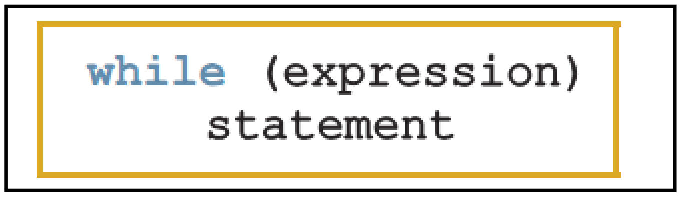
Chapter 5 - Control Flow - Loops

- Three looping (repetition) structures:
  - for
  - while
  - do…while

- How to construct and use :
  - Counter-controlled repetition structures
  - Sentinel-controlled repetition structures
  - Flag-controlled repetition structures
  - EOF-controlled repetition structures

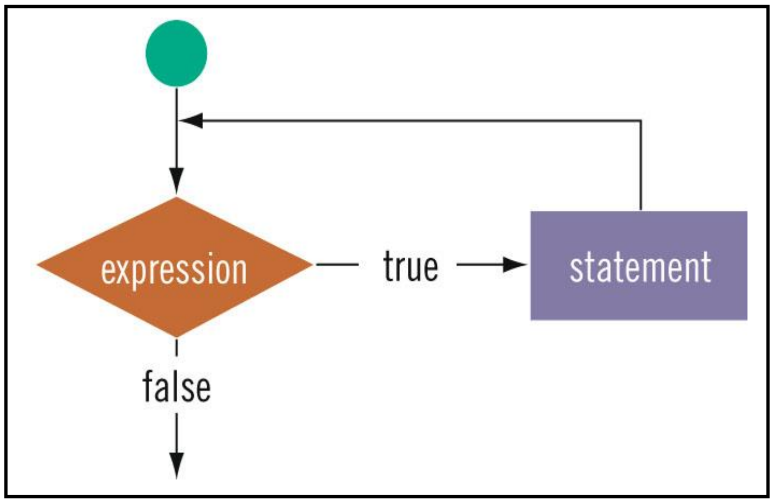
While Looping (Repetition) Structure

- Syntax of the while statement:



```
while (expression)
      statement
```

- **While** is reserve word
- Expression is called a **loop condition**
- Expression acts as a decision maker and is usually a logical expression
- Statement is called the body of the loop and can be simple or compound
- The parentheses are part of the syntax



The while Repetition Structure

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains true
- Pseudocode

```
while there are more items on my shopping list
   Purchase next item and cross it off my list
```

  - **While** loop repeated **until** the condition becomes **false**.

- Example

Loop body executes **9** times

```
int product = 2;

while ( product <= 1000 )
{
        cout << product << endl ;
         product = 2 * product;
}
```

Formulating Algorithms with Top-Down, Stepwise Refinement *(Sentinel-Controlled or Flag Repetition)*

- Suppose the problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - Unknown number of students - how will the program know to end?

- **Sentinel value**
  - Indicates “end of data entry”
  - Loop ends when sentinel inputted
  - Sentinel value chosen so it cannot be confused with a regular input (such as -1 in this case)

Example : Determine an average for unknown number of grades.

```cpp
/*
 * While-Statement.cpp
 *  Sentinel-Controlled or Flag Repetition
 *
 * Author: Husain Gholoom
 */



#include<iostream>
using namespace std;
int main ()
{
             int total,                  // sum of grades
                 gradecounter,           // number of grades entered
                 grade;                  // one grade
              double   average;          // average of grades with decimal

             //   Initialization Phase

             total = 0;                  // clear total
             gradecounter = 0;           // prepare loop counter

             // Processing Phase


             cout<< " Enter Grade  0 - 100  or  -1 to end the program:" ;
             //     Prompt to enter a grade
             cin>> grade;                //   get the grade

             while ( grade != -1  )             // Loop Counter
             {

                  total += grade;        //     add grade to total
                  gradecounter+=1;       //  Increment the Loop Counter
                  cout<< " Enter Grade  0 - 100  or  -1 to end the program:" ;
                                         //     Prompt to enter a grade
                    cin>> grade;         //   get the grade
             }


             //     Termination Phase

             if (gradecounter != 0 )  {
                    average = static_cast <double> (total) / gradecounter;
                    cout<< " Class Average is   " <<average<<endl;
             }
             else
                    cout<< " No Grades were entered   " <<endl;

       return 0;                  // Indicate the program ended successfully
}
```

Data type **double** used to represent decimal numbers

Nested control structures

- Problem:

  A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results. If more than 8 students pass, print "Raise Tuition".

- We can see that

  – The program must process 10 test results. A counter-controlled loop will be used.

  – Two counters can be used : <u>one</u> to count the number of students who passed the exam <u>and</u> <u>one</u> to count the number of students who failed the exam.

  – Each test result is a number : <u>either</u> a 1 <u>or</u> a 2. If the number is not a 1, we assume that it is a 2.

- Top level outline:

  Analyze exam results and decide if tuition should be raised

```
/*
 *  While-Statement.cpp
 *  Nested control structures
 *
 *  Author: Husain Gholoom
 */

#include<iostream>
using namespace std;
int main ()
{
             // initialize variables in declarations
      int    passes = 0,            // number of passes
             failures = 0,          // number of failures
             studentCounter = 1,    // student counter
             result;                // one exam result

             // process 10 students; counter-controlled loop
      while ( studentCounter <= 10 ) {
             cout << "Enter result (1=pass,2=fail): ";
             cin >> result;

             if ( result == 1 )         // if else nested in while
                 passes += 1;
             else
                 failures += 1;

             ++studentCounter;

      }
             // termination phase
             cout << "Passed " << passes << endl;
             cout << "Failed " << failures << endl;

             if ( passes > 8 )
                 cout << "Raise tuition " << endl;


      return 0;           // indicate the program ended successfully
}
```

Nested While Loops

Since the body of the while loop may consist of general C++ statements, one while loop may be nested inside of another. This is similar to nested if statements covered in Chapter 4.

Example

Write a program that produces the multiplication table for the number 10.

```
/*
 *  While-Statement.cpp
 *  Nested while loop - Multiplication Table for 10
 *  Created on: Oct 1, 2014
 *  Author: Husain Gholoom
 */

#include<iostream>
#include<iomanip>
using namespace std;
int main ()
{

            // initialize variables in declarations

      int i = 1, j=1;

            // process numbers 1 - 10; 1st counter-controlled loop
              while (i <= 10)
              {
            // process numbers 1 - 10; 2nd counter-controlled loop

                   j = 1;
                   while (j <=10)
                   {
            // process numbers 1 - 10; I*J AND Print

                        cout <<setw(5)<<i*j;
                        j++;
                   }
                   cout << endl;
                   i++;
              }

      return 0;   // Indicate the program ended successfully
}
```

Watch out

1) What is the output of the following ?

```
int x = 13;
while (x <= 10) {
            cout << "Repeat! " << endl;
            x = x + 1;
}

cout << "Done! " << endl;
```

If the condition is false the first time, the body is NEVER executed.

2) What is the output of the following ?

```
int x = 1;
while (x <= 10)
            cout << "Repeat! " << endl;
            cout << "Done! " << endl;
```

- Something inside the body must eventually make the condition false.
  If not, you have an infinite loop.

  - try ctrl-c to exit

3) What is the output of the following ?

```
int x = 1;
while (x <= 10)
            cout << "Repeat! " << endl;
            x = x + 1;
cout << "Done! " << endl;
```

- Don't forget the braces!!

- Another watch out:

  - don't use = for ==

Using while for Input Validation : Can check for valid characters

```
char answer ;
cout << "Enter the answer to question 1 (a,b,c or d): ";
cin >> answer;
while (answer != 'a' && answer != 'b' && answer != 'c' && answer != 'd')
 {
        cout << "Please enter a letter a, b, c or d: ";
        cin >> answer;
 }                // Do something with answer here
```

Infinite Loop

```
Infinite loop- A loop that never ends
```

```
while ( 4 > 2 )   cout<<"Hello";
```

an infinite loop ***{ Do Not Try this Example }***

Logical expression in while statement may be complex

```
while ((numGuess < 7) &&  (!guessRight))
```

Reading from or writing to a file:

Once a file has been successfully opened, you can read from it in the same way as you would read with `cin` or write to it in the same way as you write using `cout`.

Suppose the input file (in.list) consists of lines with a *char* and an *integer test score*, e.g.:

```
in.list
-------
p70
b98
a100

...


char  ch ;
int score;

...

while (!inFile.eof()) {
  inFile >> ch >> score;
  outFile << ch << " " << score+10 << endl;
}
```

In the `while` loop, we keep on reading `ch` and `score` until we hit the end of the file.

This is tested by calling the member function **`eof().`**

The bad thing about using `eof()` is that if the file is not in the right format (e.g., a letter is found when a number is expected):

```
in.list
-------
p70
b98
af100
...
```

then >> will not be able to read that line (since there is no integer to read) and it won't advance to the next line in the file.

For this error, `eof()` will not return true (it's not at the end of the file)....

Errors like that will at least mess up how the rest of the file is read. In some cases, they will cause an *infinite loop*.

One solution is to test against the number of values we expect to be read by >> operator each time. Since there are two types *a char* and *an integer*, we expect it to read in 2 values, so our condition could be:

```
while (inFile >> ch >> score) {

  outFile << ch << " " << score+10 << endl;

...

}
```

Example

```
// Program I-O Demo demonstrates how to use EOF

#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

int main()
{
   float value;
   float sum = 0;
   float average = 0;
   int count = 0;
   ifstream inData;                    // declares input stream
   ofstream outData;                   // declares output stream

   // binds program variable inData to file "Input.txt"

   inData.open("Input.txt");

   //      Testing the state of the stream
   //      true means the last I/O operation on that stream succeeded
   //      false means the last I/O operation on that stream failed

   if (!inData)
   {
     cout << "Can't open the input file successfully." << endl;
     return 1;
   }

   // binds program variable outData to file "Output.txt"

   outData.open("Output.txt");

   //      Testing the state of the stream

   if (!outData)
   {
     cout <<"Can't open the output file successfully." << endl;
     return 2;
   }
//        Read value from the input file
//        Loop terminates when EOF is encountered

   inData >> value;  // read in the first value

   while (inData)   //while previous input succeeded ...
   {
     sum = sum + value;
     count++;
     inData  >> value;      // read in the next value
   }


   // outputs sum and average

   if ( count != 0)
     average = sum / count;
     else
     average = 0;

   outData << fixed << setprecision(1) << "Total Number of Records  "
                    <<  count << "\nSum\t   "<<  sum  << "\nAverage\t"  <<  average
                    << endl;

   return 0;
}
```

Sample Run

Input.txt

```
13.1
24.2
35.3
60.4
67.5
89.6
69.7
```

Output.txt

```
Total Number of Records  7
Sum          359.8
Average            51.4
```