Looping

- A loop executes the same statement (simple or compound) over and over, as long as a condition or set of conditions is satisfied.
- A loop is a control structure that causes a statement or group of statements to be executed repeatedly.

The `while` Statement

- WhileStatement
  While (Expression)
    Statement
- Example:

```
while (inputVal != 25)
   cin >> inputVal;
```

- If the expression has a value of false, the program skips the loop body and execution continues at the statement immediately following the loop.

Phases of Loop Execution

- Loop test – the point at which the while expression is evaluated.
- Loop entry – the point at which the flow of control reaches the first statement of the loop body
- Iteration – an individual repetition of the body of a loop
- Loop exit – the point at which control passes to the first statement following the loop
- Termination condition – the condition causing a loop exit

Loops Using the While Statement

- Count-controlled loop
  – Executes a specified number of times
- Event-controlled loop
  – Terminates when something changes the expression's value

Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10)
{
   .
   .
   .
   loopCount = loopCount + 1;
}
```


Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10) loop test
{
   .
   .
   .
   loopCount = loopCount + 1;
}
```

Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10)
{
   .            loop entry
   .
   .
   loopCount = loopCount + 1;
}
```


Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10)
{
   .                          iteration
   .
   .
   loopCount = loopCount + 1;
}
```


Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10)
{
   .
   .
   .
   loopCount = loopCount + 1;
}
loop exit
```


Count-Controlled Loops

- A loop control variable is used – it must be initialized before it is tested and it must be updated in the body of the loop.
- Example:

```
loopCount = 1;
while (loopCount <= 10) termination condition
                        is loopCount = 11
{
   .
   .
   .
   loopCount = loopCount + 1;
}
```


Event-Controlled Loops

- Sentinel-Controlled Loops
  - Execute until a special value is encountered
- End-of-File Controlled Loops
  - Execute until all data in a file has been read
- Flag-Controlled Loops
  - Execute until a flag changes value

Sentinel-Controlled Loops

- A special data value is used to signal the program that there is no more data to be processed.
- A sentinel value must be something that never shows up in the normal input to a program.

Sentinel-Controlled Loop Example

```
while (!(month == 2 && day == 31))
{
  cin >> month >> day;
  .
  .
  .
}
```

- Problem: `month` and `day` aren't initialized

More On Sentinel-Controlled Loops

```
cin >> month >> day;
while (!(month == 2 && day == 31))
{
   cin >> month >> day;
   .
   .
}
```


More On Sentinel-Controlled Loops

```
cin >> month >> day;
while (!(month == 2 && day == 31))
{
   cin >> month >> day;
   .
   .
}

cin >> month >> day;
while (!(month == 2 && day == 31))
{
   .
   .
    cin >> month >> day;
}
```


Infinite Loop Example

```
cin >> dataValue >> sentinel;
while (sentinel = 1)
{
  .
  .
  .
  cin >> dataValue >> sentinel;
}
```


End-of-File Controlled Loops

- After a program has read the last piece of data from an input file, the computer is at the end of the file (EOF).
- If we try to input any more values, the stream goes into the fail state.
- We can use the failure of the input stream as a sentinel.
- Example:

```
inData >> intVal;
while (inData)
{
   cout << intVal << endl;
   inData >> intVal;
}
```


More on End-of-File Loops

- Keep in mind that the input stream can fail even though we haven't reached the end of the file.
- EOF-controlled loops are similar to sentinel-controlled loops in that the program doesn't know in advance how many data items are to be input.
- It is possible to use an EOF-controlled loop when we read from the standard input device via the cin stream instead of a data file.

cin EOF Example

```
cout << "Enter an integer (or Ctrl-D to quit): ";
cin >> someInt;
while (cin)
{
     cout << someInt << " doubled is "
          << 2 * someInt << endl;
     cout << "Next number (or Ctrl-D to quit): ";
     cin >> someInt;
}
```


Flag-Controlled Loops

- A flag is a Boolean variable that is used to control the logical flow of a program.
- We can use the Boolean variable to record whether or not the event that controls the process has occurred.

Flag-Controlled Loop Example

```
sum = 0;
nonNegative = true;           // Initialize flag
while (nonNegative)
{
    cin >> number;
    if (number < 0)
       nonNegative = false;   // Test input value
    else                      // Set flag if event
       sum = sum + number;    // occurred.
}
```


An Equivalent Loop

```
sum = 0;
negative = false;             // Initialize flag
while (!negative)
{
    cin >> number;
    if (number < 0)
        negative = true;      // Test input value
    else                      // Set flag if event
        sum = sum + number;   // occurred.
}
```


Another Equivalent Loop

```
sum = 0;
cin >> number;

while (number > 0)            // Test input value
{
       sum = sum + number;
}
```