

Welcome to “C++”

C++ is a very powerful programming language. Of course, being powerful means that there is a lot to learn in order to take advantage of that power. C++ is also portable: a program written in C++ can be compiled and run on many machines in addition to the machine it was written on. One of the ways that C++ is made portable is to put most of the functions in separate files instead of making these functions built in.

C++ is C with some extra features added. (*C++ includes objects.*) Any program that works in C will also work in C++. If you want to make sure that the programs you write are C and not C++, most compilers will force the program to compile as a C program if you save it as **name.c** instead of **name.cpp**

When you type the programs in this book, there are three important things to remember:

- C, and C++ are case sensitive. That means that **M** and **m** are not treated as the same thing. All of the lines must be typed exactly as shown, you can not reverse the case shown.
- Unless a space is part of a string enclosed in double quotes, anywhere one space is allowed, you can have any number of spaces, including a new line. Sometimes a line is indented to help the programmer see which lines belong to a particular part of the program, but the number of spaces is not important.
- The programs in this book are shown with numbers on the left. These numbers are provided for reference only. They are not part of the program. Type only the lines in the shaded box.

Hello World: The First Program

Traditionally, the first program is to print out the words “Hello World”. The program below is the C++ version of “Hello World”.

```
0 // Hello World
1 #include <iostream.h>
2 void main()
3 { cout<< "Hello World\n";
4 } // main
```

Explanation:

- 0: This line is a comment. When the compiler encounters `//` it skips everything else on that line. Although comments are ignored by the compiler, they are very important to humans. When you look at the program months from now, the comments will help you to understand the program.
- 1: C++ is made portable by putting lots of functions in header files. This line tells the compiler to include functions found in `iostream.h` when it compiles and links the program. The filename **iostream** stands for *input and output stream*. **H** stands for *header*. The commands **cout** and **cin** will not work without the code provided by this header file.
- 2: Every C++ program begins execution with the function **main**. The word **void** indicates that this function will not return an answer. The parentheses after `main` are for the arguments that the function will receive. This program will not receive any arguments, so the parentheses are empty.
- 3: All of the statements that make up a function are enclosed between `{` and `}`.

The expression `cout<<` is used to send information to the console **output** device. The output string is enclosed in double quotes. (Single quotes have a different purpose: they are not interchangeable!) The `\n` indicates a line feed. After printing the words Hello World, there is a line feed. The `\n` could also be placed in the middle of the output string:

```
cout<< "Hello\nWorld\n";
```

 prints **Hello** on one line, does a line feed and then prints **World** on the next line, then another line feed.

Note the semicolon ; at the end of the statement. Every statement ends with a semicolon.

- 4: The `}` signals the end of a block of code. Main is just one block of code that might appear in your program. It is a good habit to always label every `}` with a comment.

The program below is a slight variation on the program above. In this program an integer value of 0 is returned by main. A return value of 0 is used to indicate that the program worked, it is the last statement a function executes.

```
0 // Hello World
1 #include <iostream.h>
2 int main()
3 { cout<< "Hello World\n";
4   return 0;
5 }
```

- 2: The word `int` indicates that this function will return an integer value. Actually, `int` is the default value. If this statement had been written `main()` it would have been the same thing.
- 4: A return value of 0 indicates that *zero* is the error that occurred. In other words, it worked! (*Zero is considered false, any other number is true. There are many reasons a program could fail but only one way that it can work!*)

Check-Up

It is important that you understand the sample programs. Give yourself the following checkup:

- Look through the program line by line, make sure you know what each piece of code does.
- Enter the program and debug without looking at the sample program.
- Modify the program. For the Hello program, try to display a different message.

Variables

A program that always displays the same message can get a bit boring. Most programs get some information, do some calculations, then display the results of the calculations. Information is stored in memory as variables. Variables can *vary*, or change. Before a variable can be used, it must be declared. To declare a variable, first decide what type of information you want to store. Whole numbers, number with no decimal places are called **integers**. Numbers with decimal places are called **double**. Single letters such as ‘Y’ or ‘N’ are **characters**. Words, such as “Hello”, or a persons name are called strings. Strings are somewhat more complicated because they require an *array* of memory locations. Strings are discussed in another chapter.

After deciding on the type, the next thing a programmer must do is decide on a variable name.

- A variable name must start with a letter of the alphabet or an underscore.

- After the first letter, there can be additional letters, digits and underscores.
- There cannot be any spaces in a variable name.
- Words that have a special meaning such as `include` or `main` cannot be used to name variables.
- The name of the variable should indicate its purpose. Variable names like `a`, `b` and `c` are not very good names for variables unless you are using them for the three sides of a triangle.
- If two words are joined together (you can't have any spaces), a capital letter for the second word or an underscore can improve the clarity. Example: `hoursWorked`, `state_tax`, and `dueDate` are all good variable names.
- Remember, C++ is case sensitive. If a variable is declared as `hoursWorked`, it must be written exactly the same way each time it is used. It will be easier to remember the names if you develop a particular style and stick to it. The programs in this book will usually use a lower case letter for the first word, and capitalize the first letter of the other words. Some employers and professors may require a different style.

The statement to declare a variable lists the type first then the variable names separated by commas. Example: `int midterm, final;` declares two variables as integers. This assumes that midterm and final exam grades do not have decimal places. Numbers with decimal places are declared as double: `double hourlyRate;` A variable must be declared before it can be used. The declaration is usually the first line inside the braces for the function where the variable will be used. The variable is local to the function it is declared in. (A variable that is declared outside of any function is global.)

Assignment Statements

The easiest way to give a value to a variable is the assignment statement. The assignment statement has the following format: `variable = value;` The variable that is being assigned a value must be on the left. The equal sign is the assignment operator. The new value is on the right.

- The new value can be a constant: `x = 5;`
- An equation: `sum = a + b;`
- The new value can even use the current value of the variable: `total = total + num;` (This statement adds the current value of `total` to `num` and then assigns that number as the new value for `total`. The statement `total = total + num;` can also be written as: `total += num;`)

Inches to Centimeters

The program below declares an integer variable `inches` and a double variable `cm` (centimeters). `inches` is assigned a value of 12, then `cm` is computed by multiplying `inches` times 2.54. The asterisk `*` is used to multiply.

```
0 // Convert inches to centimeters, first draft
1 #include <iostream.h>
2 int main()
3 { int inches;
4   double cm; // centimeters
5   inches = 12;
6   cm = inches * 2.54;
7   cout<<inches<<" inches = "<<cm<<" centimeters.\n";
8   return 0;
9 }
```

Explanation:

- 0: The first line of every program should be a comment to tell what the program does
- 1: `iostream` is the header file that contains the input and output functions.
- 2: The function `main` will return an `integer`. It does not receive any arguments.
- 3: The variable `inches` is declared as an `integer`.
- 4: The variable `cm` is declared as a `double`: a number with decimal places.
- 5: The variable `inches` is assigned a value of 12.
- 6: The variable `cm` is assigned a value equal to 2.54 times the value of `inches`.
- 7: Several items are sent to the console output object, one after another. The first item is value of the variable `inches`, then the string " inches = ". Next, the value of the variable `cm` is sent. The last item is the string " centimeters.\n". The \n at the end of the last string cause a line feed. The output will appear as:
12 inches = 30.48 centimeters.
- 8: If the program gets to this point, 0 is returned to indicate that the program worked.
- 9: The curly brace `}` ends function `main`. A comment should always be used to label a `}`.

Note: A variable can also be given a value when it is declared. Statements 3 and 5 could be combined as `int inches = 12;`

The Input Object

Both `cin` (console input) and `cout`(console output) are objects. The `<<` indicates sending to the object, while `>>` indicates receiving from the object. The `cin` object looks somewhat like the `cout` object, it is also included in `iostream.h`.


The program below is exactly the same as the previous program except the user is asked to enter the number of inches instead of using 12 each time.

```
0 // Convert inches to centimeters, second draft
1 #include <iostream.h>
```

```
2 int main()
3 { int inches;
4   double cm; // centimeters
5   cout<<"Enter the number of inches:"; //prompt
6   cin>>inches; // sends from input to variable inches
7   cm = inches * 2.54;
8   cout<<inches<<" inches = "<<cm<<" centimeters.\n";
9   return 0;
10 } // main
```

The only changes are lines 5 and 6.

- 5: Prints a prompt so that the user knows what to do.
- 6: A value is sent from the input object to the variable inches.

When the program runs, the user will see the prompt. The cursor will be flashing right after the prompt. The program will wait until the user types a number and presses , then it will calculate cm and print the line of output.

Sample Output:

```
Enter the number of inches: 72
72 inches = 182.88 centimeters.
```

Special Characters

Suppose we would like to print the result as **72" = 182.88 centimeters**. Printing the double quote " and other characters such as % can be done by preceding the special character with a backslash: \. The **cout** statement would be rewritten as shown below:

```
cout<< inches << "\" = "; This will print 12" =
```

Purchase: An Example with double

```
0 // Purchase
1 #include <iostream.h>
2 int main()
3 { int quantity;
4   double cost, subtotal, tax, total;
5   cout<<"Enter cost of item: ";
6   cin>>cost; // user types in cost
7   cout<<"Enter quantity: ";
8   cin>>quantity;
9   subtotal = cost * quantity;
10  tax = subtotal * .04; //tax rate varies between states
11  total = subtotal + tax;
12  cout<<"Total cost = $"<<total<<"\n";
13  //don't worry about format for now
14  return 0;
15 }
```

Sample Output:

```
Enter cost of item: 3.95  
Enter quantity: 4  
Total cost = $ 16.43
```

Arithmetic Operators

C++ has the following operators:

Operator	Purpose:	Example:	X
+	Addition	$X = 5 + 3;$	8
-	Minus	$X = 8 - 2;$	6
	(using most recent value of x)	$X = X - 1;$	5
*	Multiplication	$X = 6 * 2;$	12
/	Division	$X = 17 / 5;$	3
		$X = 15 / 5;$	3
%	Remainder	$X = 17 \% 5;$	2

An actual program would not use a statement such as $X=5+3$; it would save time to simply use $X=8$. An actual program would be more likely to use variables: $X=Y+Z$; for example.

Note that there is a *times*, or *multiplication*, operator: $*$. In algebra, variables are always a single letter, XY in algebra means X times Y . In programming, variables can be several letters, and we could not be sure whether XY meant X times Y or a variable called XY .

Remainder: %

The $\%$ operator is used to find the remainder. Before children learn about decimal numbers, they may give the answer to division problems as:

“17 divided by 5 is 3 with a **remainder of 2**”

$$\begin{array}{r} 3 \\ 5 \overline{)17} \\ \underline{15} \\ 2 \end{array} \rightarrow 17 \% 5$$

Note that $17 / 5$ results in 3, while $17 \% 5$ results in 2.

Example: The program below converts inches to feet and inches.

```

0 // Convert inches to feet and inches
1 #include <iostream.h>
2 int main()
3 { int inches, feet, inch;
4   cout<<"Enter the number of inches: ";
5   cin>>inches;
6   feet = inches/12; // divide inches by 12
7   inch = inches%12; // remainder of inches divided by 12
8   cout<<inches<<"\" = "<<feet<<"\"<<inch<<"\"\\n";
9   return 0;
10 } // main

```

Sample Output:

```

Enter the number of inches:62
62" = 5'2"

```

Study tip: After studying the examples, try evaluating the expressions without looking at the answers; then do the exercise. You can check your answers using a short program.

Exercise: Show the value of **z** after executing each statement:

Statement:	x	y	z
<code>int z;</code>			
<code>int x = 8;</code>	8		
<code>int y = 5;</code>		5	
<code>z = x+y;</code>			
<code>z = x*2;</code>			
<code>z = x/2;</code>			
<code>z = x % 2;</code>			
<code>z = x-y;</code>			
<code>z = y % 3;</code>			

Functions

In addition to the basic operators, C++ also has functions to find square root, raise to a power and numerous others. In order to use these functions, include the math header file:

```
#include <math.h>
```

The manuals and help files will give a complete list of functions.

Order of Operations

A statement can perform more than one arithmetic operation. If there is more than one arithmetic operation in a statement, they are executed according to the level of the operators: Parenthesis take precedence over all other operators; multiplication and division take precedence over mod, mod takes precedence over addition and subtraction.

Evaluate an expression in the following order:

1. Anything inside parenthesis, including functions, are done first: working from the inner-most parenthesis out;
2. Multiplication, division and remainder (*, /, %) are performed from left to right;
3. Addition and subtraction (+ and -), are performed from left to right.

Examples: The order of evaluation is shown by underlining the next operation, then replacing it with the result (in bold) on the next line.

$$\begin{array}{r}
 \text{A. } 3 + 5 * 2 \\
 \hline
 3 + \mathbf{10} \\
 \hline
 \mathbf{13}
 \end{array}$$

** takes precedence over +*

*5*2=10: the 10 replaces 5*2 on this line*

3+10=13: the 13 replaces 3+10 on this line: Done!

$$\begin{array}{r}
 \text{B. } (3 + 5) * 2 \\
 \hline
 \mathbf{8} * 2 \\
 \hline
 \mathbf{16}
 \end{array}$$

parenthesis first

Notice the difference between the first and second problems!

C. $3 * (5 - 3 * 8 + 4)$ *parenthesis first, within parenthesis, * before + or -*
 $3 * (5 - \underline{24} + 4)$ *if same level (+ and -), work from left to right*
 $3 * (\underline{-19} + 4)$
 $3 * \underline{-15}$
 $\underline{-45}$

D. $3 * (5 - 3) * (8 + 4)$ *parenthesis first, left to right*
 $3 * \underline{2} * (8 + 4)$ *parenthesis first*
 $\underline{3 * 2} * \underline{12}$ *if same level(*, /), work from left to right*
 $\underline{6} * \underline{12}$
 $\underline{72}$

E. $-3 * (6 * (3 - 8) + 10)$ *inner-most parenthesis first*
 $-3 * (6 * \underline{-5} + 10)$ *if same level (+ and -), work from left to right*
 $-3 * (\underline{-30} + 10)$ *parenthesis first*
 $\underline{-3 * -20}$ *the minus in front of 3: -3, is a sign, not an operator*
 $\underline{60}$

Variables and Functions

If there are variables or functions in an expression, replace the variables with their current values (shown below); evaluate any functions; and then evaluate using order given above:

Examples:

x	y	z
4	5	6

A. $z - x \% y$ *replace variables with current values*
 $6 - \underline{4 \% 5}$ *% takes precedence over -, 4/5=0 with 4 remainder!*
 $\underline{6 - 4}$ *4 replaces 4%5 on this line*
 $\underline{2}$ *6-4=2, the 2 replaces 6-4 on this line: Done!*

D. $(x + y + z) / 3$ *replace variables with current values*
 $(\underline{4 + 5} + 6) / 3$ *parenthesis first: + from left to right*
 $(\underline{9} + 6) / 3$ *parenthesis first*
 $\underline{15} / 3$ *Notice that this is the average of the 3 numbers:*
 $\underline{5}$ *What is the result is the parenthesis are left out?*

Exercise: Evaluate each expression, use the values of x, y and z shown below:

x	y	z
----------	----------	----------

3	7	9
---	---	---

1. $x + y * z$
2. $(x + y) * z$
3. $z / 2 + x$
4. $z / (2 + x)$
5. $z \% 2 + x$
6. $z \% (2 + x)$

Algebra

Many times a programmer is given an algebraic formula, and must write an assignment statement that will correctly calculate the answer. After writing a statement, check it by hand, using sample values and the method shown above to verify that the equation is correct. Use the following guidelines to convert algebraic expressions to programming statements:

Algebra	Program	Explanation:
XY	<code>x*y</code>	<i>Algebra uses single letters for variables, and can omit the *, programming languages always require an operator.</i>
$X \cdot Y$	<code>x*y</code>	<i>Algebra sometimes uses a dot for multiplication.</i>
X^2	<code>x*x</code>	<i>There is no way to type a smaller exponent</i>
X^N	<code>pow(x,n)</code>	<i>Use #include <math.h> for power function</i>
\sqrt{X}	<code>sqrt(x)</code>	<i>Use #include <math.h> for square root function</i>
$\frac{A+B}{C}$	<code>(a+b)/c</code>	<i>Compound numerators and divisors must be enclosed in parenthesis: Notice the difference between this example and the one below:</i>
$A + \frac{B}{C}$	<code>a+b/c</code>	<i>Notice the difference between this example and the one above.</i>

Remember: Algebra uses single letters for variables: programmers should use good variable names wherever possible!

Exercise: Write an assignment statement using the variables in each equation:

1. $a = lw$ 2. $c = \sqrt{a^2 + b^2}$ 3. $\frac{a + 3b}{5x - y^2}$

Grades

The next program asks the user for midterm and final grades and computes the average. Notice the parenthesis around midterm and final, without the parenthesis the average would be the midterm plus half of the final. (Actually, that's not so terrible from a students point of view!)

```
0 // Calculate average from midterm and final grade
1 #include <iostream.h>
2 main()
3 { int midterm, final;
4   double avg;
5   cout<<"Enter the midterm grade: ";
6   cin>>midterm;
7   cout<<"Enter the final grade: ";
8   cin>>final;
9   avg = (midterm + final)/2.0; // try this with just 2
10  cout<<"The average is "<<avg<<"\n";
11  return 0;
12 }
```

Explanation:

- 2: `int` was left off as the return type for function `main`, but the type is still `int` because `int` is the default.
- 9: Division will return an integer if both operands are integers. When an integer is divided by a double, a double will be returned. Using `2.0` instead of `2` forces a double to be returned.

Sample Output:

```
Enter the midterm grade: 95
Enter the final grade: 90
The average is 92.5
```

Hypotenuse: Using math.h

The next program asks the user for side A and side B of a right triangle and prints the hypotenuse, side C. You may know the equation as $c^2 = a^2 + b^2$, rewritten as $c = \sqrt{a^2 + b^2}$

```
0 // Find the hypotenuse of a right triangle
1 #include <iostream.h>
2 #include <math.h>
3 main()
4 { int sideA, sideB; // sides A and B are integers
5   double sideC;      // side C++ will have decimal places
6   cout<<"Enter side A: "; // print a prompt
7   cin>>sideA;
8   cout<<"Enter side B: ";
9   cin>>sideB;
10  sideC = sqrt(sideA*sideA+sideB*sideB); //hypotenuse
11  cout<<"Side C, the hypotenuse is "<<sideC<<"\n";
12  return 0;
13 }
```

Explanation:

2: The header file `math.h` must be included in order to use the square root function.

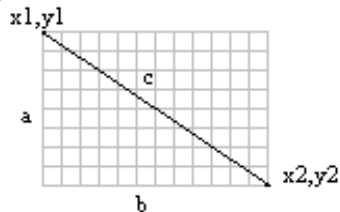
10: The `pow` function could be used as shown below, but it is more efficient to use `x*x` when raising to the power of 2. `sideC = sqrt(pow(sideA,2)+pow(sideB,2));`

Sample Output:

```
Enter side A:3
Enter side B:4
Side C, the hypotenuse is 5
```

Note: There is a function `hypot`. Line 10 could also have been written as `sideC = hypot(sideA, sideB);`

This is an important function, because you will use it to find the length of a line when you have the x and y coordinates of the two ends of the line.



Rounding

There is no function in `math.h` to round. Putting a type in parenthesis before an expression will force the expression to that type. For example `(int)89.5` returns `89`. However, if we add 0.5 to the expression, we can use it to round. The grade program is rewritten to round the average:

```
0 // Calculate rounded average from midterm and final grades
1 #include <iostream.h>
2 #include <math.h>
3 main()
4 { int midterm, final, avg;
5   double average;
6   cout<<"Enter the midterm grade: ";
7   cin>>midterm;           //enter 89
8   cout<<"Enter the final grade: ";
9   cin>>final;             //enter 90
10  average = (midterm+final)/2.0; //average is 89.5
11  avg= (int)average;       //avg=89, not an A!
12  cout<<"The average without rounding is "<<avg<<"\n";
13  //better: add 0.5 to force rounding
14  avg= (int)(average+0.5); //avg=90, YES an A!
15  cout<<"The average with rounding is "<<avg<<"\n";
16  return 0;
17 }
```

(Obviously lines 11..13 are not needed and can be omitted.)

Sample Output:

```
Enter the midterm grade: 89
Enter the final grade: 90
The average without rounding is 89
The average with rounding is 90
```

Programming Exercises:

1. Write a program to convert Fahrenheit to Celsius. $C = \frac{5}{9}(F - 32)$
2. Write a program to convert U.S. dollars to Canadian dollars.
(\$1 U.S. usually equal \$1.10 Canadian, but you can check the current exchange rate.)
3. Everything is on sale for 20% off. Ask the original price, print the sale price.
4. Ask for the number of cups. Print the number of gallons, quarts and cups.
1 gallon = 4 quarts; 1 quart = 4 cups.
5. Given an amount of money, print out the equivalent in quarters, dimes, nickels and pennies.
6. Redo the grade program. Figure the grade as follows: midterm is 30% of the grade, term paper is 25%, the final is the rest of the grade.
7. A shipping company charges \$5.00 plus 65¢ per pound for each package. Ask the weight of the package and print the shipping cost.
8. Find the average of 3 numbers. Round the answer to the nearest *hundred*.
Example: The user enters 3456, 1308 and 2765. (Total is 7529.) Print the answer as 7500.