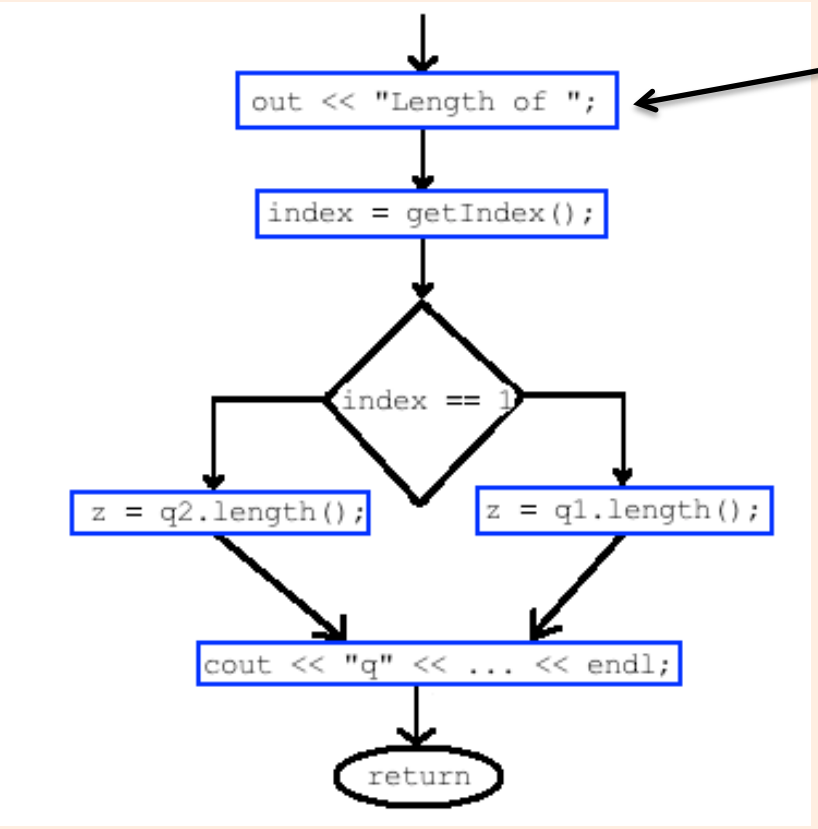
Diagramming Code

Diagramming *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
```

Diagramming Statements



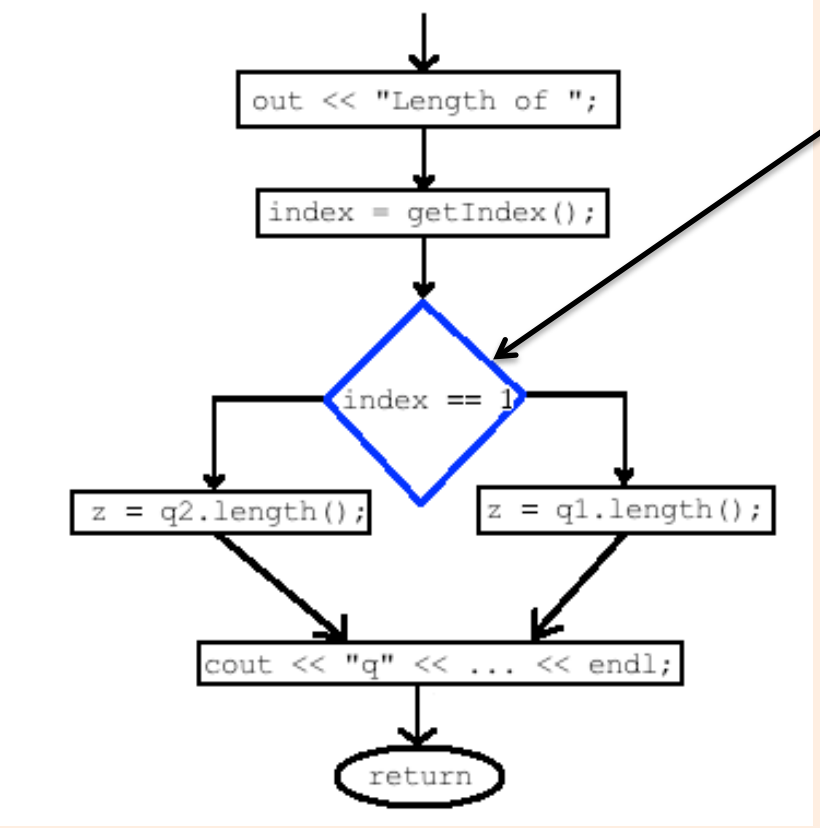
Rectangles are used for statements

- Except conditionals

```
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
```

Diagramming Conditionals



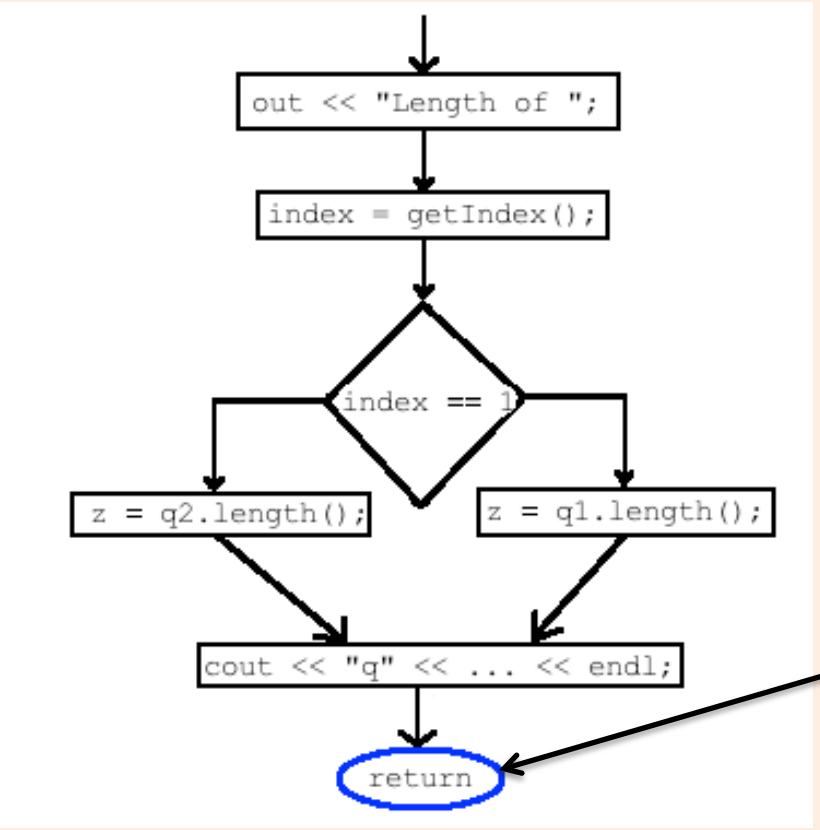
Diamonds are used for the conditional part of:

- an *if*
- a *while*
- a *for*
- a *do-while*

```
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
```

Diagramming Implicit Return



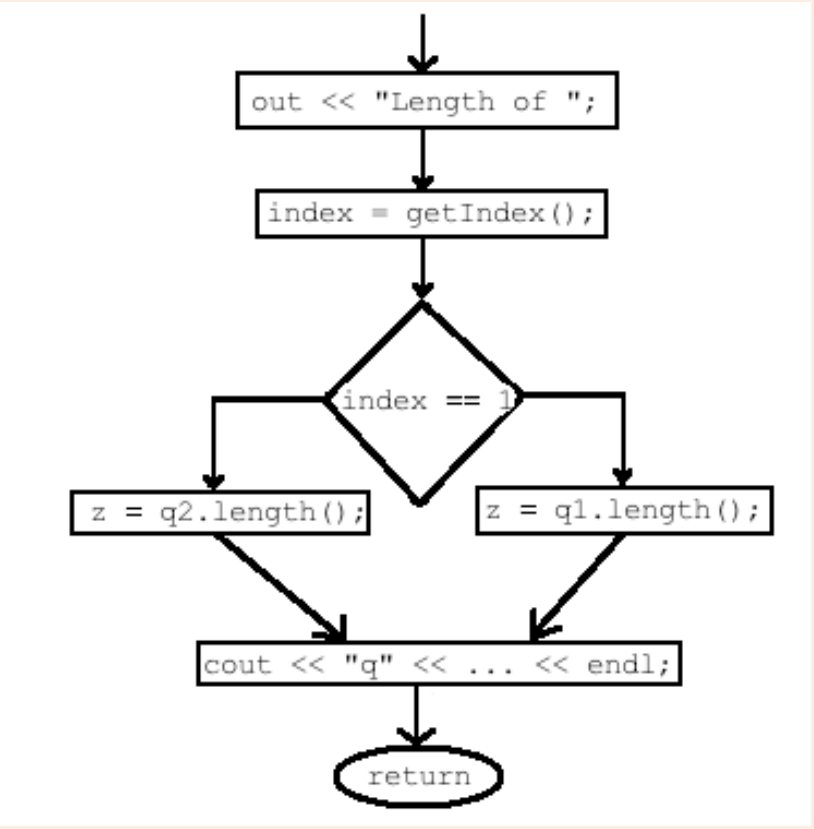
An **oval** is used for an implicit *return*

- When the *return* appears explicitly in the code of a function, then use a rectangle

```
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
```

What About Declarations?

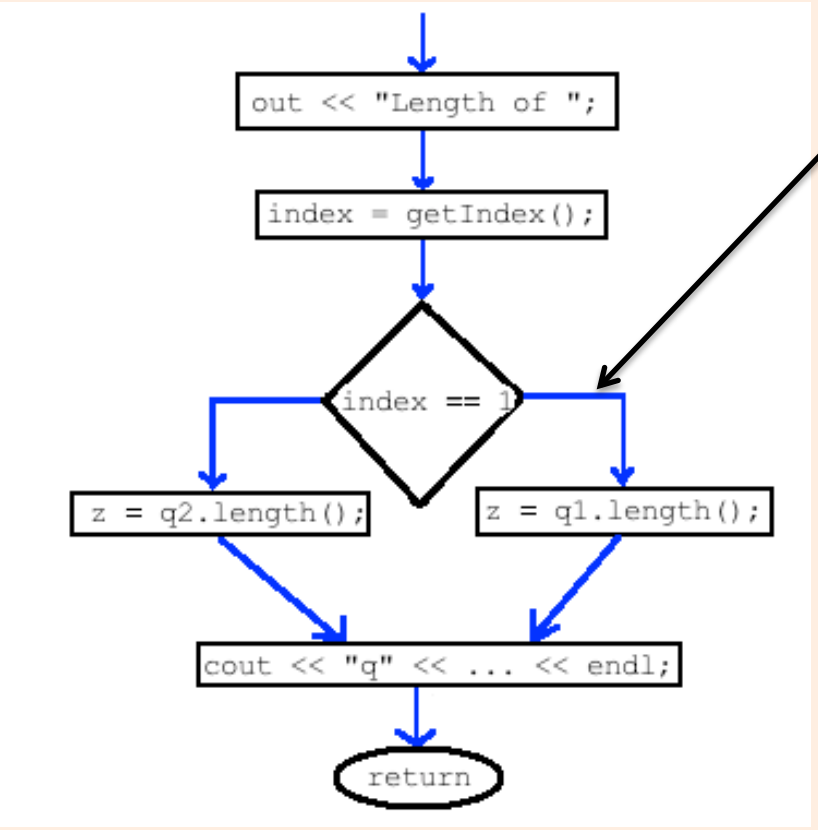


Do not diagram declarations

```
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
```

Diagramming Flow of Control



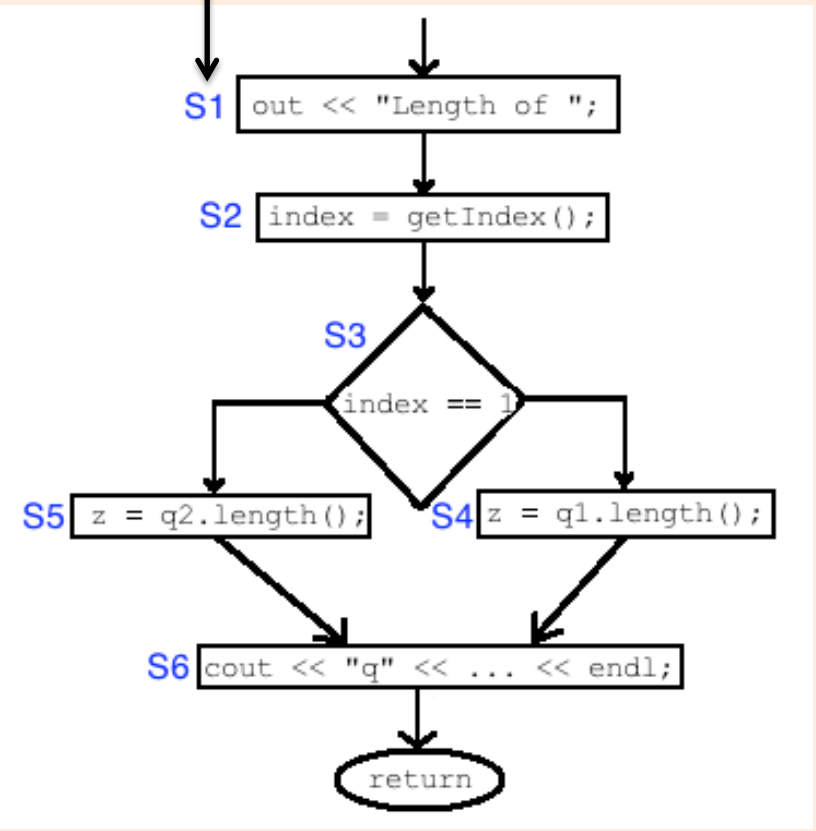
Use **arrows** to diagram the *flow of control* (*branch*)

- Arrows point downward in the diagram except in a loop which has one arrow that points back up higher in the diagram
- Conditionals always have two branches emanating from the diamond, one branch for *true* the other for *false*
- *One-way-out* statements have only one tail of an arrow adjacent to the statement
- *One-way-in* statements have only one arrow head adjacent to the statement

```
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
```

Labeling Statements

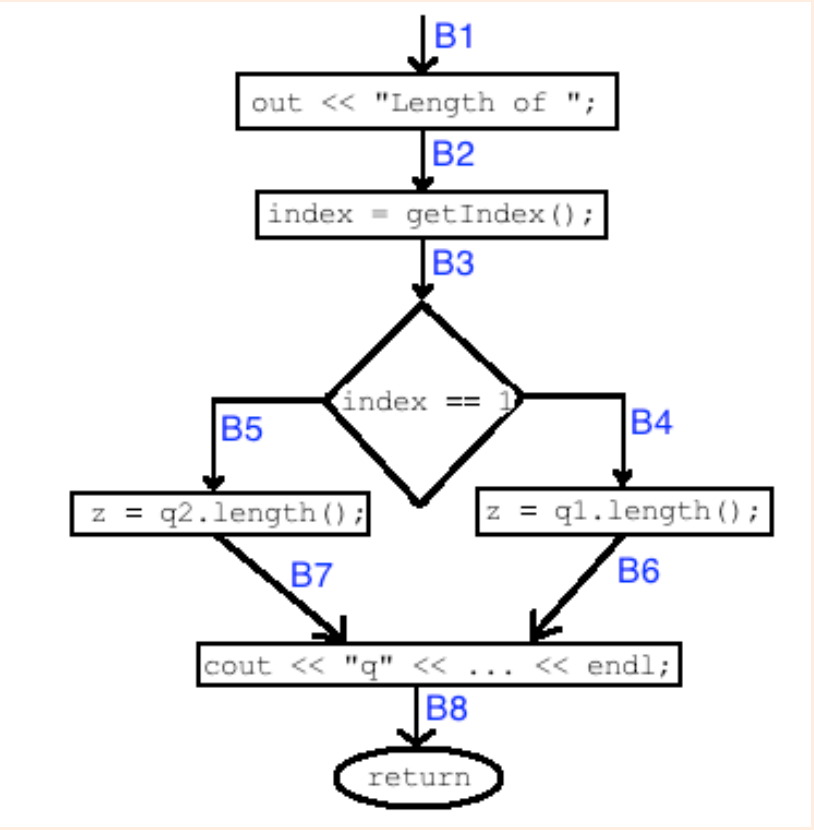


Label statements with **S1**, **S2**, **S3**, …
Begin labeling with the first executable line

```
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
```

Labeling Branches

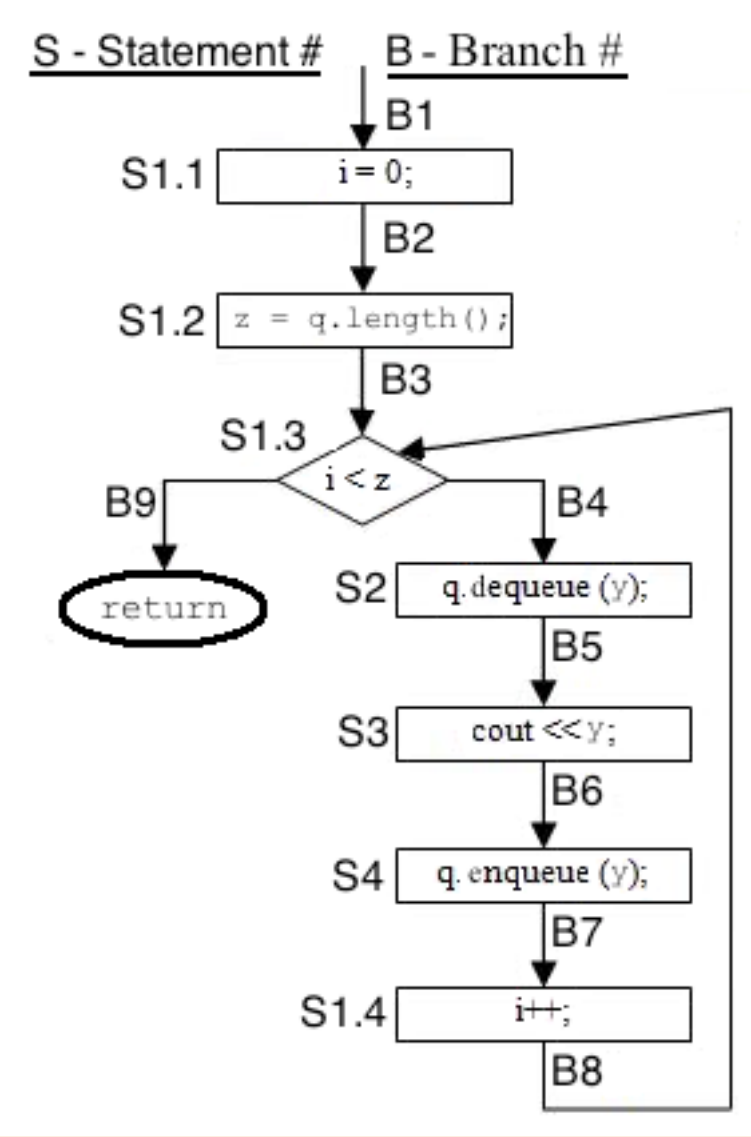


Label branches with **B1**, **B2**, **B3**, …
Begin labeling with the branch leading into statement S1

```
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
```

Example: *for* Loop



The *for* statement must be broken down into sub-parts

```
for(initialization; conditional; increment) { … }
```

- *initialization*: **S1.1** & **S1.2** executed one time before executing the conditional
- *conditional*: **S1.3** executed at the beginning of each pass through the loop
- *increment*: **S1.4** executed as the last statement of the loop body

```
void display(QueueOfText& q)
{
   Text y;

   //    S1.1        S1.2             S1.3    S1.4
   for (int i = 0, z = q.length(); i < z; i++){ // S1
      q.dequeue(y);                             // S2
      cout << y;                                // S3
      q.enqueue(y);                             // S4
   }  // end for
}  // display
```