Additional Control Structures

- The `switch` statement – an alternative for multiway branches.
- The `do-while` statement – an alternative for looping.
- The `for` statement – another alternative for looping.
- The `break` and `continue` statements – `break` and `continue` are to a control structure as `return` is to a function.

The `switch` Statement (A Template)

```
switch (letter)
{
   case 'X': Statement1;
             break;
   case 'L':
   case 'M': Statement2;
             break;
   case 'S': Statement3;
             break;
   default : Statement4;
}
Statement5;
```


The `switch` Statement (An Example)

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
             break;
   case 'C': cout << "Average Work";
             break;
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
             break;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```

The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `switch` Statement (An Incorrect Example, grade = 'C')

```
switch (grade)
{
   case 'A':
   case 'B': cout << "Good Work";
   case 'C': cout << "Average Work";
   case 'D':
   case 'F': cout << "Poor Work";
             numberInTrouble++;
   default : cout << grade << " is not a valid letter"
                  << " grade;
}
```


The `do-while` Statement

DoWhileStatement
do
  Statement
while (Expression)**;**

Example

```
do
{
   cout << "Enter your age: " ;
   cin >> age;
   if (age <= 0)
      cout << "Your age must be positive." << endl;
} while (age <= 0);
```


The `for` Statement

ForStatement
For (InitStatement Expression1; Expression2)
  Statement

Example

```
for (lastNum = 1; lastNum <= 7; lastNum++)
{
   for (numtoPrint = 1; numtoPrint <= lastNum; numToPrint++)
      cout << numToPrint;
   cout << endl;
}
```

Note: The InitStatement can be the null statement and the two Expressions are optional.

The `break` and `continue` Statements

- The `break` statement causes an immediate exit from the innermost `switch`, `while`, `do-while`, or `for` statement in which it appears.
- If `break` is in a loop that is nested inside another loop, control exits the inner loop but not the other.
- A `continue` statement terminates only the current iteration of a loop (not the whole loop).

An Example Using `break` and `continue`

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```

An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


An Example Using `break` and `continue` (num1 = 200, num2 = 55)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue` (num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue` (num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue` (num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```

Example Using `break` and `continue`

(num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue`

(num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue`

(num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue`

(num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Example Using `break` and `continue`

(num1 = 2, num2 = 55, loopCount = 15)

```
loopCount = 1;
while (true)
{
   cin >> num1;
   if (!cin || num1 >= 100)
      continue;
   if (!cin || num2 <= 50)
      continue;
   cout << sqrt(float(num1 + num2)) << endl;
   loopCount++
   if (loopCount > 10)
      break;
}
```


Guidelines for Choosing a Looping Statement

- For count-controlled loops, think `for`.
- If you know it will execute at least once, `do-while` is good for event control.
- If you don't know whether the loop will ever execute, use a pretest loop (`while` or `for`).
- When in doubt, use a `while` statement.

Testing and Debugging Hints

- Put a `break` statement at the end of each case alternative in a `switch` statement.
- Case labels in a `switch` statement are made up of values, not variables.
- A `switch` expression cannot be a floating-point or string expression.
- A `for` loop must have semicolons, whether the expressions exist or not.