Software Engineering 2 (C++)

CSY2006

The Parts of a C++ Program

```
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
      cout << "Hello, there!";
      return 0;
}
```

comment

preprocessor directive

which namespace to use

beginning of function named `main`

beginning of block for `main`

output statement

string literal

send `0` to operating system

end of block for `main`

Note: For Visual Studio and Visual C++ IDE, use system("PAUSE") to prevent console from disappearing!

The C++ `string` Class

- Special data type supports working with strings
- `#include <string>`
- Can define `string` variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "George";
lastName = "Washington";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

Program 10-15

```
1   // This program demonstrates the string class.
2   #include <iostream>
3   #include <string>    // Required for the string class.
4   using namespace std;
5
6   int main()
7   {
8      string movieTitle;
9
10     movieTitle = "Wheels of Fury";
11     cout << "My favorite movie is " << movieTitle << endl;
12     return 0;
13  }
```

Program Output

```
My favorite movie is Wheels of Fury
```

Input into a `string` Object

- Use `cin >>` to read an item into a string:

```
string firstName;
cout << "Enter your first name: ";
cin >> firstName;
```

Program 10-16

```
// This program demonstrates how cin can read a string into
// a string class object.
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string name;

   cout << "What is your name? ";
   cin >> name;
   cout << "Good morning " << name << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
What is your name? Peggy [Enter]
Good morning Peggy
```

Input into a `string` Object

- Use `getline` function to put a line of input, possibly including spaces, into a string:

```
string address;
cout << "Enter your address: ";
getline(cin,address);
```

`string` Comparison

- Can use relational operators directly to compare string objects:

```
string str1 = "George",
        str2 = "Georgia";
if (str1 < str2)
  cout << str1 << " is less than "
        << str2;
```

- Comparison is performed similar to `strcmp` function.

i.e. if str1 is alphabetically greater than str2 (compares ASCII values of each character)

- Result is `true` or `false`

Program 10-18

```
// This program uses relational operators to alphabetically
// sort two strings entered by the user.
#include <iostream>
#include <string>
using namespace std;

int main ()
{
   string name1, name2;

   // Get a name.
   cout << "Enter a name (last name first): ";
   getline(cin, name1);

   // Get another name.
   cout << "Enter another name: ";
   getline(cin, name2);

   // Display them in alphabetical order.
   cout << "Here are the names sorted alphabetically:\n";
   if (name1 < name2)
      cout << name1 << endl << name2 << endl;
   else if (name1 > name2)
      cout << name2 << endl << name1 << endl;
   else
      cout << "You entered the same name twice!\n";
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter a name (last name first): Smith, Richard [Enter]
Enter another name: Jones, John [Enter]
Here are the names sorted alphabetically:
Jones, John
Smith, Richard
```

Other Definitions of C++ `strings`

Definition	Meaning
`string name;`	defines an empty string object
`string myname("Chris");`	defines a string and initializes it
`string yourname(myname);`	defines a string and initializes it
`string aname(myname, 3);`	defines a string and initializes it with first 3 characters of `myname`
`string verb(myname,3,2);`	defines a string and initializes it with 2 characters from `myname` starting at position 3
`string noname('A', 5);`	defines string and initializes it to 5 `'A'`s

`string` Operators

OPERATOR	MEANING
`>>`	extracts characters from stream up to whitespace, insert into string
`<<`	inserts string into stream
`=`	assigns string on right to string object on left
`+=`	appends string on right to end of contents on left
`+`	concatenates two strings
`[]`	references character in string using array notation
`>, >=, <, <=, ==, !=`	relational operators for string comparison. Return `true` or `false`

`string` Operators

```
string word1, phrase;
string word2 = " Dog";
cin >> word1; // user enters "Hot Tamale"
              // word1 has "Hot"
phrase = word1 + word2; // phrase has
                        // "Hot Dog"
phrase += " on a bun";
for (int i = 0; i < 16; i++)
     cout << phrase[i]; // displays
                // "Hot Dog on a bun"
```

Program 10-20

```
1  // This program demonstrates the C++ string class.
2  #include <iostream>
3  #include <string>
4  using namespace std;
5
6  int main ()
7  {
8     // Define three string objects.
9     string str1, str2, str3;
10
11    // Assign values to all three.
12    str1 = "ABC";
13    str2 = "DEF";
14    str3 = str1 + str2;
15
16    // Display all three.
17    cout << str1 << endl;
18    cout << str2 << endl;
19    cout << str3 << endl;
20
21    // Concatenate a string onto str3 and display it.
22    str3 += "GHI";
23    cout << str3 << endl;
24    return 0;
25 }
```

Program Output

```
ABC
DEF
ABCDEF
ABCDEFGHI
```

`string` Member Functions

- Are behind many overloaded operators
- Categories:
  - assignment: `assign, copy, data`
  - modification: `append, clear, erase, insert, replace, swap`
  - space management: `capacity, empty, length, resize, size`
  - substrings: `find, substr`
  - comparison: `compare`

  `See:`
  - `http://www.cplusplus.com/reference/string/string/`

`string` Member Functions

```
string word1, word2, phrase;
cin >> word1;            // word1 is "Hot"
word2.assign(" Dog");
phrase.append(word1);
phrase.append(word2);  // phrase has "Hot Dog"
phrase.append(" with mustard relish", 13);
         // phrase has "Hot Dog with mustard"
phrase.insert(8, "on a bun ");
cout << phrase << endl; // displays
         // "Hot Dog on a bun with mustard"
```

Program 10-21

```
// This program demonstrates a string
// object's length member function.
#include <iostream>
#include <string>
using namespace std;

int main ()
{
   string town;

   cout << "Where do you live? ";
   cin >> town;
   cout << "Your town's name has " << town.length() ;
   cout << " characters\n";
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Where do you live? Jacksonville [Enter]
Your town's name has 12 characters
```

Working with Characters and `string` Objects

- Remember to use #include <string>
- Using **`cin`** with the >> operator to input strings can cause problems (Pr 3-18):
- It passes over and ignores any leading *whitespace characters (spaces, tabs, or line breaks)*
- To work around this problem, you can use a C++ function named **`getline`**.

Using `getline`

Program 3-19

```
// This program demonstrates using the getline function
// to read character data into a string object.
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string name;
   string city;

   cout << "Please enter your name: ";
   getline(cin, name);
   cout << "Enter the city you live in: ";
   getline(cin, city);

   cout << "Hello, " << name << endl;
   cout << "You live in " << city << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Please enter your name: Kate Smith [Enter]
Enter the city you live in: Raleigh [Enter]
Hello, Kate Smith
You live in Raleigh
```

Working with Characters and `string` Objects

- To read a single character:
  - Use `cin`:

    ```
    char ch;
    cout << "Strike any key to continue";
    cin >> ch;
    ```

    Problem: will skip over blanks, tabs, `<CR>`
  - Use `cin.get()`:

    ```
    cin.get(ch);
    ```

    Will read the next character entered, even whitespace

Using `cin.get()`

Program 3-21

```
// This program demonstrates three ways
// to use cin.get() to pause a program.
#include <iostream>
using namespace std;

int main()
{
   char ch;

   cout << "This program has paused. Press Enter to continue.";
   cin.get(ch);
   cout << "It has paused a second time. Please press Enter again.";
   ch = cin.get();
   cout << "It has paused a third time. Please press Enter again.";
   cin.get();
   cout << "Thank you!";
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
This program has paused. Press Enter to continue. [Enter]
It has paused a second time. Please press Enter again. [Enter]
It has paused a third time. Please press Enter again. [Enter]
Thank you!
```

Working with Characters and `string` Objects

- Mixing `cin >>` and `cin.get()` in the same program can cause input errors that are hard to detect
- To skip over unneeded characters that are still in the keyboard buffer, use `cin.ignore()`:

```
cin.ignore(); // skip next char
cin.ignore(10, '\n'); // skip the next
                      // 10 char. or until a '\n'
```

`string` Member Functions and Operators

- To find the length of a string:

```
string state = "Texas";
int size = state.length();
```

- To concatenate (join) multiple strings:

```
greeting2 = greeting1 + name1;
greeting1 = greeting1 + name2;
```

Or using the **+=** combined assignment operator: greeting1 += name2;

More Mathematical Library Functions

- Require `cmath` header file
- Take `double` as input, return a `double`
- Commonly used functions:

`sin`	Sine
`cos`	Cosine
`tan`	Tangent
`sqrt`	Square root
`log`	Natural (e) log
`abs`	Absolute value (takes and returns an int)

More Mathematical Library Functions

- These require `cstdlib` header file
- `rand()`: returns a random number (`int`) between `0` and the largest int the computer holds. Yields same sequence of numbers each time program is run.
- `srand(x)`: initializes random number generator with `unsigned int x`

The if Statement-What Happens

To evaluate:

```
if (expression)
    statement;
```

- If the *expression* is `true`, then *statement* is executed.
- If the *expression* is `false`, then *statement* is skipped.

Expanding the `if` Statement

- To execute more than one statement as part of an `if` statement, enclose them in `{ }`:

```
if (score > 90)
{
    grade = 'A';
    cout << "Good Job!\n";
}
```

- `{ }` creates a <u>block</u> of code
- Remember: No semicolon after if (expression)

Relational Operators

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Fig. 2.12 | Equality and relational operators.

The `if/else` statement

- General Format:

```
if (expression)
        statement1;  // or block
else
        statement2;  // or block
```

`if-else-if` Statements

```
if (expression_1)
{
   statement;
   statement;
   etc.
}
else if (expression_2)
{
   statement;
   statement;
   etc.
}
```

If `expression_1` is true these statements are executed, and the rest of the structure is ignored.

Otherwise, if `expression_2` is true these statements are executed, and the rest of the structure is ignored.

Insert as many else if clauses as necessary

```
else
{
   statement;
   statement;
   etc.
}
```

These statements are executed if none of the expressions above are true.

Nested `if` Statements

- An `if` statement that is nested inside another `if` statement
- Nested `if` statements can be used to test more than one condition

Nested `if` Statements

```
20      // Determine the user's loan qualifications.
21      if (employed == 'Y')
22      {
23         if (recentGrad == 'Y') //Nested if
24         {
25            cout << "You qualify for the special ";
26            cout << "interest rate.\n";
27         }
28      }
```

Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

&&	AND	New relational expression is true if both expressions are true
||	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Logical Operators-Examples

```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

The logical && operator in Program

```
21    // Determine the user's loan qualifications.
22    if (employed == 'Y' && recentGrad == 'Y')
23    {
24       cout << "You qualify for the special "
25            << "interest rate.\n";
26    }
27    else
28    {
29       cout << "You must be employed and have\n"
30            << "graduated from college in the\n"
31            << "past two years to qualify.\n";
32    }
```

The logical || Operator in Program

```
23    // Determine the user's loan qualifications.
24    if (income >= MIN_INCOME || years > MIN_YEARS)
25       cout << "You qualify.\n";
26    else
27    {
28       cout << "You must earn at least $"
29            << MIN_INCOME << " or have been "
30            << "employed more than " << MIN_YEARS
31            << " years.\n";
32    }
```

The logical ！ Operator in Program

```
23    // Determine the user's loan qualifications.
24    if (!(income >= MIN_INCOME || years > MIN_YEARS))
25    {
26       cout << "You must earn at least $"
27            << MIN_INCOME << " or have been "
28            << "employed more than " << MIN_YEARS
29            << " years.\n";
30    }
31    else
32       cout << "You qualify.\n";
```

Logical Operator-Notes

- `!` has highest precedence, followed by `&&`, then `||`
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)

Comparing Characters

- Characters are compared using their ASCII values
- 'A' < 'B'
  - The ASCII value of 'A' (65) is less than the ASCII value of 'B'(66)
- '1' < '2'
  - The ASCII value of '1' (49) is less than the ASCI value of '2' (50)
- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

Relational Operators Compare Characters

```
10    // Get a character from the user.
11    cout << "Enter a digit or a letter: ";
12    ch = cin.get();
13
14    // Determine what the user entered.
15    if (ch >= '0' && ch <= '9')
16       cout << "You entered a digit.\n";
17    else if (ch >= 'A' && ch <= 'Z')
18       cout << "You entered an uppercase letter.\n";
19    else if (ch >= 'a' && ch <= 'z')
20       cout << "You entered a lowercase letter.\n";
21    else
22       cout << "That is not a digit or a letter.\n";
```

Comparing `string` Objects

- Like characters, strings are compared using their ASCII values

```
string name1 = "Mary";
string name2 = "Mark";

name1 > name2   // true
name1 <= name2 // false
name1 != name2  // true

name1 < "Mary Jane" // true
```

The characters in each string must match before they are equal

The Conditional Operator

- Can use to create short `if/else` statements
- Format: `expr ? expr : expr;`

```
x<0 ? y=10 : z=20;
```

First Expression: Expression to be tested

2nd Expression: Executes if first expression is true

3rd Expression: Executes if the first expression is false

The Conditional Operator

```
1  // This program calculates a consultant's charges at $50
2  // per hour, for a minimum of 5 hours. The ?: operator
3  // adjusts hours to 5 if less than 5 hours were worked.
4  #include <iostream>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10    const double PAY_RATE = 50.0;  // Hourly pay rate
11    const int MIN_HOURS = 5;       // Minimum billable hours
12    double hours,                  // Hours worked
13           charges;                // Total charges
14
15    // Get the hours worked.
16    cout << "How many hours were worked? ";
17    cin >> hours;
18
19    // Determine the hours to charge for.
20    hours = hours < MIN_HOURS ? MIN_HOURS : hours;
21
22    // Calculate and display the charges.
23    charges = PAY_RATE * hours;
24    cout << fixed << showpoint << setprecision(2)
25         << "The charges are $" << charges << endl;
26    return 0;
27 }
```

The `switch` Statement

- Used to select among statements from several alternatives
- In some cases, can be used instead of `if/else if` statements

`switch` Statement Format

```
switch (expression) //integer
{
  case exp1: statement1;
  case exp2: statement2;
  ...
  case expn: statementn;
  default:   statementn+1;
}
```

The `switch` Statement

Program 4-23

```
// The switch statement in this program tells the user something
// he or she already knows: the data just entered!
#include <iostream>
using namespace std;

int main()
{
   char choice;

   cout << "Enter A, B, or C: ";
   cin >> choice;
   switch (choice)
   {
      case 'A': cout << "You entered A.\n";
                break;
      case 'B': cout << "You entered B.\n";
                break;
      case 'C': cout << "You entered C.\n";
                break;
      default:  cout << "You did not enter A, B, or C!\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

`break` and `default` statements

Program 4-25

```
// This program is carefully constructed to use the "fall through"
// feature of the switch statement.
#include <iostream>
using namespace std;

int main()
{
   int modelNum;  // Model number

   // Get a model number from the user.
   cout << "Our TVs come in three models:\n";
   cout << "The 100, 200, and 300. Which do you want? ";
   cin >> modelNum;

   // Display the model's features.
   cout << "That model has the following features:\n";
   switch (modelNum)
   {
      case 300: cout << "\tPicture-in-a-picture.\n";
      case 200: cout << "\tStereo sound.\n";
      case 100: cout << "\tRemote control.\n";
                break;
      default:  cout << "You can only choose the 100,";
                cout << "200, or 300.\n";
   }
   return 0;
}
```

Continued…

`break` and `default` statements

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
   Picture-in-a-picture.
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```

The Increment and Decrement Operators

- ++ is the increment operator.

  It adds one to a variable.

  `val++;` is the same as `val = val + 1;`

- ++ can be used before (prefix) or after (postfix) a variable:

  `++val;` `val++;`

The Increment and Decrement Operators

- `--` is the decrement operator.

  It subtracts one from a variable.

  `val--;` is the same as `val = val - 1;`

- `--` can be also used before (prefix) or after (postfix) a variable:

  `--val;` `val--;`

Prefix vs. Postfix

- `++` and `--` operators can be used in complex statements and expressions
- In prefix mode (`++val, --val`) the operator increments or decrements, *then* returns the value of the variable
- In postfix mode (`val++, val--`) the operator returns the value of the variable, *then* increments or decrements

Prefix vs. Postfix - Examples

```
int num, val = 12;
cout << val++; // displays 12,
               // val is now 13;
cout << ++val; // sets val to 14,
                // then displays it
num = --val;    // sets val to 13,
               // stores 13 in num
num = val--;    // stores 13 in num,
               // sets val to 12
```

Notes on Increment and Decrement

- Can be used in expressions:

```
result = num1++ + --num2;
```

- Must be applied to something that has a location in memory. Cannot have:

```
result = (num1 + num2)++;
```

- Can be used in relational expressions:

```
if (++num > limit)
```

  pre- and post-operations will cause different comparisons

Introduction to Loops: The `while` Loop

- Loop: a control structure that causes a statement or statements to repeat
- General format of the `while` loop:

```
while (expression)
    statement;
```

- *`statement;`* can also be a block of statements enclosed in `{ }`

The `while` Loop – How It Works

```
while (expression)
        statement;
```

- *`expression`* is evaluated
  - if `true`, then *`statement`* is executed, and *`expression`* is evaluated again
  - if `false`, then the loop is finished and program statements following *`statement`* execute

The `while` loop

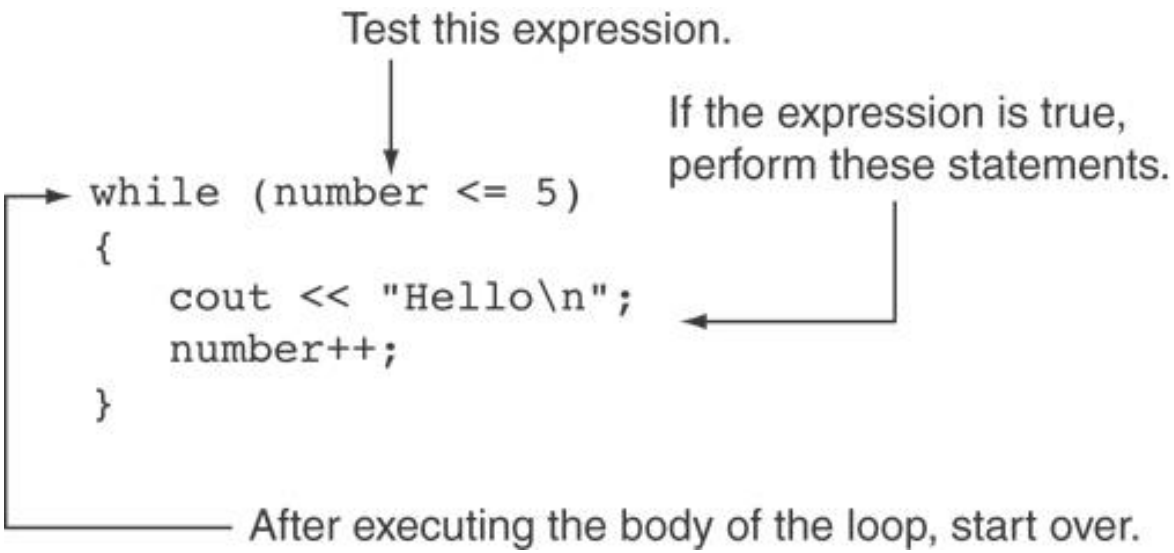
Program 5-3

```
// This program demonstrates a simple while loop.
#include <iostream>
using namespace std;

int main()
{
   int number = 1;

   while (number <= 5)
   {
      cout << "Hello\n";
      number++;
   }
   cout << "That's all!\n";
   return 0;
}
```

Program Output

```
Hello
Hello
Hello
Hello
Hello
That's all!
```

How the `while` Loop in Program Works



The `while` Loop is a Pretest Loop

`expression` is evaluated *before* the loop executes. The following loop will never execute:

```
int number = 6;
while (number <= 5)
{
    cout << "Hello\n";
    number++;
}
```

Example of an Infinite Loop

```
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```

Using the `while` Loop for Input Validation

- Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.

- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Input Validation Example

```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
   cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
   cin >> number;
}
```

A Counter Variable Controls the Loop in Program

Program 5-6

```
// This program displays a list of numbers and
// their squares.
#include <iostream>
using namespace std;

int main()
{
   const int MIN_NUMBER = 1,   // Starting number to square
             MAX_NUMBER = 10;  // Maximum number to square

   int num = MIN_NUMBER;       // Counter

   cout << "Number Number Squared\n";
   cout << "-------------------------\n";
```

Continued…

A Counter Variable Controls the Loop in Program

```
15     while (num <= MAX_NUMBER)
16     {
17        cout << num << "\t\t" << (num * num) << endl;
18        num++; //Increment the counter.
19     }
20     return 0;
21  }
```

Program Output

```
Number Number Squared
---------------------
1            1
2            4
3            9
4            16
5            25
6            36
7            49
8            64
9            81
10           100
```

The `do-while` Loop

- `do-while`: a posttest loop – execute the loop, then test the `expression`
- General Format:

```
do
   statement;  // or block in { }
while (expression);
```

- Note that a semicolon is required after `(expression)`

An Example `do-while` Loop

```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because `do-while` is a posttest loop.

A do-while Loop

Program 5-7

```
// This program averages 3 test scores. It repeats as
// many times as the user wishes.
#include <iostream>
using namespace std;

int main()
{
   int score1, score2, score3; // Three scores
   double average;             // Average score
   char again;                 // To hold Y or N input

   do
   {
      // Get three scores.
      cout << "Enter 3 scores and I will average them: ";
      cin >> score1 >> score2 >> score3;

      // Calculate and display the average.
      average = (score1 + score2 + score3) / 3.0;
      cout << "The average is " << average << ".\n";

      // Does the user want to average another set?
      cout << "Do you want to average another set? (Y/N) ";
      cin >> again;
   } while (again == 'Y' || again == 'y');
   return 0;
}
```

Continued…

A do-while Loop

Program Output with Example Input Shown in Bold

```
Enter 3 scores and I will average them: 80 90 70 [Enter]
The average is 80.
Do you want to average another set? (Y/N) y [Enter]
Enter 3 scores and I will average them: 60 75 88 [Enter]
The average is 74.3333.
Do you want to average another set? (Y/N) n [Enter]
```

The `for` Loop

- Useful for counter-controlled loop
- General Format:

```
for(initialization; test; update)
    statement; // or block in { }
```

- No semicolon after the `update` expression or after the `)`

`for` Loop - Example

```
int count;

for (count = 1; count <= 5; count++)
   cout << "Hello" << endl;
```

A for Loop

Program 5-9

```
// This program displays the numbers 1 through 10 and
// their squares.
#include <iostream>
using namespace std;

int main()
{
   const int MIN_NUMBER = 1,   // Starting value
             MAX_NUMBER = 10;  // Ending value
   int num;

   cout << "Number Number Squared\n";
   cout << "-------------------------\n";

   for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
      cout << num << "\t\t" << (num * num) << endl;

   return 0;
}
```

Continued…

A `for` Loop

Program Output

```
Number Number Squared
---------------------
1            1
2            4
3            9
4            16
5            25
6            36
7            49
8            64
9            81
10           100
```

The `for` Loop is a Pretest Loop

- The for loop tests its test expression before each iteration, so it is a pretest loop.
- The following loop will never iterate:

```
for (count = 11; count <= 10; count++)
   cout << "Hello" << endl;
```

`for` Loop - Modifications

- You can have multiple statements in the *`initialization`* expression. Separate the statements with a comma:

Initialization Expression

```
int x, y;
for (x=1, y=1; x <= 5; x++)
{
    cout << x << " plus " << y
         << " equals " << (x+y)
         << endl;
}
```

`for` Loop - Modifications

- You can also have multiple statements in the *`test`* expression. Separate the statements with a comma:

Test Expression

```
int x, y;
for (x=1, y=1; x <= 5; x++, y++)
{
    cout << x << " plus " << y
         << " equals " << (x+y)
         << endl;
}
```

`for` Loop - Modifications

- You can omit the *`initialization`* expression if it has already been done:

```
int sum = 0, num = 1;
for (; num <= 10; num++)
     sum += num;
```

`for` Loop - Modifications

- You can declare variables in the *`initialization`* expression:

```
int sum = 0;
for (int num = 0; num <= 10;
num++)
      sum += num;
```

The scope of the variable `num` is the `for` loop.

Sentinels

- <u>sentinel</u>: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, *e.g.*, `-999` for a test score
- Used to terminate input when user may not know how many values will be entered

A Sentinel in Program

Program 5-13

```
// This program calculates the total number of points a
// soccer team has earned over a series of games. The user
// enters a series of point values, then -1 when finished.
#include <iostream>
using namespace std;

int main()
{
   int game = 1,    // Game counter
       points,      // To hold a number of points
       total = 0;   // Accumulator

   cout << "Enter the number of points your team has earned\n";
   cout << "so far in the season, then enter -1 when finished.\n\n";
   cout << "Enter the points for game " << game << ": ";
   cin >> points;

   while (points != -1)
   {
      total += points;
      game++;
      cout << "Enter the points for game " << game << ": ";
      cin >> points;
   }
   cout << "\nThe total points are " << total << endl;
   return 0;
}
```

Continued…

A Sentinel in Program

Program Output with Example Input Shown in Bold

```
Enter the number of points your team has earned
so far in the season, then enter -1 when finished.

Enter the points for game 1: 7 [Enter]
Enter the points for game 2: 9 [Enter]
Enter the points for game 3: 4 [Enter]
Enter the points for game 4: 6 [Enter]
Enter the points for game 5: 8 [Enter]
Enter the points for game 6: -1 [Enter]

The total points are 34
```

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop
  - Iterates as long as a certain condition exits
  - Validating input
  - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The `for` loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

Nested Loops

- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```
for (row=1; row<=3; row++)  //outer
  for (col=1; col<=3; col++)//inner
     cout << row * col << endl;
```

Nested `for` Loop in Program

```
// Determine each student's average score.
for (int student = 1; student <= numStudents; student++)
{
   total = 0;       // Initialize the accumulator.
   for (int test = 1; test <= numTests; test++)
   {
      double score;
      cout << "Enter score " << test << " for ";
      cout << "student " << student << ": ";
      cin >> score;
      total += score;
   }
   average = total / numTests;
   cout << "The average score for student " << student;
   cout << " is " << average << ".\n\n";
}
```

Inner Loop

Outer Loop

Nested Loops - Notes

- Inner loop goes through all repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops.

Breaking Out of a Loop

- Can use `break` to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand and debug
- When used in an inner loop, terminates that loop only and goes back to outer loop

The `continue` Statement

- Can use `continue` to go to end of loop and prepare for next repetition
  - `while`, `do-while` loops: go to test, repeat loop if test passes
  - `for` loop: perform update step, then test, then repeat loop if test passes
- Use sparingly – like `break`, can make program logic hard to follow