Chapter (1)

C++ Programming Basic

C++ History

C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language.**

It was develop for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

Usage of C++

By the help of C++ programming language, we can develop different types of secured and robust applications:

- Window application
- Client-Server application
- Device drivers
- Embedded firmware etc

Programs

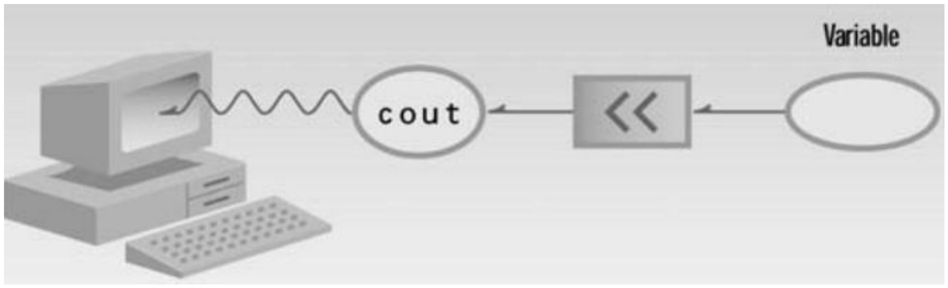
To get a computer to do something, you (or someone else) have to tell it exactly — in excruciating detail — what to do. Such a description of “what to do” is called a *program*, and *programming* is the activity of writing and testing such programs.

The classic first program

Here is a version of the classic first program. It writes “Hello, World!” to your screen:

```
// This program outputs the message “Hello, World!” to the monitor

#include <iostream>

using namespace std;

int main( )                          // C++ programs start by executing the function main

{

        cout << "Hello, World!\n";   // output “Hello, World!”

        return 0;

}
```

In C++, string literals are delimited by double quotes ("); that is, "Hello, World!\n" is a string of characters. The \n is a “special character” indicating a newline. The name cout refers to a standard output stream. Characters “put into cout” using the output operator (or) put to operator << will appear on the screen. The name cout is pronounced “see-out” and is an abbreviation of “character output stream.

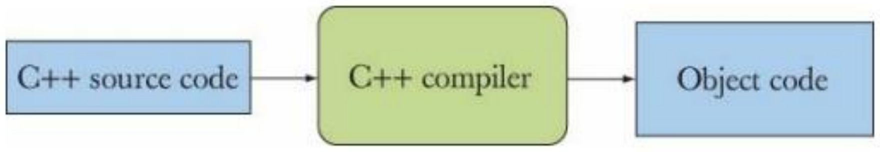


Functions

Functions are one of the fundamental building blocks of C++. The program consists almost entirely of a single function called main(). The parentheses following the word main are the distinguishing feature of a function. Without the parentheses the compiler would think that main refers to a variable or to some other program element. The body of a function is surrounded by braces (sometimes called curly brackets). a function body can consist of many statements. The main() function may also contain calls to other standalone functions. The last statement in the function body is return 0;. This tells main() to return the value 0 to whoever called it, in this case the operating system or compiler.

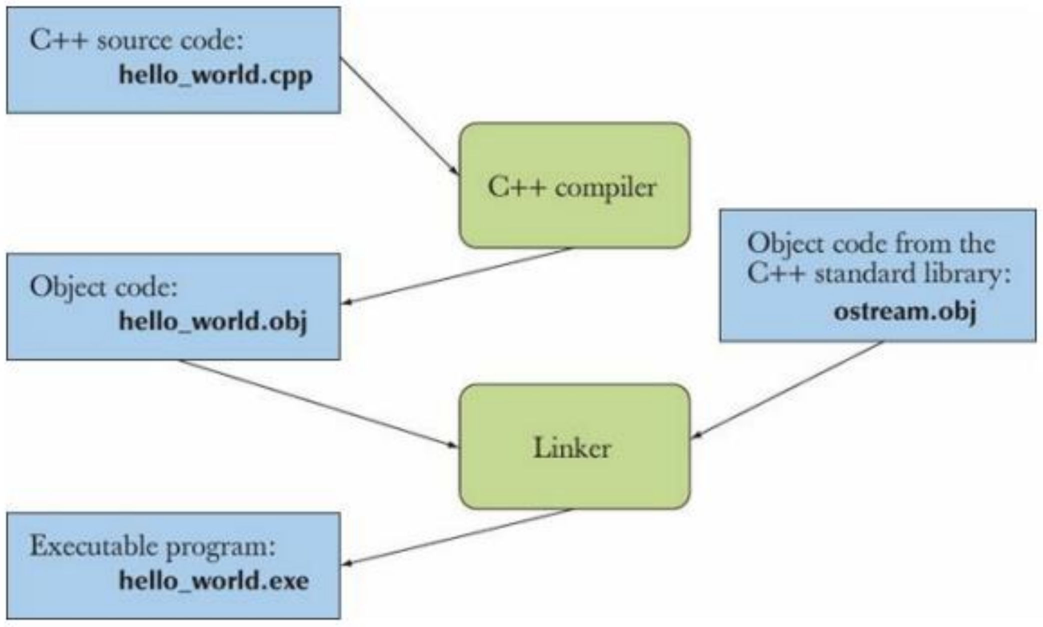
Compilation

C++ is a compiled language. That means that to get a program to run, you must first translate it from the human-readable form to something a machine can "understand." That translation is done by a program called a compiler. What you read and write is called source code or program text, and what the computer executes is called executable, object code, or machine code. Typically C++ source code files are given the suffix .cpp (e.g., hello_world.cpp) or .h (as in std_lib_facilities.h), and object code files are given the suffix .obj (on Windows) or .o (Unix).



Linking

A program usually consists of several separate parts, often developed by different people. For example, the "Hello, World!" program consists of the part we wrote plus parts of the C++ standard library. These separate parts (sometimes called translation units) must be compiled and the resulting object code files must be linked together to form an executable program. The program that links such parts together is (unsurprisingly) called a linker:



Errors

Errors found by the compiler are called compile-time errors, errors found by the linker are called link-time errors, and errors not found until the program is run are called run-time errors or logic errors. Generally, compile-time errors are easier to understand and fix than link-time errors, and link-time errors are often easier to find and fix than run-time errors and logic errors.

Directives

(1) Preprocessor directive
(2) Using directive

(1) Preprocessor Directive

The first line of the FIRST program

```
#include<iostream>
```

A preprocessor directive is an instruction to the compiler. A part of the compiler called the preprocessor deals with these directives before it begins the real compilation process. The preprocessor directive #include tells the compiler to insert another file into your source file. In effect, the #include directive is replaced by the contents of the file indicated. The type file usually included by #include is called a header file.

(2) Using Directive

A C++ program can be divided into different namespaces. A namespace is a part of the program in which certain names are recognized; outside of the namespace they're unknown. The directive using namespace std; says that all the program statements that follow are within the std namespace.

```
std :: cout << "Hello, World!\n";
```

To avoid adding std:: dozens of times in programs we use the using directive instead.

Comments

Comments are an important part of any program. They help the person writing a program, and anyone else who must read the source file, understand what's going on. The compiler ignores comments, so they do not add to the file size or execution time of the executable program.

```
// This is a single line comment.

/* this is a potentially

   very long

   multiline comment */
```

Chapter (2)

Data Types and Variables

Variables

Variables are the most fundamental part of any language. A variable has a symbolic name and can be given a variety of values. Variables are located in particular places in the computer's memory. When a variable is given a value, that value is actually placed in the memory space assigned to the variable.

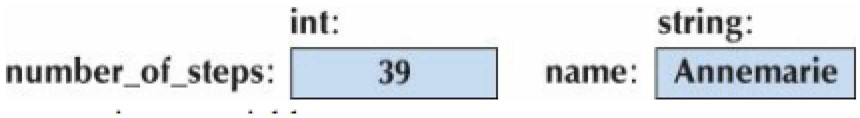
```
// intvars.cpp

// demonstrates integer variables

#include<iostream>

using namespace std;

int main()

{

        int var1;        //define var1

        int var2;        //define var2

        var1 = 20;       //assign value to var1

        var2 = var1 + 10;            //assign value to var2

        cout << "var1+10 is ";       //output text

        cout << var2 << endl;        //output value of var2

        return 0;

}
```

The statements

```
        int var1;

        int var2;
```

define two integer variables, var1 and var2. The keyword int signals the type of variable. These statements, which are called declarations, must terminate with a semicolon, like other program statements. You must declare a variable before using it. However, you can place variable declarations anywhere in a program.

```
int number_of_steps = 39; // int for integers

double flying_time = 3.5; // double for floating-point numbers

char decimal_point = '.'; // char for individual characters

string name = "Annemarie"; // string for character strings

bool tap_on = true; // bool for logical variables
```



Variable Names

The names given to variables (and other program features) are called identifiers. What are the rules for writing identifiers? You can use upper- and lowercase letters, and the digits from 1 to 9. You can also use the underscore (_). The first character must be a letter or underscore. Identifiers can be as long as you like. You can't use a C++ keyword as a variable name.

So, Var is not the same as var or VAR.

A keyword is a predefined word with a special meaning. int, class, if, and while are examples of keywords.

Basic C++ Variable Types

	Numerical Range		*Digits of*	*Bytes of*
Keyword	*Low*	*High*	*Precision*	*Memory*
`bool`	false	true	n/a	1
`char`	–128	127	n/a	1
`short`	–32,768	32,767	n/a	2
`int`	–2,147,483,648	2,147,483,647	n/a	4
`long`	–2,147,483,648	2,147,483,647	n/a	4
`float`	3.4×10^{-38}	3.4×10^{38}	7	4
`double`	1.7×10^{-308}	1.7×10^{308}	15	8

unsigned Data Types

By eliminating the sign of the character and integer types, you can change their range to start at 0 and include only positive numbers. This allows them to represent numbers twice as big as the signed type.

	Numerical Range		*Bytes of*
Keyword	*Low*	*High*	*Memory*
`unsigned char`	0	255	1
`unsigned short`	0	65,535	2
`unsigned int`	0	4,294,967,295	4
`unsigned long`	0	4,294,967,295	4

The endl Manipulator

Manipulators are instructions to the output stream that modify the output in various ways. This causes a linefeed to be inserted into the stream, so that subsequent text is displayed on the next line. It has the same effect as sending the '\n' character.

Character Constants

Character constants use single quotation marks around a character, like 'a' and 'b'. (Note that this differs from string constants, which use double quotation marks.) When the C++ compiler encounters such a character constant, it translates it into the corresponding ASCII code. The constant 'a' appearing in a program, for example, will be translated into 97, as shown in Figure.

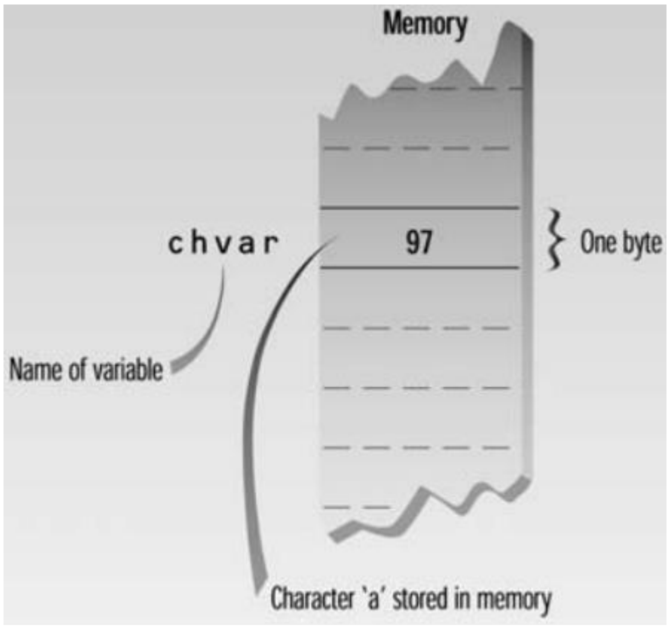


Figure: Variable of type char in memory

```
// charvars.cpp
// demonstrates character variables
#include<iostream>     //for cout, etc.
using namespace std;
int main()
{
            char charvar1 = ‘A’;      //define char variable as character
            char charvar2 = ‘\t’;     //define char variable as tab
            cout << charvar1;         //display character
            cout << charvar2;         //display character
            charvar1 = ‘B’;           //set char variable to char constant
            cout << charvar1;         //display character
            cout << ‘\n’;             //display newline character
            return 0;
}
```

Escape Sequence

Escape sequences can be used as separate characters or embedded in string constants. Table shows a list of common escape sequences.

Table: Common Escape Sequence

Escape Sequence	*Character*
\ a	Bell (beep)
\ b	Backspace
\ f	Formfeed
\ n	Newline
\ r	Return
\ t	Tab
\ \	Backslash
\ '	Single quotation mark
\ "	Double quotation marks
\ xdd	Hexadecimal notation

```
cout << “\”Run, Spot, run,\” she said.”;
```

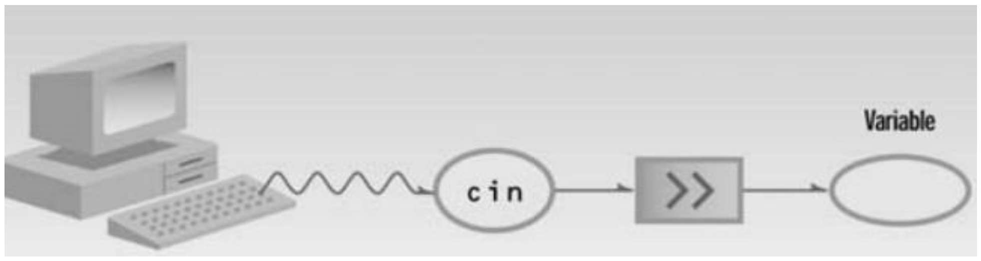
This translates to “Run, Spot, run,” she said.

Input with cin

```
// read name and age

int main()

{

        cout << "Please enter your first name and age\n";

        string first_name;        // string variable

        int age;                  // integer variable

        cin >> first_name;        // read a string

        cin >> age;               // read an integer

        cout << "Hello, " << first_name << " (age " << age << ")\n";

}

// read name and age (2nd version)

int main()

{

        cout << "Please enter your first name and age\n";

        string first_name = "???"; // string variable // ("???” means “don’t know the name”)
        int age = –1;              // integer variable (–1 means “don’t know the age”)

        cin >> first_name >> age;       // read a string followed by an integer

        cout << "Hello, " << first_name << " (age " << age << ")\n";

}

// fahren.cpp

// demonstrates cin, newline

#include <iostream>

using namespace std;

int main()

{
        int ftemp;          //for temperature in fahrenheit

        cout << “Enter temperature in fahrenheit: “;

        cin >> ftemp;

        int ctemp = (ftemp-32) * 5 / 9;

        cout << “Equivalent in Celsius is: “ << ctemp << ‘\n’;

        return 0;

}
```

Here’s some sample interaction with the program:

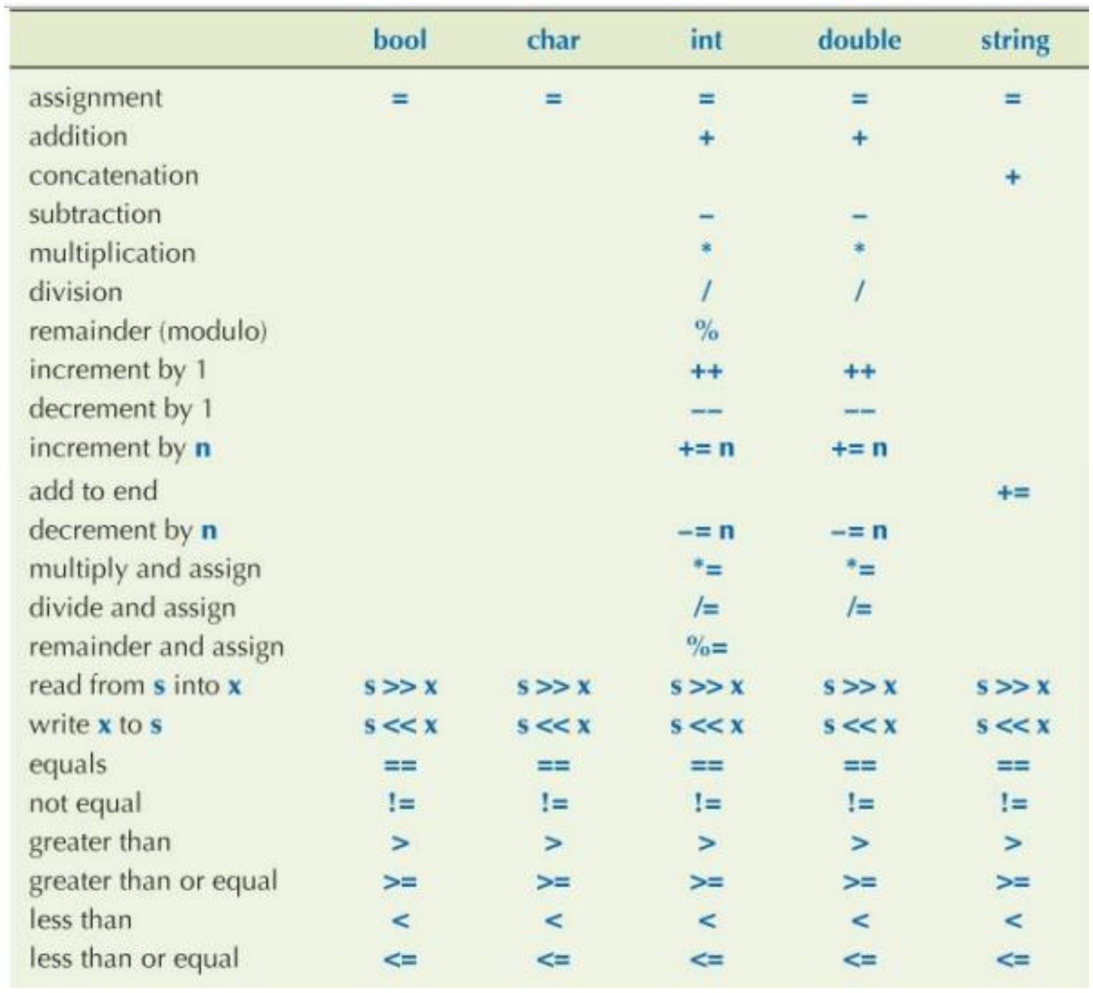
```
Enter temperature in fahrenheit: 212

Equivalent in Celsius is: 100
```



Operations and Operators

In addition to specifying what values can be stored in a variable, the type of a variable determines what operations we can apply to it and what they mean. For example:

```
    int count;

    cin >> count;           // >> reads an integer into count

    string name;

    cin >> name;            // >> reads a string into name

    int c2 = count+2;       // + adds integers

    string s2 = name + " Jr. ";     // + appends characters

    int c3 = count–2;               // – subtracts integers

    string s3 = name – " Jr. ";     // error: – isn’t defined for strings
```



	bool	char	int	double	string
assignment	=	=	=	=	=
addition			+	+	
concatenation					+
subtraction			–	–	
multiplication			*	*	
division			/	/	
remainder (modulo)			%		
increment by 1			++	++	
decrement by 1			––	––	
increment by **n**			+= n	+= n	
add to end					+=
decrement by **n**			–= n	–= n	
multiply and assign			*=	*=	
divide and assign			/=	/=	
remainder and assign			%=		
read from **s** into **x**	s >> x	s >> x	s >> x	s >> x	s >> x
write **x** to **s**	s << x	s << x	s << x	s << x	s << x
equals	==	==	==	==	==
not equal	!=	!=	!=	!=	!=
greater than	>	>	>	>	>
greater than or equal	>=	>=	>=	>=	>=
less than	<	<	<	<	<
less than or equal	<=	<=	<=	<=	<=

Remainder Operator (or) Modulus Operator

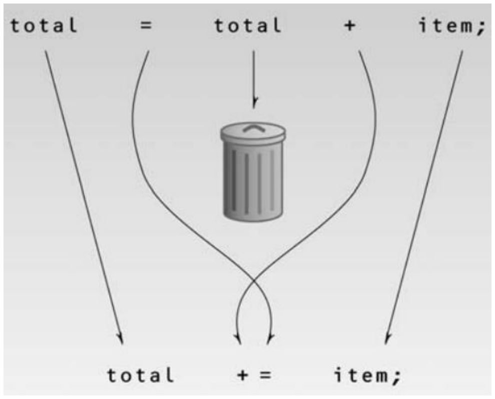
```
// remaind.cpp
// demonstrates remainder operator
#include <iostream>
using namespace std;
int main()
{
        cout << 6 % 8 << endl       // 6
             << 7 % 8 << endl       // 7
             << 8 % 8 << endl       // 0
             << 9 % 8 << endl       // 1
             << 10 % 8 << endl;     // 2
        return 0;
    }
```

Arithmetic Assignment Operator

The following kind of statement is common in most languages.

```
total = total + item;    // adds "item" to "total"
```

The arithmetic assignment operator, which combines an arithmetic operator and an assignment operator and eliminates the repeated operand. total += item; // adds "item" to "total"



There are arithmetic assignment operators corresponding to all the arithmetic operations:

+=, -=, *=, /=, and %=

```
// assign.cpp
// demonstrates arithmetic assignment operators
#include
using namespace std;
int main()
{
    int ans = 27;
    ans += 10;      //same as: ans = ans + 10;
    cout << ans << ", ";
    ans -= 7;       //same as: ans = ans - 7;
    cout << ans << ", ";
    ans *= 2;       //same as: ans = ans * 2;
    cout << ans << ", ";
```

```
        ans /= 3;          //same as: ans = ans / 3;
        cout << ans << ", ";
        ans %= 3;          //same as: ans = ans % 3;
        cout << ans << endl;
        return 0;
}
```

Here's the output from this program: 37, 30, 60, 20, 2

Increment Operators

count = count + 1;   // adds 1 to "count"

Or you can use an arithmetic assignment operator:

count += 1;   // adds 1 to "count"

But there's an even more condensed approach:

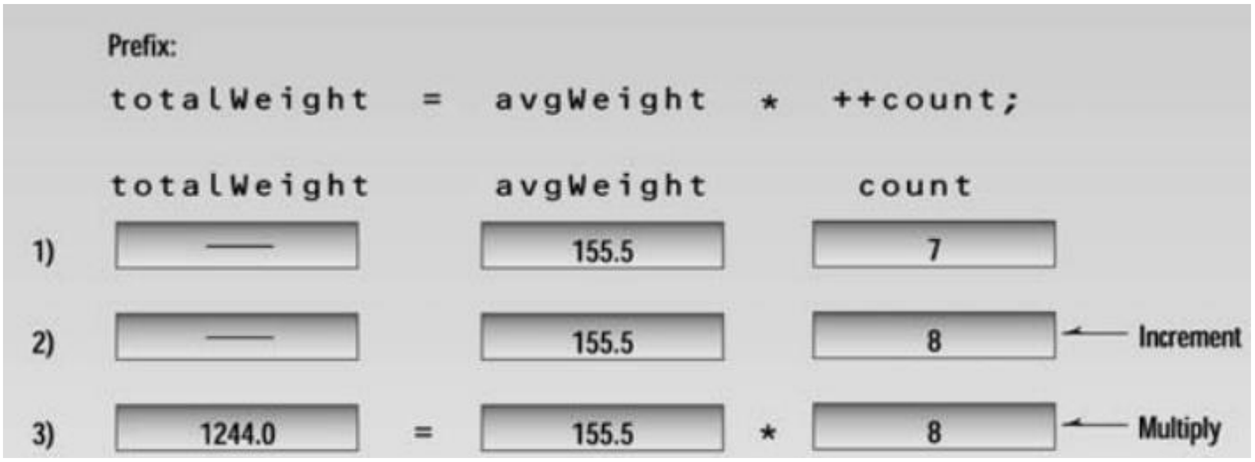
++count;   // adds 1 to "count"

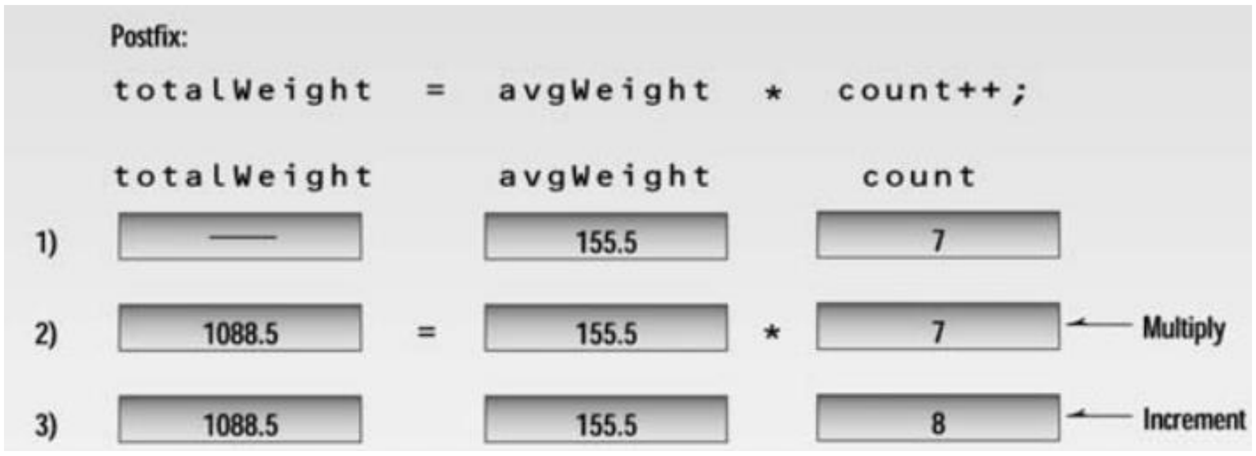
The ++ operator increments (adds 1 to) its argument.

Prefix and Postfix

The increment operator can be used in two ways: as a prefix, meaning that the operator precedes the variable; and as a postfix, meaning that the operator follows the variable. What's the difference?

For example: totalWeight = avgWeight * ++count;





```
// increm.cpp
// demonstrates the increment operator
#include <iostream>
using namespace std;
int main()
{
        int count = 10;
        cout << "count=" << count << endl;          //displays 10
        cout << "count=" << ++count << endl;        //displays 11 (prefix)
        cout << "count=" << count << endl;          //displays 11
        cout << "count=" << count++ << endl;        //displays 11 (postfix)
        cout << "count=" << count << endl;          //displays 12
        return 0;
}
```

Here’s the program’s output:

```
        count=10
        count=11
        count=11
        count=11
        count=12
```

The Decrement (--) Operator

The decrement operator, --, behaves very much like the increment operator, except that it subtracts 1 from its operand.

Library Functions

Many activities in C++ are carried out by library functions. These functions perform file access, mathematical computations, and data conversion, among other things. The next example, SQRT, uses the library function sqrt() to calculate the square root of a number entered by the user.

```
// sqrt.cpp

// demonstrates sqrt() library function

#include <iostream>              //for cout, etc.

#include <cmath>                 //for sqrt()

using namespace std;

int main()

{

        double number, answer;        //sqrt() requires type double

        cout << “Enter a number: “;

        cin >> number;                //get the number

        answer = sqrt(number);        //find square root

        cout << “Square root is “ << answer << endl;        //display it

        return 0;

}
```

Here’s some output from the program:

```
Enter a number: 1000

Square root is 31.622777
```

```
// simple program to exercise operators

int main()

{

        cout << "Please enter a floating-point value: ";
```

```
    double n;
    cin >> n;
    cout << "n == " << n
        << "\nn+1 == " << n+1
        << "\nthree times n == " << 3*n
        << "\ntwice n == " << n+n
        << "\nn squared == " << n*n
        << "\nhalf of n == " << n/2
        << "\nsquare root of n == " << sqrt(n) << '\n';
}
```

The library functions and objects will be linked to your program to create an executable file. These files contain the actual machine-executable code for the functions. Such library files often have the extension .LIB. The sqrt() function is found in such a file. It is automatically extracted from the file by the linker. The IOSTREAM header file contains information for various I/O functions and objects, including cout, while the CMATH header file contains information for mathematics functions such as sqrt(). If you were using string functions such as strcpy(), you would include STRING.H, and so on.

The const Qualifier

```
const float PI = 3.14159F;    //type const float
```

The keyword const (for constant) precedes the data type of a variable. It specifies that the value of a variable will not change throughout the program. Any attempt to alter the value of a variable defined with this qualifier will elicit an error message from the compiler.

```
// circarea.cpp
// demonstrates floating point variables
#include <iostream>          //for cout, etc.
using namespace std;
int main()
{
    float rad;                    //variable of type float
    const float PI = 3.14159F;    //type const float
    cout << “Enter radius of circle: “;    //prompt
    cin >> rad;                          //get
    float area = PI * rad * rad;         //find area c
    cout << "Area is " << area << endl;  //display answer
    return 0;
}
```

Here's a sample interaction with the program:

```
Enter radius of circle: 0.5
Area is 0.785398
```

The #define Directive

Constants can also be specified using the preprocessor directive #define.

For example,   #define PI 3.14159

The setw Manipulator

setw, which changes the field width of output. That is, the integer 567 will occupy a field three characters wide, and the string "pajamas" will occupy a field seven characters wide.

```
// width1.cpp
// demonstrates need for setw manipulator
#include <iostream>
using namespace std;
int main()
{
    long pop1=2425785, pop2=47, pop3=9761;
    cout << "LOCATION " << "POP." << endl
        << "Portcity " << pop1 << endl
        << "Hightown " << pop2 << endl
        << "Lowville " << pop3 << endl;
    return 0;
}
```

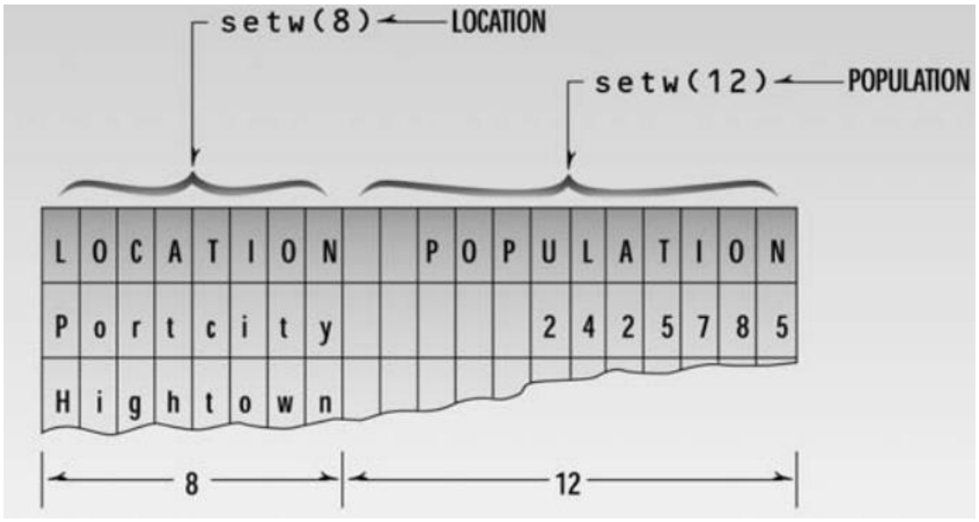
Here’s the output from this program:

```
LOCATION POP.
Portcity 2425785
Hightown 47
Lowville 9761
```

```
// width2.cpp
// demonstrates setw manipulator
#include <iostream>
#incluce <iomanip>        // for setw
using namespace std;
int main()
{
        long pop1=2425785, pop2=47, pop3=9761;
        cout << setw(8) << “LOCATION” << setw(12) << “POPULATION” << endl
                << setw(8) << “Portcity” << setw(12) << pop1 << endl
                << setw(8) << “Hightown” << setw(12) << pop2 << endl
                << setw(8) << “Lowville” << setw(12) << pop3 << endl;
        return 0;
}
```

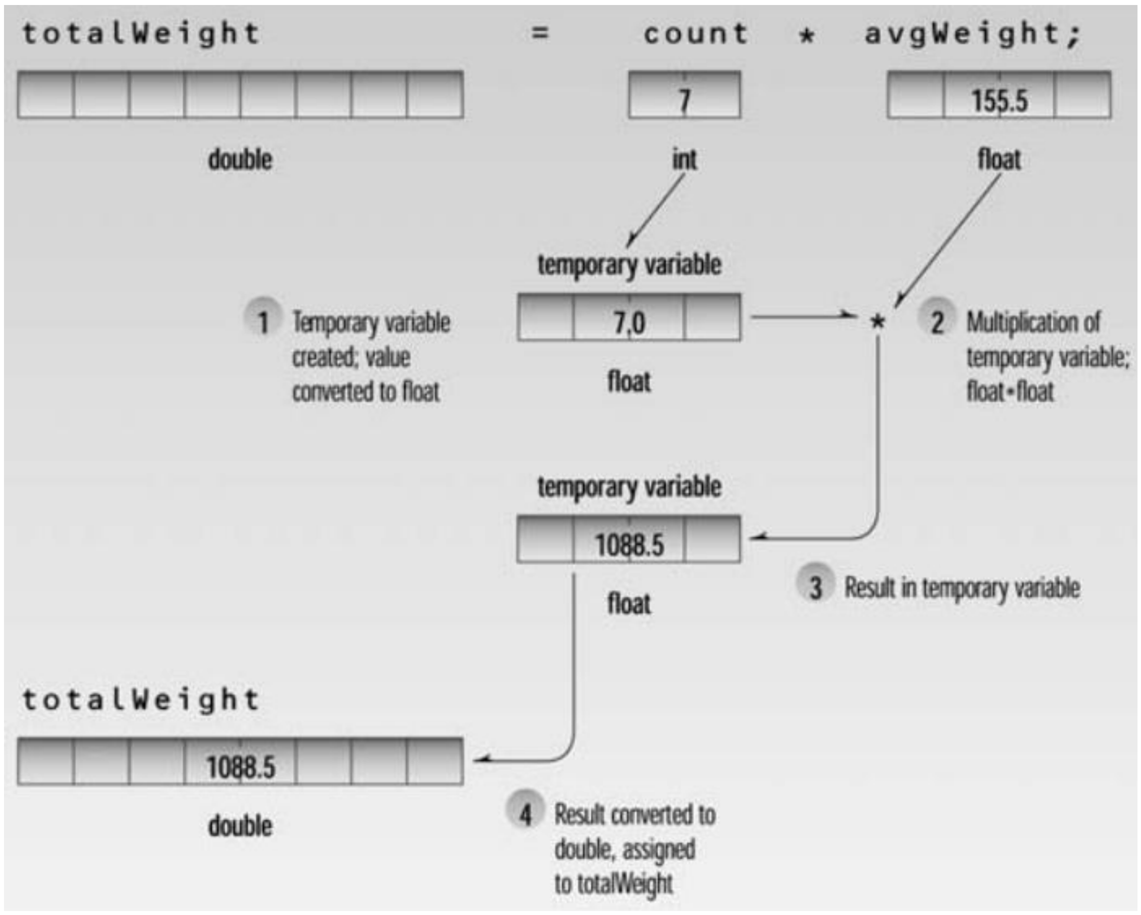
Here’s the output of WIDTH2:

```
LOCATION POPULATION
Portcity     2425785
Hightown          47
Lowville        9761
```



Type Conversion

```
// mixed.cpp
// shows mixed expressions
#include <iostream>
using namespace std;
int main()
{
	int count = 7;
	float avgWeight = 155.5F;
	double totalWeight = count * avgWeight;
	cout << “totalWeight=” << totalWeight << endl;
	return 0;
}
```


totalWeight = count * avgWeight;
7
155.5
double
int
float
temporary variable
7.0
float
1 Temporary variable created; value converted to float
2 Multiplication of temporary variable; float*float
temporary variable
1088.5
float
3 Result in temporary variable
totalWeight
1088.5
double
4 Result converted to double, assigned to totalWeight

Automatic Conversion

When two operands of different types are encountered in the same expression, the lower-type variable is converted to the type of the higher-type variable.

5/2 is 2 (not 2.5)

2.5/2 means 2.5/double(2), that is, 1.25

'a'+1 means int{'a'}+1

Table: Order of Data Types

Data Type	*Order*
`long double`	Highest
`double`	
`float`	
`long`	
`int`	
`short`	
`char`	Lowest

Casts

Casts are also called type casts. What are casts for? Sometimes a programmer needs to convert a value from one type to another in a situation where the compiler will not do it automatically or without complaining.

Eg: aCharVar=static_cast<char>(anIntVar)

d=static_cast<double>(intVar)

Questions

1. A function name must be followed by ________.

2. A function body is delimited by ________.

3. Why is the main() function special?

4. A C++ instruction that tells the computer to do something is called a ________.

5. Write an example of a normal C++ comment.

6. An expression

    a. usually evaluates to a numerical value.

    b. indicates the emotional state of the program.

    c. always occurs outside a function.

    d. may be part of a statement.

7. Specify how many bytes are occupied by the following data types in a 32-bit system:

    a. Type int

    b. Type long double

    c. Type float

    d. Type long

8. True or false: A variable of type char can hold the value 301.

9. What kind of program elements are the following?

    a. 12

    b. ‘a’

    c. 4.28915

    d. JungleJim

    e. JungleJim()

10. Write statements that display on the screen

a. the character ‘x’

b. the name Jim

c. the number 509

11. True or false: In an assignment statement, the value on the left of the equal sign is always equal to the value on the right.

12. Write a statement that displays the variable george in a field 10 characters wide.

13. What header file must you #include with your source file to use cout and cin?

14. Write a statement that gets a numerical value from the keyboard and places it in the variable temp.

15. What header file must you #include with your program to use setw?

16. Two exceptions to the rule that the compiler ignores whitespace are ________ and ________.

17. True or false: It’s perfectly all right to use variables of different data types in the same arithmetic expression.

18. The expression 11%3 evaluates to ________.

19. An arithmetic assignment operator combines the effect of what two operators?

20. Write a statement that uses an arithmetic assignment operator to increase the value of the variable temp by 23. Write the same statement without the arithmetic assignment operator.

21. The increment operator increases the value of a variable by how much?

22. Assuming var1 starts with the value 20, what will the following code fragment print out?

```
cout << var1--;
cout << ++var1;
```

23. In the examples we’ve seen so far, header files have been used for what purpose?

24. The actual code for library functions is contained in a ________ file.

Exercises

1. Assuming there are 7.481 gallons in a cubic foot, write a program that asks the user to enter a number of gallons, and then displays the equivalent in cubic feet.

2. Write a program that generates the following table:

```
1990      135
1991     7290
1992    11300
1993    16200
```

Use a single cout statement for all output.

3. Write a program that generates the following output:

```
10
20
19
```

Use an integer constant for the 10, an arithmetic assignment operator to generate the 20, and a decrement operator to generate the 19.

4. Write a program that displays your favorite poem. Use an appropriate escape sequence for the line breaks. If you don’t have a favorite poem, you can borrow this one by Ogden Nash:

Candy is dandy,

But liquor is quicker.

5. A library function, islower(), takes a single character (a letter) as an argument and returns a nonzero integer if the letter is lowercase, or zero if it is uppercase. This function requires the header file CTYPE.H.

Write a program that allows the user to enter a letter, and then displays either zero or nonzero, depending on whether a lowercase or uppercase letter was entered. (See the SQRT program for clues.)

6. On a certain day the British pound was equivalent to $1.487 U.S., the French franc was $0.172, the German deutschemark was $0.584, and the Japanese yen was $0.00955.

Write a program that allows the user to enter an amount in dollars, and then displays this value converted to these four other monetary units.

7. You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by 9/5 and adding 32. Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit.

8. When a value is smaller than a field specified with setw(), the unused locations are, by default, filled in with spaces. The manipulator setfill() takes a single character as an argument and causes this character to be substituted for spaces in the empty parts of a field. Rewrite the WIDTH program so that the characters on each line between the location name and the population number are filled in with periods instead of spaces, as in

```
Portcity.....2425785
```

9. If you have two fractions, a/b and c/d, their sum can be obtained from the formula

$$\frac{a}{b} + \frac{c}{d} = \frac{a \times d + b \times c}{b \times d}$$

For example, 1/4 plus 2/3 is

$$\frac{1}{4} + \frac{2}{3} = \frac{1 \times 3 + 2 \times 4}{4 \times 3} = \frac{3+8}{12} = \frac{11}{12}$$

Write a program that encourages the user to enter two fractions, and then displays their sum in fractional form. (You don’t need to reduce it to lowest terms.) The interaction with the user might look like this:

```
Enter first fraction: 1/2

Enter second fraction: 2/5

Sum = 9/10
```

You can take advantage of the fact that the extraction operator (>>) can be chained to read in more than one quantity at once:

```
cin >> a >> dummychar >> b;
```

Chapter(3)

Selections and Loops

Relational Operators

A relational operator compares two values. The values can be any built-in C++ data type, such as char, int, and float, or they can be user-defined classes. The comparison involves such relationships as equal to, less than, and greater than. The result of the comparison is true or false.

```
// relat.cpp

// demonstrates relational operators

#include<iostream>

using namespace std;

int main()

{

        int numb;

        cout << “Enter a number: “;

        cin >> numb;

        cout << “numb<10 is ”<< (numb < 10) << endl;

        cout << “numb>10 is “ << (numb > 10) << endl;

        cout << “numb==10 is “ << (numb == 10) << endl;

        return 0;

}
```

Here’s the output when the user enters 20:

```
Enter a number: 20

numb<10 is 0

numb>10 is 1

numb==10 is 0
```

C++ Relational Operators

Operator	*Meaning*
>	Greater than (greater than)
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

```
jane = 44; //assignment statement
harry = 12; //assignment statement
(jane == harry) //false
(harry <= 12) //true
(jane > harry) //true
(jane >= 44) //true
(harry != 12) // false
(7 < harry) //true
(0) //false (by definition)
(44) //true (since it's not 0)
```

Selections

In programs, as in life, we often have to select among alternatives. In C++, that is done using either an if-statement or a switch-statement.

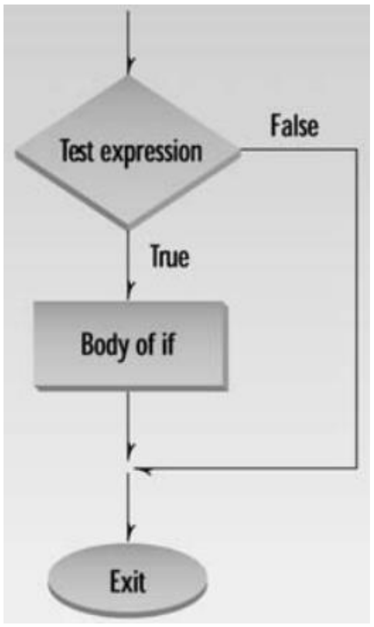
The if Statement

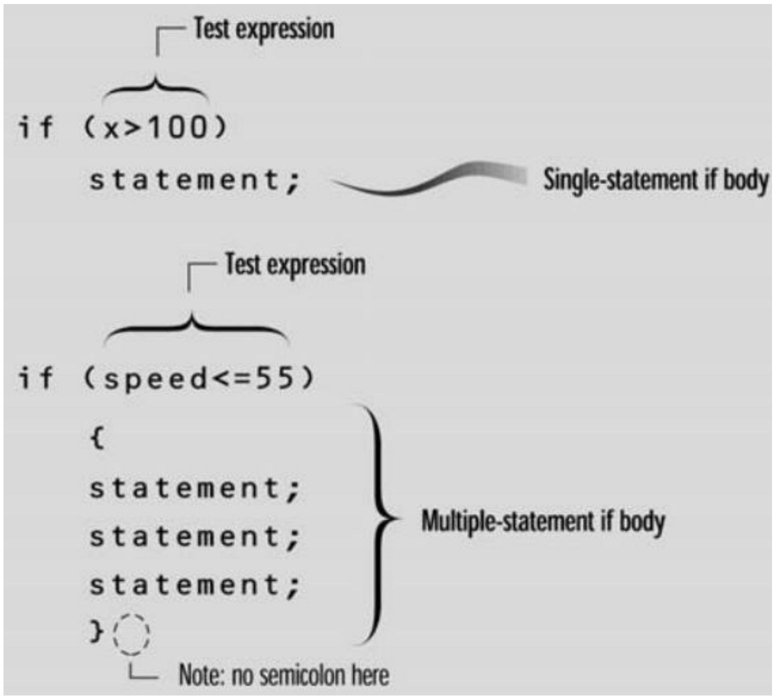
The if statement is the simplest of the decision statements.

```
// ifdemo.cpp
// demonstrates IF statement
#include<iostream>
using namespace std;
int main()
{
        int x;
        cout << “Enter a number: “;
        cin >> x;
        if( x > 100 )
                cout << “That number is greater than 100\n”;
        return 0;
}
```

Here’s an example of the IFDEMO program’s output when the number entered by the user is greater than 100:

```
Enter a number: 2000
That number is greater than 100
```





```
if (x>100)
    statement;

if (speed<=55)
    {
    statement;
    statement;
    statement;
    }
```

Figure: Operation of the if statement Figure: Syntax of the if statement

```
// if2.cpp
// demonstrates IF with multiline body
#include<iostream>
using namespace std;
int main()
{
        int x;
        cout << “Enter a number: “;
        cin >> x;
        if( x > 100 )
        {
            cout << “The number “ << x;
            cout << “ is greater than 100\n”;
        }
        return 0;
}
```

Here’s some output from IF2:

```
Enter a number: 12345
The number 12345 is greater than 100
```

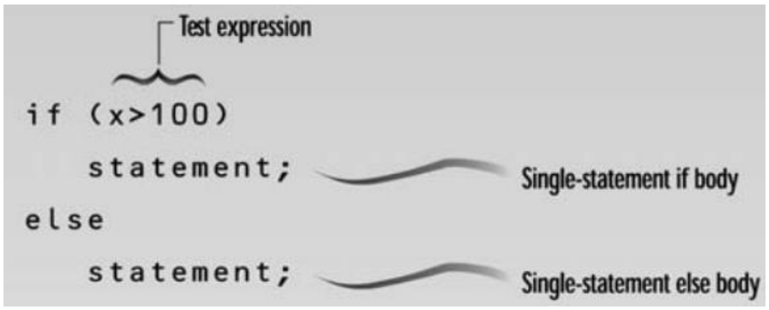
The if .... else Statement

```
// ifelse.cpp
// demonstrates IF...ELSE statememt
#include<iostream>
using namespace std;
int main()
{
        int x;
        cout << “\nEnter a number: “;
        cin >> x;
        if( x > 100 )
            cout << “That number is greater than 100\n”;
        else
            cout << “That number is not greater than 100\n”;
        return 0;
```

}

Here’s output from two different invocations of the program:

```
Enter a number: 300
That number is greater than 100
Enter a number: 3
That number is not greater than 100
```



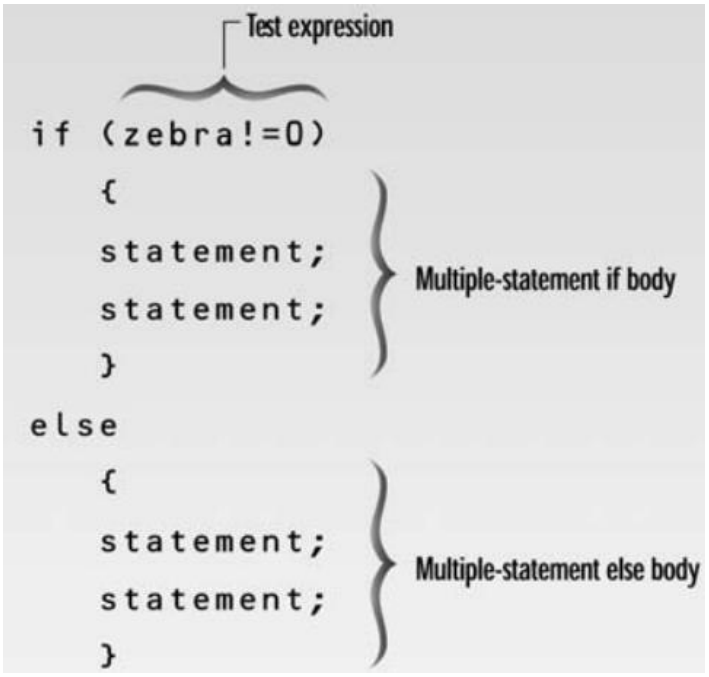


Figure: Syntax of the if ... else statement

```
#include<iostrem>
using namespace std;
int main()
{
        int a = 0;
        int b = 0;
        cout << "Please enter two integers\n";
        cin >> a >> b;
        if (a>b)
                cout<< "max(" << a << "," << b <<") is " << a<<"\n";
        else
                cout << "max(" << a << "," << b <<") is " << b << "\n";
}
```

The else if Construction

```
#include<iostream>
using namespace std;
int main( )
{
        int num;
        cout<<”Enter a number:”;
        cin>>num;
        if(num>0)
                cout<<”The number is positive”;
        else if (num==0)
                cout<<”The number is zero”;
        else
                cout<<”The number is negative”;
        return 0;
}
```

Nested if Statement

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

   int a = 20, b = 10, c = 2;
    if (a > b) {
        if (a > c) {
           cout << " a is the largest " << endl;
       }
   }
   return 0;
}
```

```
#include<iostream>

using namespace std;

int main( )

{

        int a=100, b=200;

        if(a = = 100){

                if(b = = 200){

                        cout<<"Value of a is 100 and b is 200"<<endl;

                                }

        }

        cout<< "Exact value of a is "<<a<<endl;

        cout<<"Exact value of b is"<<b<<endl;

        return 0;

}
```

Output:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

The Switch Statement

Here are some technical details about switch-statements:

1. The value on which we switch must be of an integer, char, or enumeration type. In particular, you cannot switch on a string.

2. The values in the case labels must be constant expressions. In particular, you cannot use a variable in a case label.

3. You cannot use the same value for two case labels.

4. You can use several case labels for a single case.

5. Don't forget to end each case with a break. Unfortunately, the compiler probably won't warn you if you forget.

For example:

```
int main() // you can switch only on integers, etc.
{
        cout << "Do you like fish?\n";
        string s;
        cin >> s;
        switch (s) {    // error: the value must be of integer, char, or enum type
                case "no":
                        // . . .
                        break;
                case "yes":
                        // . . .
                        break;
                  }
}
```

```
// platters.cpp
// demonstrates SWITCH statement
#include<iostream>
using namespace std;
```

```
int main()
{
	int speed;
	cout << “\nEnter 33, 45, or 78: “;
	cin >> speed;
	switch(speed)
	{
		case 33:
			cout << “LP album\n”;
			break;
		case 45:
			cout << “Single selection\n”;
			break;
		case 78:
			cout << “Obsolete format\n”;
			break;
	}
	return 0;
}
```

Here’s an example of PLATTER’s output:

```
Enter 33, 45, or 78: 45
Single selection
```

switch versus if ... else

```
if( SteamPressure*Factor > 56 )
	// statements
else if( VoltageIn + VoltageOut < 23000)
	// statements
else if( day==Thursday )
```

```
        // statements
else
        // statements
```

You can't say

```
        case a<3:
                //do something
                break;
```

The case constant must be an integer or character constant, like 3 or 'a', or an expression that evaluates to a constant, like 'a'+32.

The Conditional Operator

For example, here's an if...else statement that gives the variable min the value of alpha or the value of beta, depending on which is smaller:

```
        if( alpha < beta )
                min = alpha;
        else
                min = beta;
```

This operator consists of two symbols, which operate on three operands. It's the only such operator in C++; other operators operate on one or two operands. Here's the equivalent of the same program fragment, using a conditional operator:

```
        min = (alpha<beta)? alpha : beta
```

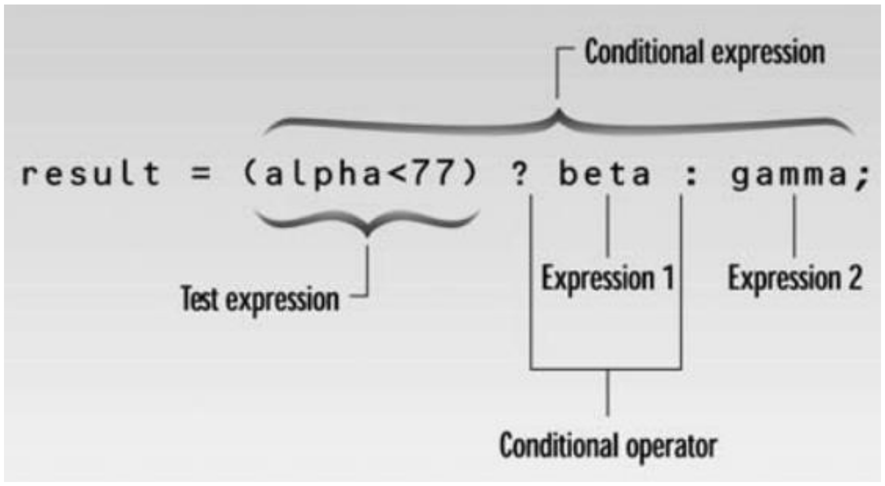


Figure: Syntax of the conditional operator

Loops

Loops cause a section of your program to be repeated a certain number of times. The repetition continues while a condition is true. When the condition becomes false, the loop ends and control passes to the statements following the loop.

There are three kinds of loops in C++: the for loop, the while loop, and the do loop.

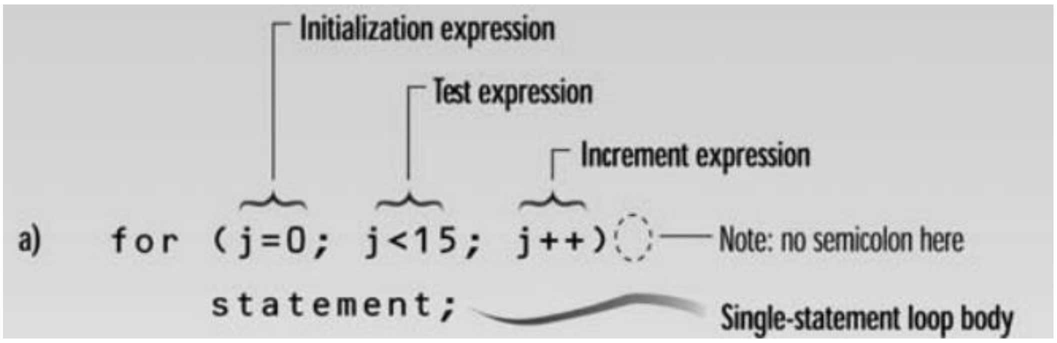
The for Loop

Here’s an example, FORDEMO, that displays the squares of the numbers from 0 to 14:

```
// fordemo.cpp

// demonstrates simple FOR loop

#include<iostream>

using namespace std;

int main()

{         int j;    //define a loop variable

          for(j=0; j<15; j++)

                    cout<< j * j << “ “;     //displaying the square of j

          cout << endl;

          return 0;

}
```

Here’s the output:

```
0 1 4 9 16 25 36 49 64 81 100 121 144 169 196
```



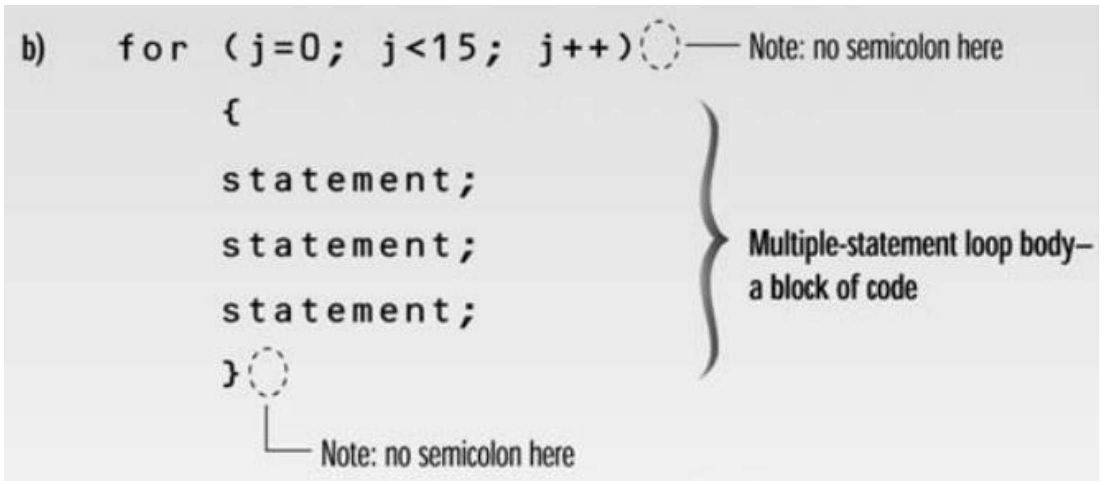


Figure: Syntax of the for loop

The next example, CUBELIST, uses three statements in the loop body. It prints out the cubes of the numbers from 1 to 10, using a two-column format.

```
// cubelist.cpp

// lists cubes from 1 to 10

#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

        int numb;       //define loop variable

        for(numb=1; numb<=10; numb++)           //loop from 1 to 10

        {

                cout << setw(4) << numb;        //display 1st column

                int cube = numb*numb*numb;      //calculate cube

                cout << setw(6) << cube << endl;  //display 2nd column

        }

        return 0;

}
```

Here's the output from the program:

```
 1      1
 2      8
 3     27
 4     64
 5    125
 6    216
 7    343
 8    512
 9    729
10   1000
```

The factorial is calculated by multiplying the original number by all the positive integers smaller than itself. Thus the factorial of 5 is 5*4*3*2*1, or 120.

```
// factor.cpp
// calculates factorials, demonstrates FOR loop
#include<iostream>
using namespace std;
int main()
{       unsigned int numb;
        unsigned long fact=1; //long for larger numbers
        cout << “Enter a number: “;
        cin >> numb;            //get number
        for(int j=numb; j>0; j--)       //multiply 1 by
                fact *= j;              //numb, numb-1, ..., 2, 1
        cout << “Factorial is “ << fact << endl;
        return 0;
}
```

Here’s the output from the program:

```
Enter a number: 10
Factorial is 3628800
```

The while Loop

```
// calculate and print a table of squares 0–99
int main()
{
        int i = 0;       // start from 0
        while (i<100){
                cout<< i << '\t' << i*i << '\n';
                ++i;            // increment i (that is, i becomes i+1)
        }
}
```

```
// endon0.cpp
// demonstrates WHILE loop
#include<iostream>
using namespace std;
int main()
{
        int n = 99;      // make sure n isn't initialized to 0
        while( n != 0 ) // loop until n is 0
                cin >> n; // read a number into n
        cout << endl;
        return 0;
}
```

Here's some sample output. The user enters numbers, and the loop continues until 0 is entered, at which point the loop and the program terminate.

```
        1
        27
        33
        144
```

```
9
0
```

```
// prints numbers raised to fourth power
#include<iostream>
#include<iomanip>     //for setw
using namespace std;
int main()
{
        int pow=1;                  //power initially 1
        int numb=1;                 //numb goes from 1 to ???
        while( pow<=10000)  //loop while power <= 4 digits
        {
                cout << setw(2) << numb;                //display number
                cout << setw(5) << pow << endl;       //display fourth power
                ++numb;                                         //get ready for next power
                pow = numb*numb*numb*numb;     //calculate fourth power
        }
        cout << endl;
        return 0;
}
```

Here’s the output:

```
 1    1
 2   16
 3   81
 4  256
 5  625
 6 1296
```

```
    7       2401
    8       4096
    9       6561
```

The next program touches on the question of operator precedence. It generates the famous sequence of numbers called the Fibonacci series. Here are the first few terms of the series:

1 1 2 3 5 8 13 21 34 55

Each term is found by adding the two previous ones: 1+1 is 2, 1+2 is 3, 2+3 is 5, 3+5 is 8, and so on.

```
// fibo.cpp
// demonstrates WHILE loops using fibonacci series
#include<iostream>
using namespace std;
int main()
{                               //largest unsigned long const
        unsigned long limit = 4294967295;
        unsigned long next=0;           //next-to-last term
        unsigned long last=1;           //last term
        while( next < limit / 2 )       //don’t let results get too big
        {
                cout << last << “ “;            //display last term
                long sum = next + last;         //add last two terms
                next = last;                    //variables move forward
                last = sum;                     // in the series
        }
        cout << endl;
        return 0;
}
```

Here’s the output:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465
14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437
701408733 1134903170 1836311903 2971215073
```

The do Loop

```
// divdo.cpp

// demonstrates DO loop

#include<iostream>

using namespace std;

int main()

{

        long dividend, divisor;

        char ch;

        do      //start of do loop

        {       //do some processing

                cout << “Enter dividend: “;

                cin >> dividend;

                cout << “Enter divisor: “;

                cin >> divisor;

                cout << “Quotient is “ << dividend / divisor;

                cout << “, remainder is “ << dividend % divisor;

                cout << “\nDo another? (y/n): “;        //do it again?

                cin >> ch;

        } while( ch != ‘n’ );           //loop condition

        return 0;

}
```

Here’s an example of DIVDO’s output:

```
Enter dividend: 11
Enter divisor: 3
Quotient is 3, remainder is 2
Do another? (y/n): y
Enter dividend: 222
Enter divisor: 17
Quotient is 13, remainder is 1
Do another? (y/n): n
```

Here’s an example, PRIME, that nests an if within a for loop. This example tells you whether a number you enter is a prime number. (Prime numbers are integers divisible only by themselves and 1. The first few primes are 2, 3, 5, 7, 11, 13, 17.)

```
// prime.cpp
// demonstrates IF statement with prime numbers
#include<iostream>
using namespace std;
#include<process.h>    //for exit()
int main()
{
        unsigned long n, j;
        cout << “Enter a number: “;
        cin >> n;        //get number to test
        for(j=2; j <= n/2; j++)          //divide by every integer from
                if(n%j == 0)             //2 on up; if remainder is 0,
                {                        //it’s divisible by j
                        cout << “It’s not prime; divisible by “ << j << endl;
                        exit(0);                 //exit from the program
                }
        cout << “It’s prime\n”;
        return 0;
}
```

Here’s output from three separate invocations of the program:

```
Enter a number: 13
It’s prime
Enter a number: 22229
It’s prime
Enter a number: 22231
It’s not prime; divisible by 11
```

The getche( ) Library Function

```
// chcount.cpp
// counts characters and words typed in
#include<iostream>
using namespace std;
#include<conio.h>                //for getche()
int main()
{
        int chcount=0;           //counts non-space characters
        int wdcount=1;           //counts spaces between words
        char ch = ‘a’;           //ensure it isn’t ‘\r’
        cout << “Enter a phrase: “;
        while( ch != ‘\r’ )              //loop until Enter typed
        {
                ch = getche();           //read one character
                if( ch = =’ ‘ )          //if it’s a space
                        wdcount++;       //count a word
        else            //otherwise, c
            chcount++;      //count a character
    }               //display results
    cout << “\nWords=” << wdcount << endl << “Letters=” << (chcount-1) << endl;
    return 0;
}
```

Here’s some sample interaction with CHCOUNT:

```
For while and do
Words=4 Letters=13
```

Here’s the rewritten version, called CHCNT2:

```
// chcnt2.cpp
// counts characters and words typed in
#include<iostream>
using namespace std;
#include<conio.h>      // for getche()
int main()
{   int chcount=0;
    int wdcount=1;          // space between two words
    char ch;
    while( (ch=getche()) != ‘\r’ )          // loop until Enter typed
    {   if( ch==’ ‘ )               // if it’s a space
            wdcount++;              // count a word
        else                    // otherwise,
            chcount++;      // count a character
    }                           // display results
    cout << “\nWords=” << wdcount << endl << “Letters=” << chcount << endl;
    return 0;
}
```

The Logical Operator

There are three logical operators in C++:

Operator	*Effect*
&&	Logical AND
||	Logical OR
!	Logical NOT

There is no logical XOR (exclusive OR) operator in C++.

Logical AND Operator

Here's an example, ADVENAND, that uses a logical operator to spruce up the adventure game from the ADSWITCH example. We'll bury some treasure at coordinates (7,11) and see whether the player can find it.

```
// advenand.cpp

// demonstrates AND logical operator

#include<iostream>

using namespace std;

#include<process.h>            //for exit()

#include<conio.h>              //for getche()

int main()

{

        char dir='a';

        int x=10, y=10;

        while( dir != '\r' )

        {

                cout << "\nYour location is " << x << ", " << y;

                cout << "\nEnter direction (n, s, e, w): ";

                dir = getche();          //get direction

                switch(dir) {

                                case 'n': y--;

                                        break;          //update coordinates
```

```
                        case ‘s’: y++;
                                break;
                        case ‘e’: x++;
                                break;
                        case ‘w’: x--;
                                break;
                        }
                        if( x==7 && y==11 )          //if x is 7 and y is 11
                        { cout << nYou found the treasure!\n”;
                          exit(0);                  //exit from program }
                    } //end switch
                } return 0;
            }
```

Here’s some interaction as the user arrives at these coordinates:

```
Your location is 7, 10
Enter direction (n, s, e, w): s
You found the treasure!
```

Logical OR Operator

Suppose in the adventure game you decide there will be dragons if the user goes too far east or too far west. Here’s an example, ADVENOR, that uses the logical OR operator to implement this frightening impediment to free adventuring. It’s a variation on the ADVENAND program.

```
// advenor.cpp
// demonstrates OR logical operator
#include<iostream>
using namespace std;
#include<process.h>          //for exit()
#include<conio.h>            //for getche()
int main()
{
    char dir='a';
    int x=10, y=10;
    while( dir != '\r' )            //quit on Enter key
    {
        cout << "\n\nYour location is " << x << ", " << y;
        if( x<5 || x>15 )           //if x west of 5 OR east of 15
            cout << "\nBeware: dragons lurk here";
        cout << "\nEnter direction (n, s, e, w): ";
        dir = getche();             //get direction
        switch(dir)
        {
            case 'n': y--;
                break;              //update coordinates
            case 's': y++;
                break;
            case 'e': x++;
                break;
            case 'w': x--;
                break;
        } //end switch
    } //end while
    return 0;
} //end main()
```

Precedence Summary

Operators with higher precedence are evaluated before those with lower precedence. Operators on the same row have equal precedence. You can force an expression to be evaluated first by placing parentheses around it.

Operator type	*Operators*	*Precedence*
Unary	!, ++, --, +, -	Highest
Arithmetic	Multiplicative *, /, %	
	Additive +, -	
Relational	Inequality <, >, <=, >=	
	Equality ==, !=	
Logical	And &&	
	Or ||	
Conditional	?:	
Assignment	=, +=, -=, *=, /=, %=	Lowest

The Continue Statement

The break statement takes you out of the bottom of a loop. Sometimes, however, you want to go back to the top of the loop when something unexpected happens. Executing continue has this effect.

```
// divdo2.cpp

// demonstrates CONTINUE statement

#include<iostream>

using namespace std;

int main()

{

        long dividend, divisor;

        char ch;

        do

        {

                cout << “Enter dividend: “;

                cin >> dividend;

                cout << “Enter divisor: “;

                cin >> divisor;

                if( divisor == 0 )        //if attempt to
                {                   //divide by 0,
                    cout << “Illegal divisor\n”;    //display message
                    continue;                       //go to top of loop
                }
                cout << “Quotient is “ << dividend / divisor;
                cout << “, remainder is “ << dividend % divisor;
                cout << “\nDo another? (y/n): “;
                cin >> ch;
        } while( ch != ‘n’ );
        return 0;
}
```

Here’s some sample output:

```
Enter dividend: 10
Enter divisor: 0
Illegal divisor
Enter dividend:
```

The goto Statement

You insert a label in your code at the desired destination for the goto. The label is always terminated by a colon.

```
goto SystemCrash;
// other statements
SystemCrash:
// control will begin here following goto
```

Questions

1. A relational operator

    a. assigns one operand to another.

    b. yields a Boolean result.

    c. compares two operands.

d. logically combines two operands.

2. Write an expression that uses a relational operator to return true if the variable george is not equal to sally.

3. Is –1 true or false?

4. True or false: The increment expression in a for loop can decrement the loop variable.

5. Write a for loop that displays the numbers from 100 to 110.

6. A block of code is delimited by ______.

7. Write a while loop that displays the numbers from 100 to 110.

8. True or false: Relational operators have a higher precedence than arithmetic operators.

9. How many times is the loop body executed in a do loop?

10. Write a do loop that displays the numbers from 100 to 110.

11. Write an if statement that prints Yes if a variable age is greater than 21.

12. The library function exit() causes an exit from

   a. the loop in which it occurs.

   b. the block in which it occurs.

   c. the function in which it occurs.

   d. the program in which it occurs.

13. Write an if...else statement that displays Yes if a variable age is greater than 21, and displays No otherwise.

14. The getche() library function

   a. returns a character when any key is pressed.

   b. returns a character when Enter is pressed.

   c. displays a character on the screen when any key is pressed.

   d. does not display a character on the screen.

15. What is the character obtained from cin when the user presses the Enter key?

16. An else always matches the _____ if, unless the if is _____.

17. The else...if construction is obtained from a nested if...else by _______.

18. Write a switch statement that prints Yes if a variable ch is ‘y’, prints No if ch is ‘n’, and prints Unknown response otherwise.

19. Write a statement that uses a conditional operator to set ticket to 1 if speed is greater than 55, and to 0 otherwise.

20. The && and || operators

a. compare two numeric values.

b. combine two numeric values.

c. compare two Boolean values.

d. combine two Boolean values.

21. Write an expression involving a logical operator that is true if limit is 55 and speed is greater than 55.

22. Arrange in order of precedence (highest first) the following kinds of operators: logical, unary, arithmetic, assignment, relational, conditional.

23. The break statement causes an exit

a. only from the innermost loop.

b. only from the innermost switch.

c. from all loops and switches.

d. from the innermost loop or switch.

24. Executing the continue operator from within a loop causes control to go to ________.

25. The goto statement causes control to go to

a. an operator.

b. a label.

c. a variable.

d. a function.

Exercises

1. Assume that you want to generate a table of multiples of any given number. Write a program that allows the user to enter the number and then generates the table, formatting it into 10 columns and 20 lines. Interaction with the program should look like this (only the first three lines are shown):

```
Enter a number: 7
     7   14   21   28   35   42   49   56   63   70
    77   84   91   98  105  112  119  126  133  140
   147  154  161  168  175  182  189  196  203  210
```

2. Write a temperature-conversion program that gives the user the option of converting Fahrenheit to Celsius or Celsius to Fahrenheit. Then carry out the conversion. Use floating-point numbers. Interaction with the program might look like this:

Type 1 to convert Fahrenheit to Celsius,

        2 to convert Celsius to Fahrenheit: 1

Enter temperature in Fahrenheit: 70

In Celsius that's 21.111111

3. Operators such as >>, which read input from the keyboard, must be able to convert a series of digits into a number. Write a program that does the same thing. It should allow the user to type up to six digits, and then display the resulting number as a type long integer. The digits should be read individually, as characters, using getche(). Constructing the number involves multiplying the existing value by 10 and then adding the new digit. (Hint: Subtract 48 or '0' to go from ASCII to a numerical digit.)

Here's some sample interaction:

Enter a number: 123456

Number is: 123456

4. Create the equivalent of a four-function calculator. The program should ask the user to enter a number, an operator, and another number. (Use floating point.) It should then carry out the specified arithmetical operation: adding, subtracting, multiplying, or dividing the two numbers. Use a switch statement to select the operation. Finally, display the result. When it finishes the calculation, the program should ask whether the user wants to do another calculation. The response can be 'y' or 'n'. Some sample interaction with the program might look like this:

Enter first number, operator, second number: 10 / 3

Answer = 3.333333

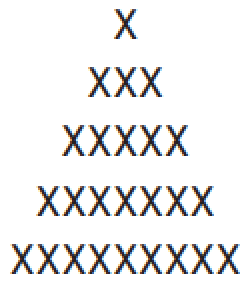
Do another (y/n)? y

Enter first number, operator, second number: 12 + 100

Answer = 112

Do another (y/n)? N

5. Use for loops to construct a program that displays a pyramid of Xs on the screen. The pyramid should look like this



```
    X
   XXX
  XXXXX
 XXXXXXX
XXXXXXXXX
```

except that it should be 20 lines high, instead of the 5 lines shown here. One way to do this is to nest two inner loops, one to print spaces and one to print Xs, inside an outer loop that steps down the screen from line to line.

6. Modify the FACTOR program in this chapter so that it repeatedly asks for a number and calculates its factorial, until the user enters 0, at which point it terminates. You can enclose the relevant statements in FACTOR in a while loop or a do loop to achieve this effect.

7. Write a program that calculates how much money you'll end up with if you invest an amount of money at a fixed interest rate, compounded yearly. Have the user furnish the initial amount, the number of years, and the yearly interest rate in percent. Some interaction with the program might look like this:

Enter initial amount: 3000

Enter number of years: 10

Enter interest rate (percent per year): 5.5

At the end of 10 years, you will have 5124.43 dollars.

At the end of the first year you have 3000 + (3000 * 0.055), which is 3165. At the end of the second year you have 3165 + (3165 * 0.055), which is 3339.08. Do this as many times as there are years. A for loop makes the calculation easy.

8. Suppose you give a dinner party for six guests, but your table seats only four. In how many ways can four of the six guests arrange themselves at the table? Any of the six guests can sit in the first chair. Any of the remaining five can sit in the second chair. Any of the remaining four can sit in the third chair, and any of the remaining three can sit in the fourth chair. (The last two will have to stand.) So the number of possible arrangements of six guests in four chairs is 6*5*4*3, which is 360. Write a program that calculates the number of possible arrangements for any number of guests and any number of chairs. (Assume there will never be fewer guests than chairs.) Don't let this get too complicated. A simple for loop should do it.

9. Write another version of the program from Exercise 7 so that, instead of finding the final amount of your investment, you tell the program the final amount and it figures out how many years it will take, at a fixed rate of interest compounded yearly, to reach this amount. What sort of loop is appropriate for this problem? (Don't worry about fractional years; use an integer value for the year.)

10. Create a four-function calculator for fractions. Here are the formulas for the four arithmetic operations applied to fractions:

Addition: a/b + c/d = (a*d + b*c) / (b*d)

Subtraction: a/b - c/d = (a*d - b*c) / (b*d)

Multiplication: a/b * c/d = (a*c) / (b*d)

Division: a/b / c/d = (a*d) / (b*c)

The user should type the first fraction, an operator, and a second fraction. The program should then display the result and ask whether the user wants to continue.

Chapter(4)

Arrays

Array Fundamentals

In computer languages we also need to group together data items of the same type. The most basic mechanism that accomplishes this in C++ is the array. Arrays can hold a few data items or tens of thousands. The data items grouped in an array can be simple types such as int or float, or they can be user-defined types such as structures and objects.

```
#include<iostream>

using namespace std;

int main()

{

        int age[4];

        for(int j=0; j<4; j++)

        {

                cout<< “Enter an age: “;

                cin >> age[j];

        }

        for(j=0; j<4; j++)

                cout<< “You entered “ << age[j] << endl;

                return 0;

}
```

Here’s a sample interaction with the program:

```
Enter an age: 44

Enter an age: 16

Enter an age: 23

Enter an age: 68

You entered 44

You entered 16

You entered 23
```

You entered 68

Figure 4.1: Syntax of array definition

Array Elements

The items in an array are called elements (in contrast to the items in a structure, which are called members). As we noted, all the elements in an array are of the same type; only the values vary. Figure 4.2 shows the elements of the array age.

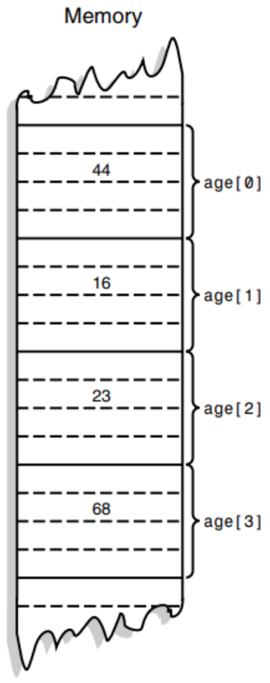


Figure: 4.2 Array elements

Averaging Array Elements

Here's another example of an array at work. This one, SALES, invites the user to enter a series of six values representing widget sales for each day of the week (excluding Sunday), and then calculates the average of these values. We use an array of type double so that monetary values can be entered. // sales.cpp // averages a weeks's widget sales (6 days)

```
#include<iostream>

using namespace std;

int main()

{

        const int SIZE = 6;                //size of array
        double sales[SIZE];             //array of 6 variables
        cout << “Enter widget sales for 6 days\n”;
        for(int j=0; j<SIZE; j++)       //put figures in array
                cin>> sales[j];
        double total = 0;
        for(j=0; j<SIZE; j++)           //read figures from array
                total+=sales[j];        //to find total
        double average=total/SIZE;      //find average
        cout<< “Average = “ << average << endl;
        return 0;
}
```

Here’s some sample interaction with SALES:

```
Enter widget sales for 6 days
352.64
867.70
781.32
867.35
746.21
189.45
Average = 634.11
```

Initializing Arrays

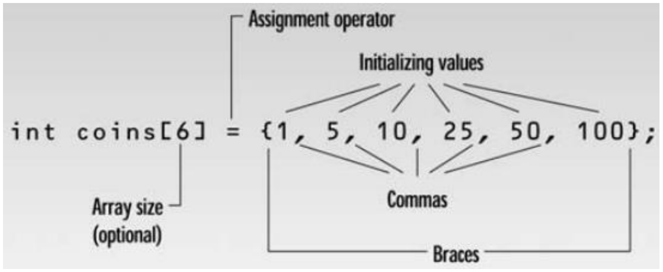


Figure 4.3: Syntax of array initialization

Here’s an example, DAYS, that sets 12 array elements in the array days_per_month to the number of days in each month.

```
// days.cpp

// shows days from start of year to date specified

#include<iostream>

using namespace std;

int main()

{

        int month, day, total_days;

        int days_per_month[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

        cout << “\nEnter month (1 to 12): “;            //get date

        cin >> month;

        cout << “Enter day (1 to 31): “;

        cin >> day;

        total_days = day;       //separate days

        for(int j=0; j<month-1; j++)

                total_days+=days_per_month[j];

        cout<< “Total days from start of year is: “ << total_days

                << endl;

         return 0;

}
```

The program calculates the number of days from the beginning of the year to a date specified by the user. (Beware: It doesn’t work for leap years.)

Here’s some sample interaction:

```
Enter month (1 to 12): 3

Enter day (1 to 31): 11

Total days from start of year is: 70
```

Multidimentional Arrays

Here’s a program, SALEMON, that uses a two-dimensional array to store sales figures for several districts and several months:

```
// salemon.cpp

// displays sales chart using 2-d array

#include<iostream>

#include<iomanip>               //for setprecision, etc.

using namespace std;


const int DISTRICTS = 4;      //array dimensions

const int MONTHS = 3;

int main()

{

        int d, m;

        double sales[DISTRICTS][MONTHS];        //two-dimensional array

        cout << endl;

        for(d=0; d<DISTRICTS; d++)              //get array values

                for(m=0; m<months; m++)

                {

                        cout<< “Enter sales for district “ << d+1;

                        cout << “, month “ << m+1 << “: “;

                        cin >> sales[d][m];         //put number in array

                }

        cout << “\n\n”;

        cout << “ Month\n”;

        cout << “ 1 2 3”;
```

```
    for(d=0; d<DISTRICTS; d++)
    {
        cout<<"\nDistrict "<< d+1;
        for(m=0;m<MONTHS;m++)
            cout<<setiosflags(ios::fixed)          //not exponential
                << setiosflags(ios::showpoint)     //always use point
                << setprecision(2)                 //digits to right
                << setw(10)                        //field width
                << sales[d][m];                    //get number from array
    }           //end for(d)
    cout << endl;
    return 0;
}       //end main
```

This program accepts the sales figures from the user and then displays them in a table.

```
Enter sales for district 1, month 1: 3964.23
Enter sales for district 1, month 2: 4135.87
Enter sales for district 1, month 3: 4397.98
Enter sales for district 2, month 1: 867.75
Enter sales for district 2, month 2: 923.59
Enter sales for district 2, month 3: 1037.01
Enter sales for district 3, month 1: 12.77
Enter sales for district 3, month 2: 378.32
Enter sales for district 3, month 3: 798.22
Enter sales for district 4, month 1: 2983.53
Enter sales for district 4, month 2: 3983.73
Enter sales for district 4, month 3: 9494.98
```

	Month		
	1	2	3
District 1	3964.23	4135.87	4397.98
District 2	867.75	923.59	1037.01
District 3	12.77	378.32	798.22
District 4	2983.53	3983.73	9494.98

Defining Multidimensional Arrays

The array is defined with two size specifiers, each enclosed in brackets:

```
double sales[DISTRICTS][MONTHS];
```

You can think about sales as a two-dimensional array, laid out like a checkerboard. Another way to think about it is that sales is an array of arrays. It is an array of DISTRICTS elements, each of which is an array of MONTHS elements. Figure 4.4 shows how this looks.

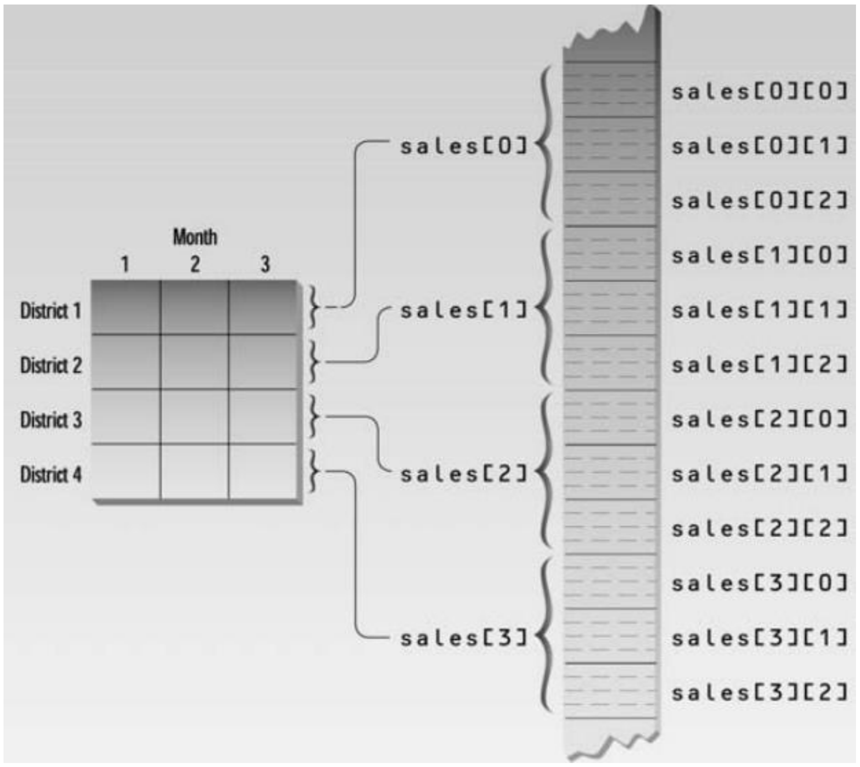


Figure: 4.4 Two-dimensional Array

Initializing Multidimensional Arrays

Here's a variation of the SALEMON program that uses an initialized array instead of asking for input from the user. This program is called SALEINIT.

```
// saleinit.cpp
// displays sales chart, initializes 2-d array
#include<iostream>
#include<iomanip>                  //for setprecision, etc.
using namespace std;
const int DISTRICTS = 4;      //array dimensions
const int MONTHS = 3;
int main()
{
        int d, m;          //initialize array elements
        double sales[DISTRICTS][MONTHS]={     { 1432.07, 234.50, 654.01 },
                                              { 322.00, 13838.32, 17589.88 },
                                              { 9328.34, 934.00, 4492.30 },
                                              { 12838.29, 2332.63, 32.93 } };
        cout << “\n\n”;
        cout << “ Month\n”;
        cout << “ 1 2 3”;
        for(d=0;d<DISTRICTS;d++)
         {
                cout<<”\nDistrict << d+1;
                for(m=0;m<MONTHS;m++)
                        cout<< setw(10) << setiosflags(ios::fixed)
                             << setiosflags(ios::showpoint) << setprecision(2)
                             << sales[d][m];              //access array element
        }
        cout << endl;
        return 0;
}
```

Questions

1. An array element is accessed using

    a. a first-in-first-out approach.

    b. the dot operator.

    c. a member name.

    d. an index number.

2. All the elements in an array must be the _________ data type.

3. Write a statement that defines a one-dimensional array called doubleArray of type double that holds 100 elements.

4. The elements of a 10-element array are numbered from ________ to ________.

5. Write a statement that takes element j of array doubleArray and writes it to cout with the insertion operator.

6. Element doubleArray[7] is which element of the array?

    a. The sixth

    b. The seventh

    c. The eighth

    d. Impossible to tell

7. Write a statement that defines an array coins of type int and initializes it to the values of the penny, nickel, dime, quarter, half-dollar, and dollar.

8. When a multidimensional array is accessed, each array index is

    a. separated by commas.

    b. surrounded by brackets and separated by commas.

    c. separated by commas and surrounded by brackets.

    d. surrounded by brackets.

9. Write an expression that accesses element 4 in subarray 2 in a two-dimensional array called twoD.

10. True or false: In C++ there can be an array of four dimensions.

11. For a two-dimensional array of type float, called flarr, write a statement that declares the array and initializes the first subarray to 52, 27, 83; the second to 94, 73, 49; and the third to 3, 6, 1.

12. Write a program that allows the user to input a number of integers, and then stores them in an int array. Then, display the largest element in the array.

Chapter (5)

Functions

A function groups a number of program statements into a unit and gives it a name. This unit can then be invoked from other parts of the program. The function's code is stored in only one place in memory, even though the function is executed many times in the course of the program. Figure 5.1 shows how a function is invoked from different sections of a program.

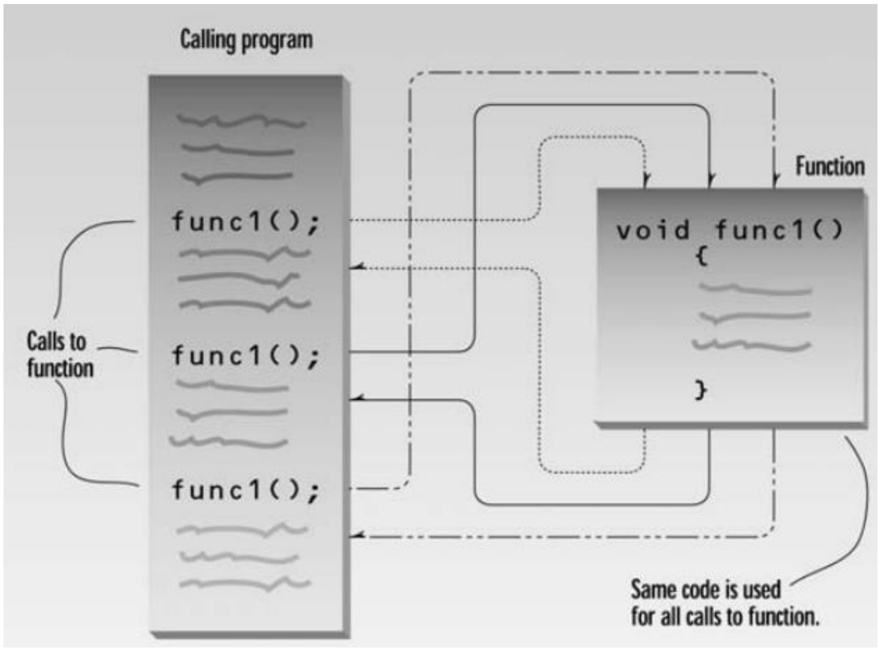


Figure: Flow of control to a function

Simple Functions

Our first example demonstrates a simple function whose purpose is to print a line of 45 asterisks. The example program generates a table, and lines of asterisks are used to make the table more readable. Here's the listing for TABLE:

```
// table.cpp

// demonstrates simple function

#include<iostream>

using namespace std;

void starline();                    //function declaration

                                    // (prototype)
```

```
int main()
{
        starline();                     //call to function
        cout << “Data type Range” << endl;
        starline();                     //call to function
        cout << “char -128 to 127” << endl
             << “short -32,768 to 32,767” << endl
             << “int System dependent” << endl
             << “long -2,147,483,648 to 2,147,483,647” << endl;
        starline();                     //call to function
        return 0;
}
// starline()
// function definition
void starline()                         //function declarator
{
        for(int j=0; j<45; j++)
                cout<< ‘*’;
        cout << endl;
}
```

The output from the program looks like this:

```
*********************************************
Data type     Range
*********************************************
char          -128 to 127
short         -32,768 to 32,767
int           System dependent
long          -2,147,483,648 to 2,147,483,647
*********************************************
```

The Function Declaration

The approach we show here is to declare the function before it is called. In the TABLE program, the function starline() is declared in the line

```
void starline();
```

The keyword void specifies that the function has no return value, and the empty parentheses indicate that it takes no arguments. Notice that the function declaration is terminated with a semicolon.

Calling the Function

The function is called (or invoked, or executed) three times from main(). Each of the three calls looks like this:

```
starline();
```

This is all we need to call the function: the function name, followed by parentheses. The syntax of the call is very similar to that of the declaration, except that the return type is not used. The call is terminated by a semicolon. Executing the call statement causes the function to execute; that is, control is transferred to the function, the statements in the function definition are executed, and then control returns to the statement following the function call.

The Function Definition

Finally we come to the function itself, which is referred to as the function definition. The definition contains the actual code for the function. Here's the definition for starline():

```
        void starline()                        //function declarator
{
        for(int j=0; j<45; j++)
                cout<< '*';
        cout << endl;
}
```

The definition consists of a line called the declarator, followed by the function body. The function body is composed of the statements that make up the function, delimited by braces.

The declarator must agree with the declaration: It must use the same function name, have the same argument types in the same order (if there are arguments), and have the same return type.

Notice that the declarator is not terminated by a semicolon.

Passing Arguments to Functions

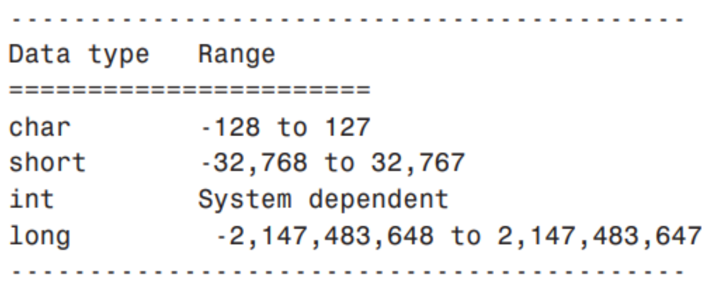
An argument is a piece of data (an int value, for example) passed from a program to the function. Arguments allow a function to operate with different values, or even to do different things, depending on the requirements of the program calling it.

Passing Constants

Here's a program, TABLEARG, that incorporates just such a function. We use arguments to pass the character to be printed and the number of times to print it.

```
// tablearg.cpp

// demonstrates function arguments

#include<iostream>

using namespace std;

void repchar(char, int);          //function declaration

int main()

{

        repchar(‘-’, 43);              //call to function

        cout << “Data type Range” << endl;

        repchar(‘=’, 23);              //call to function

        cout << “char -128 to 127” << endl

              << “short -32,768 to 32,767” << endl

              << “int System dependent” << endl

              << “double -2,147,483,648 to 2,147,483,647” << endl;

        repchar(‘-’, 43);              //call to function

        return 0;

}


// repchar()

// function definition

void repchar(char ch, int n)          //function declarator

{
        for(int j=0; j<n; j++)
            cout<< ch;
        cout << endl;
}
```

Here’s the output from TABLEARG:



```
-----------------------------------------
Data type   Range
======================
char        -128 to 127
short       -32,768 to 32,767
int         System dependent
long         -2,147,483,648 to 2,147,483,647
-----------------------------------------
```

Passing Variables

This program, VARARG, incorporates the same repchar() function as did TABLEARG, but lets the user specify the character and the number of times it should be repeated.

```
// vararg.cpp
// demonstrates variable arguments
#include<iostream>
using namespace std;
void repchar(char, int);              //function declaration
int main()
{
    char chin;
    int nin;
    cout << “Enter a character: “;
    cin >> chin;
    cout << “Enter number of times to repeat it: “;
    cin >> nin;
    repchar(chin, nin);
    return 0;
}
// repchar()
// function definition
void repchar(char ch, int n)              //function declarator
{
        for(int j=0; j<n; j++)
                cout<< ch;
        cout << endl;
}
```

Here’s some sample interaction with VARARG:

```
Enter a character: +
Enter number of times to repeat it: 20
++++++++++++++++++++
```