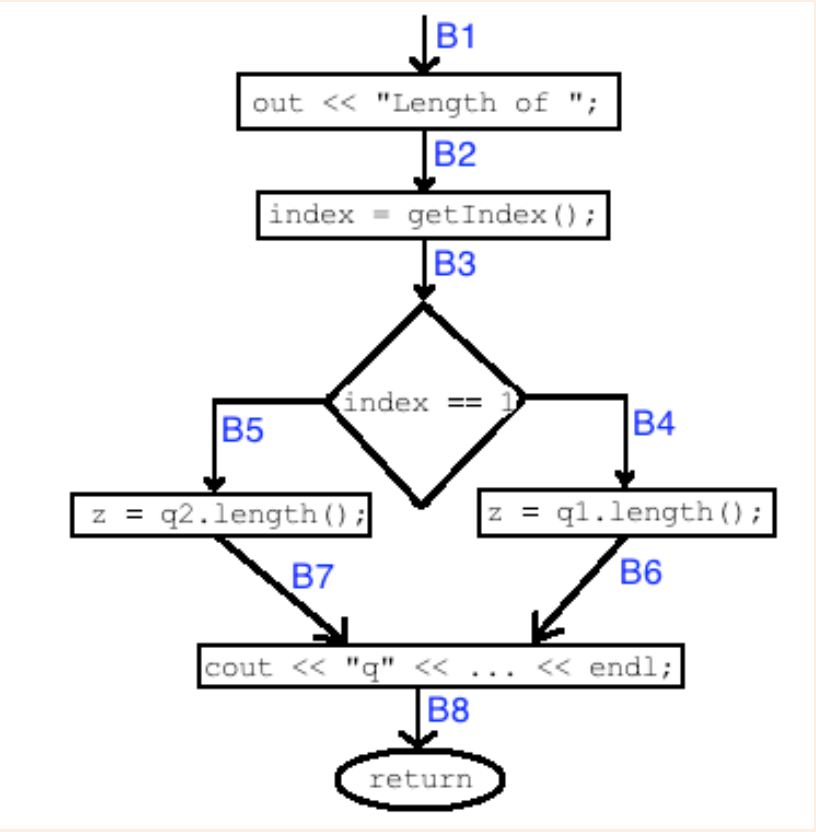
Path Coverage Testing

To achieve path coverage, *all paths* through the code under test must be traversed by a set of test cases

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage



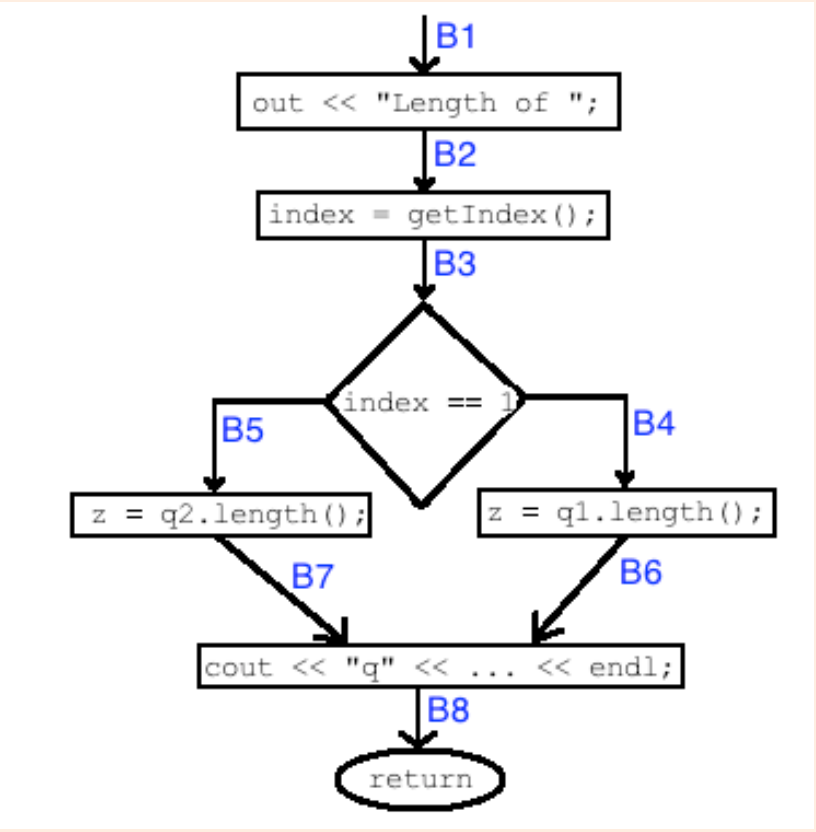
Path Coverage Testing

When a loop is present, each trip around the loop is considered a different path

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage



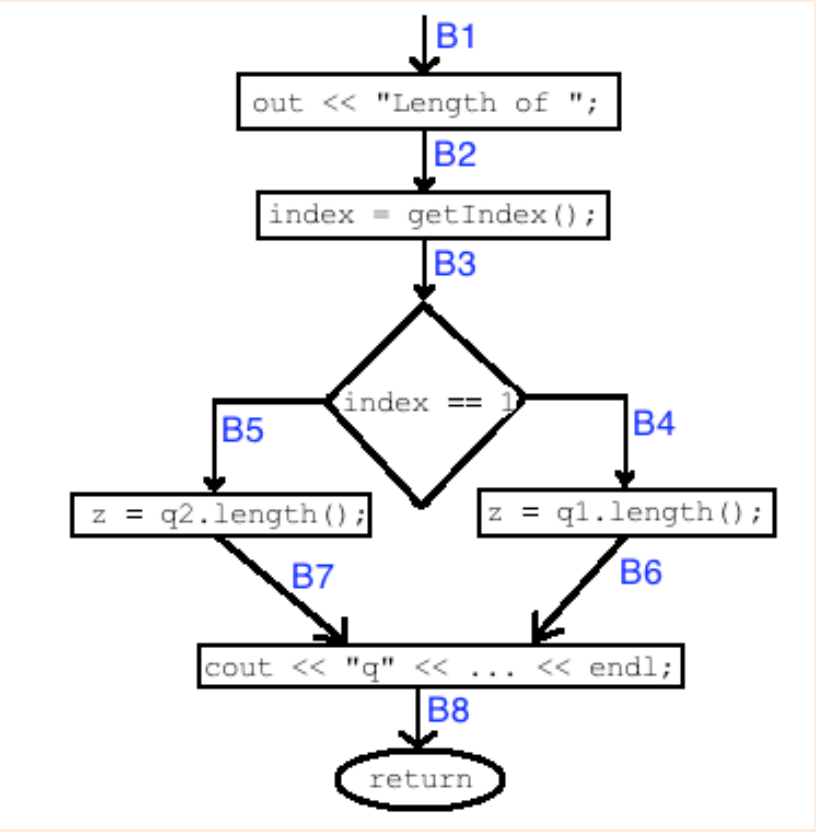
Path Coverage Testing

- *doLength* has 2 paths:
  1. B1, B2, B3, B4, B6, B8 (index = 1)
  2. B1, B2, B3, B5, B7, B8 (index ≠ 1)

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Path Coverage



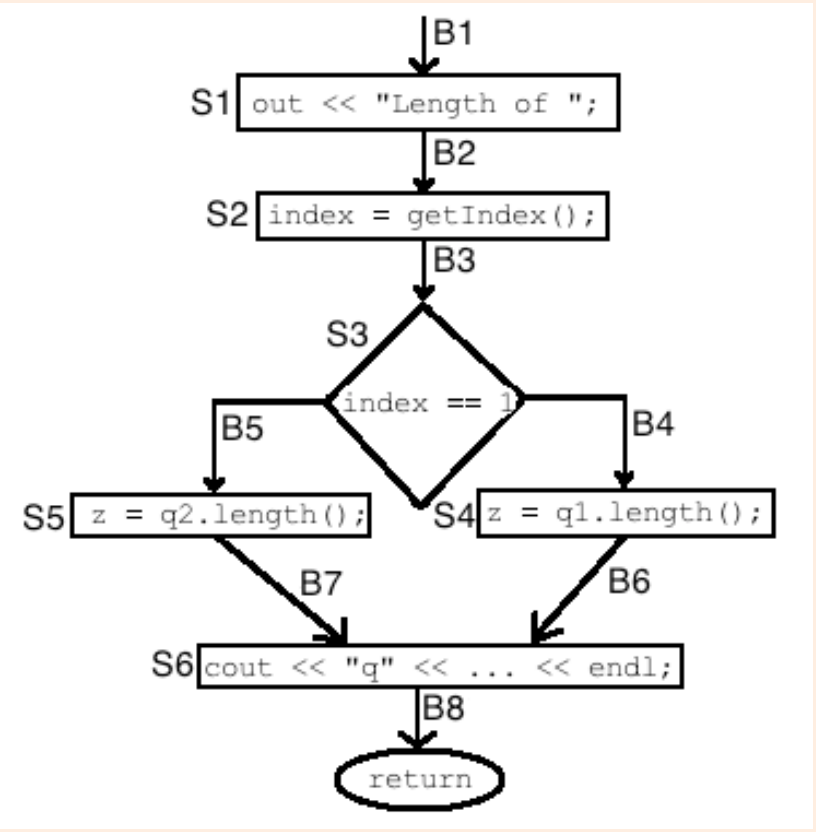
Path Coverage ≠ Branch Coverage

- For *doLength* it turns out that a set of test cases to achieve statement or branch coverage also achieves path coverage
- This is a coincidence
- It is not always the case that statement/branch coverage will also achieve path coverage
- Often additional test cases are required to achieve path coverage – this fact makes path coverage hard to achieve for code containing a loop

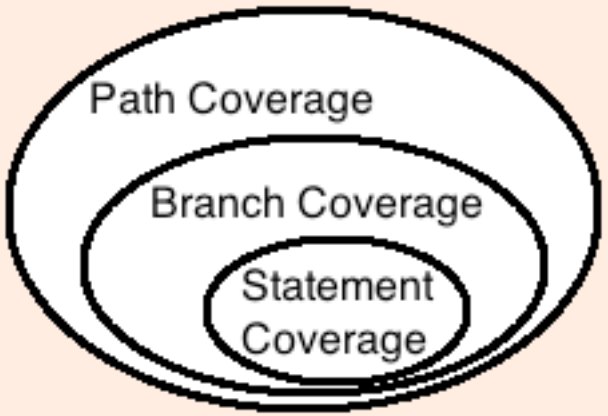
```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

Coverage Hierarchy



Statement ≤ Branch ≤ Path

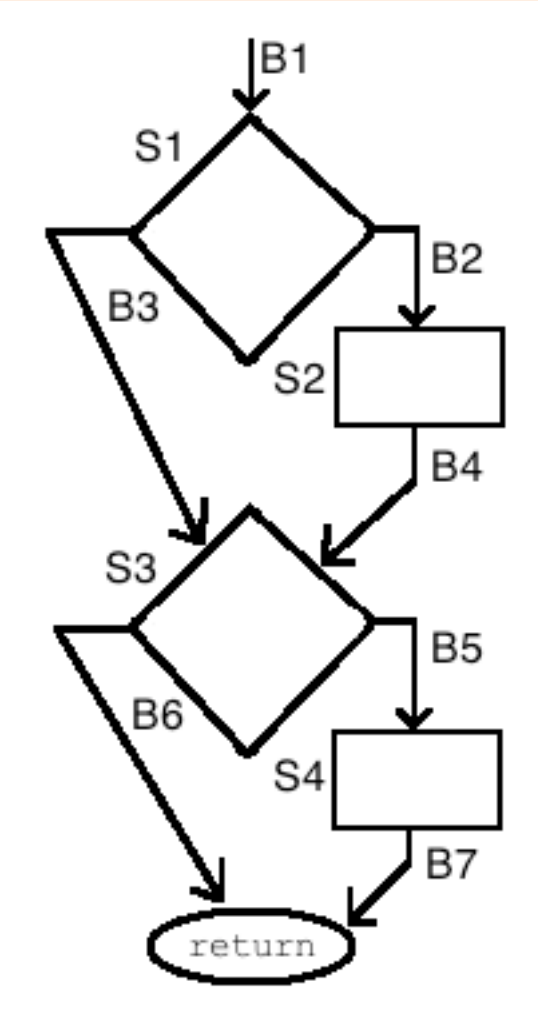


doLength is a simple example, where:
statement = branch = path

```
void doLength(QueueOfText& q1, QueueOfText& q2)
{
   Integer index, z;

   cout << "Length of ";
   index = getIndex();
   if (index == 1) {
      z = q1.length();
   } else {
      z = q2.length();
   }  // end if
   cout << "|q" << index << "| = " << z << endl;
}  // doLength
```

A More Interesting Example

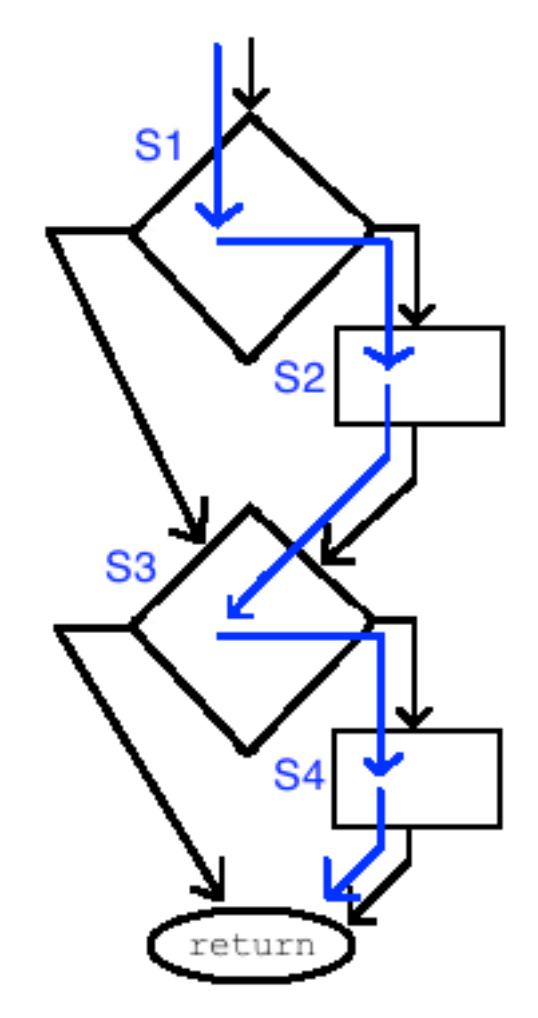


Minimum # of test cases needed:

- Statement coverage: 1
- Branch coverage: 2
- Path coverage: 4

```
void moreInterestingOp(...)
{
   if (...) {
      ...;
   } // end if
   if (...) {
      ...;
   } // end if
}  // moreInterestingOp
```

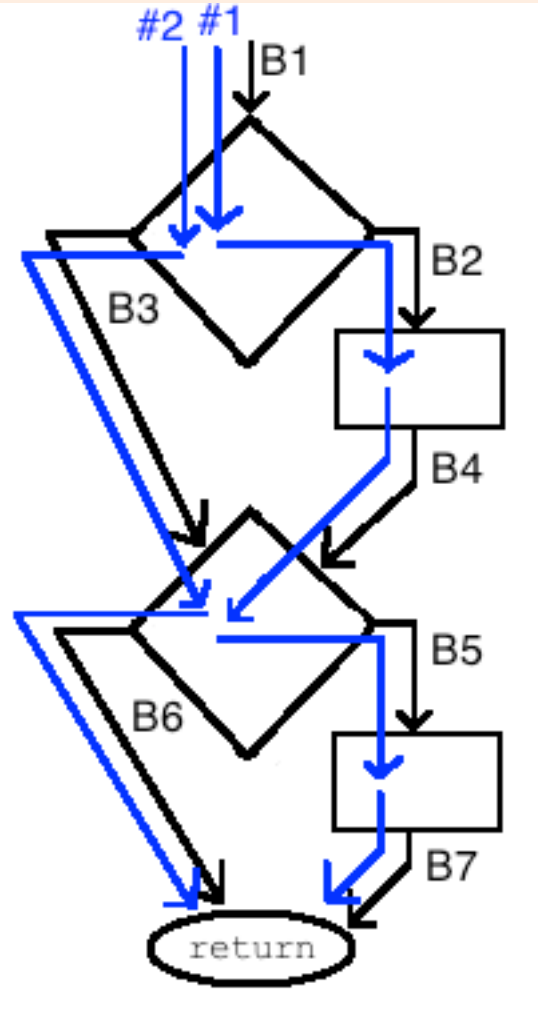
Statement Coverage



- Achieved with 1 test case where both conditionals evaluate to true
- Covers: S1, S2, S3, S4

```
void moreInterestingOp(...)
{
   if (...) {                    // S1
      ...;                       // S2
   } // end if
   if (...) {                    // S3
      ...;                       // S4
   } // end if
}  // moreInterestingOp
```

Branch Coverage



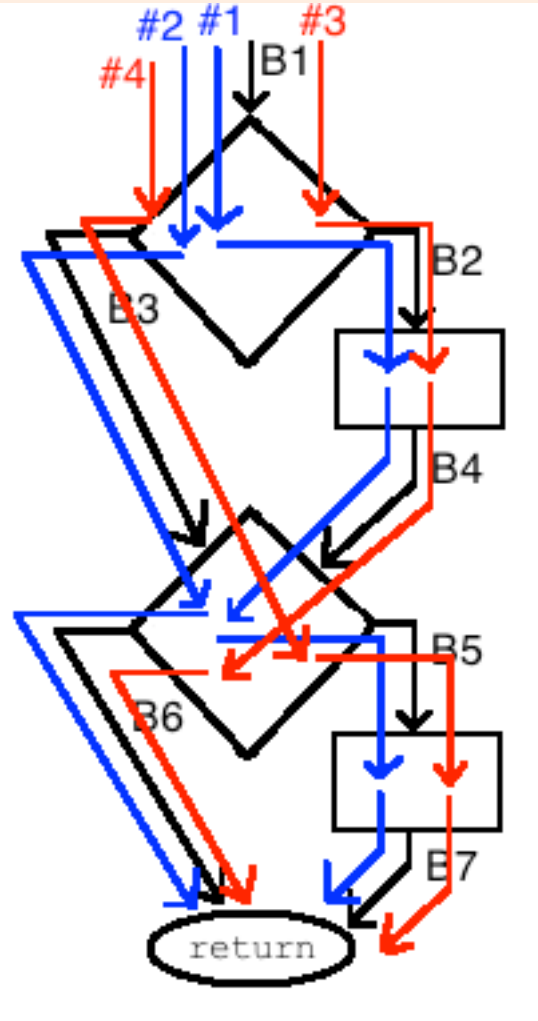
- Achieved with 2 test cases:
  1. Both conditionals evaluate to true
     #1) *B1*, *B2*, *B4*, *B5*, *B7*
  2. Both conditionals evaluate to false
     #2) *B1*, *B3*, *B6*

- The above is just one example:

  Other sets of 2 test cases are possible in order to achieve branch coverage

```
void moreInterestingOp(...)
{
   if (...) {
      ...;
   } // end if
   if (...) {
      ...;
   } // end if
}  // moreInterestingOp
```

Path Coverage

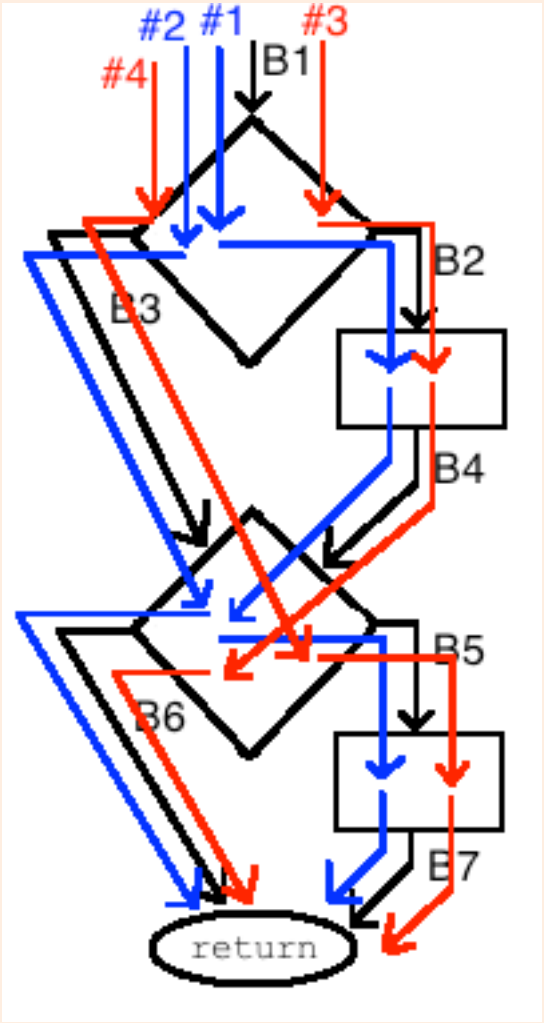


Achieved with 4 test cases:

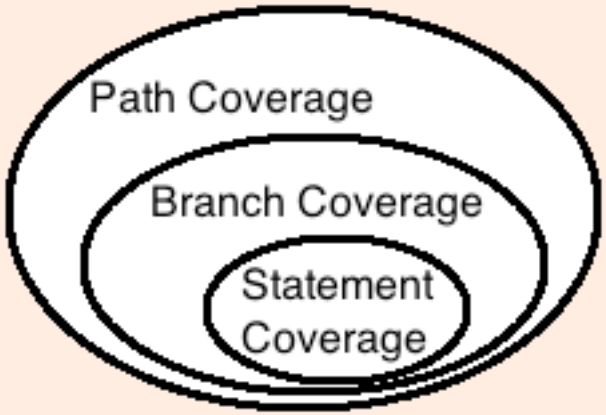
1. Both conditionals evaluate to true
   #1) *B1*, *B2*, *B4*, *B5*, *B7*
2. Both conditionals evaluate to false
   #2) *B1*, *B3*, *B6*
3. 1st conditional evaluates to true, 2nd conditional evaluates to false
   #3) *B1*, *B2*, *B4*, *B6*
4. 1st conditional evaluates to false, 2nd conditional evaluates to true
   #4) *B1*, *B3*, *B5*, *B6*

```
void moreInterestingOp(...)
{
   if (...) {
      ...;
   } // end if
   if (...) {
      ...;
   } // end if
}  // moreInterestingOp
```

Coverage Hierarchy



Statement ≤ Branch ≤ Path



moreInterestngOp is a more complex example, where:
statement < branch < path

```
void moreInterestingOp(...)
{
   if (...) {
      ...;
   } // end if
   if (...) {
      ...;
   } // end if
}  // moreInterestingOp
```