12

INTRODUCTION TO C++

330en12

Notes

In the previous lesson you have learnt about open source softwares. In this lesson you will learn about C++ programming. You may know that C++ is an extension to C Programming language. It was developed at AT&T Bell Laboratories in the early 1980s by Bjarne Stroustrup. It is a deviation from traditional procedural languages in the sense that it follows Object Oriented Programming (OOP) approach which is quite suitable for managing large and complex programs.

OBJECTIVES

After reading this lesson, you will be able to:

● learn about C++ character set, tokens and basic data types;

● explain the utility of different types of operators used in C++;

● identify the difference between implicit and explicit conversions;

● explain about Input/Output streams supported by C++;

● explain the structure of a C++ program;

● write a simple program in C++.

12.1 C++ CHARACTER SET

Character set is a set of valid characters that a language can recognize. A character represents any letter, digit or any other special character. The C++ programming language also has some character set. Now let us learn C++ language character set.

Table 12.1: C++ character set

Letters	A–Z, a–z
Digits	0–9
Special Characters	Space + - * / ^ \ ( ) [ ] { } = ! = < > . ' " $, ; : % ! & ? _ # <= >= @
White spaces	Horizontal tab (→), Blank space, Carriage return (<) Newline, form feed

12.2 BASIC DATA TYPES

Now you will learn about basic data types used in C++ language. Every program specifies a set of operations to be done on some data in a particular sequence. However, the data can be of many types such as a number, a character, boolean value etc. C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character type. A brief discussion on these types is given below:

Table 12.2: Data Types with Range

Data Type	Range
char	–128 to 127
int	–32768 to 32767
short int	–32768 to 32767
long int	–2147483648 to 2147483647
float	$–3.4 \times 10^{-38}$ to 3.4×10^{38}
double	1.7×10^{-308} to 1.7×10^{308}
long double	1.7×10^{-4932} to $1.7 \times 10^{+4932}$

Integer type (int)

An integer is an integral whole number without a decimal point. These numbers are used for counting. For example 26, 373, –1729 are valid integers. Normally an integer can hold numbers from –32768 to 32767. However, if the need be, a long integer (long int) can also be used to hold integers from –2, 147, 483, 648 to 2, 147, 483, 648.

Floating point type (float)

A floating point number has a decimal point. Even if it has an integral value, it must include a decimal point at the end. These numbers are used for measuring quantities.

Examples of valid floating point numbers are: 27.4, -927., 40.03

A float type data can be used to hold numbers from $3.4*10^{-38}$ to $3.4*10^{+38}$ with six or seven digits of precision. However, for more precision a double precision type (double) can be used to hold numbers from $1.7*10^{-308}$ to $1.7*10^{+308}$ with about 15 digits of precision.

Character Type (Char)

It is a non numeric data type consisting of single alphanumeric character. Examples of valid character types are : 'A', '9', 'P', '8', '&'.

It may be noted here that 9 and '9' are of different data types. The former is of type int and later of type char.

Notes

12.3 TOKENS

A token is a group of characters that logically belong together. The programmer can write a program by using tokens. C++ uses the following types of tokens.

- Keywords
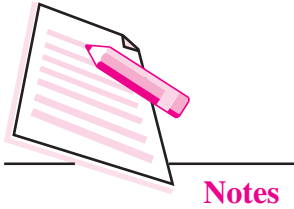- Identifiers
- Literals
- Punctuators
- Operators

12.3.1 Keywords

There are some reserved words in C++ which have predefined meaning to complier called keywords. Some commonly used keywords are given below:

List of Keywords

Table 12.3: List of keywords

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extem	protected	try
catch	float	public	typedef
char	for	register	union
class	friend	return	unsigned
const	goto	short	virtual
continue	if	signed	void
default	inline	sizeof	volatile
delete	int	static	while
do	long	struct	

Notes

12.3.2 Identifiers

Symbolic names can be used in C++ for various data items used by a programmer in his/her program. For example, if you want to store a value 50 in a memory location, you can choose any symbolic name (say MARKS) and use it as given below:

MARKS = 50

The symbol '=' is an assignment operator. The significance of the above statement is that 'MARKS' is a symbolic name for a memory location where the value 50 is being stored.

A symbolic name is generally known as an identifier. The identifier is a sequence of characters taken from C++ character set. The rules for the formation of an identifier are:

(i)   An identifier can consist of alphabets, digits and/or underscores.

(ii)   It must not start with a digit.

(iii)   C++ is case sensitive, i.e., upper case and lower case letters are considered different from each other. It may be noted that TOTAL and total are two different identifier names.

(iv)   It should not be a reserved word (keywords).

12.3.3 Literals

Literals (often referred to as constants) are data items that never change their value during the execution of the program. The following types of literals are available in C++.

(i)   integer-constants

(ii)   character-constants

(iii)   floating-constants

(iv)   string-literals

Integer constants

Integer constants are whole numbers without any fractional part. It may contain either + or – sign, but decimal point or commas does not appear in any integer constant. C++ allows three types of integer constants.

1.   decimal (Base 10)

2.   octal (Base 8)

3.   hexadecimal (Base 16)

Decimal integer constants

It consists of sequence of digits and should not begin with 0 (zero). For example 124, - 179, + 108.

Octal integer constants

It consists of sequence of digits starting with 0 (zero). For example, 014, 012.

Hexadecimal integer constant

It consists of sequence of digits preceded by ox or OX. For example OXD, OXC. The suffix l or L and u or U attached to any constant forces it to be represented as a long and unsigned respectively.

Character constants

A character constant in C++ must contain one or more characters and must be enclosed in single quotation marks. For example 'A', '9', etc. C++ allows non-graphic characters which cannot be typed directly from keyboard, e.g., backspace, tab, carriage return etc. These characters can be represented by using an escape sequence. An escape sequence represents a single character. The following table gives a listing of common escape sequences.

Table 12.4: List of escape sequence

Escape Sequence	Nongraphic Character
\a	Bell (beep)
\b	Backspace
\f	Formfeed
\n	Newline or line feed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\?	Question mark
\\	Backslash
\'	Single quote
\"	Double quote
\xhh	Hexadecimal number (hh represents the number in hexadecimal
\000	Octal number (00 represents the number in octal
\0	Null

Notes

Notes

Floating constants

Floating constants are also called real constants. These numbers have fractional parts. They may be written in fractional form or exponent form. A real constant in fractional form consists of signed or unsigned digits including a decimal point between digits. For example 3.0, -17.0, -0.627 etc.

A real constant in exponent form has two parts: a mantissa and an exponent. The mantissa is either an integer or a real constant followed by letter E or e and the exponent which must be an integer. For example 2E03, 1.23E07.

String Literals

A sequence of character enclosed within double quotes is called a string literal. String literal is by default (automatically) added with a special character '\O' which denotes the end of the string. Therefore the size of the string is increased by one character. For example "COMPUTER" will be represented as "COMPUTER\O" in the memory and its size is 9 characters.

12.3.4 Punctuators

The following characters are used as punctuators in C++.

Brackets [ ]	opening and closing brackets indicate single and multidimensional array subscript.
Parentheses ( )	opening and closing brackets indicate functions calls, function parameters for grouping expressions etc.
Braces { }	opening and closing braces indicate the start and end of a compound statement.
Comma ,	it is used as a separator in a function argument list.
Semicolon ;	it is used as a statement terminator.
Colon :	it indicates a labelled statement or conditional operator
Asterisk *	it is used in pointer declaration or as multiplication operator.
Equal sign =	it is used as an assignment operator.
Pound sign #	it is used as pre-processor directive.

12.3.5 Operators

Operators are special symbols used for specific purposes. C++ includes many operators.

- Arithmetical operators
- Relational operators

- Logical operators
- Unary operators
- Assignment operators
- Conditional operators
- Comma operator

Arithmetical operators

An operator that performs an arithmetic (numeric) operation +, -, *, / , or %. For these operations always two or more than two operands are required. Therefore these operators are called binary operators. The following table shows the arithmetic operators.

Operators	Meaning	Example	Result
+	Addition	8 + 5	13
–	Subtraction	8 – 5	3
×	Multiplication	8 × 5	40
/	Division	8 / 5	1
%	Modulus/Remainder	8 % 5	3

Note : There is no operator which gives exponent in C++.

Modulus operator will return the remainder value after the division. 8 % 5 will return 3 because when you divide 8 by 5, the remainder will be 3.

Relational operators

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Table 12.5: List of relational operators

Relational operators	Meaning
<	Less than
< =	Less than or equal to
= =	Equal to
>	Greater than
> =	Greater than or equal to
! =	Not equal

Notes

Notes

Example 1

        int x = 2;

        int l = 1;

        int y = 3;

        int z = 5;

The following statements are true.

(i)   l = = 1

(ii)   x < y

(iii)   z > y

(iv)   y > = 1

(v)   x ! = 1

(vi)   l < = 1

Logical operators

The logical operators are used to combine one or more relational expression. The table 12.6 shows the logical operators.

Table 12.6: List of logical operators

Operators	Meaning
||	OR
&&	AND
!	NOT

The NOT operator is called the unary operator because it requires only one operand.

Example 2

        int x = 5; int z = 9; int y = 7;

        (x > y) & & (z > y)

The first expression (x > y) evaluates to false and second expression (z > y) evaluates to true. Therefore, the final expression is false.

In AND operation, if any one of the expression is false, the entire expression is false.

In OR operation, if any one of the expression is true, the entire expression is true.

In NOT operation, only one expression is required.

If the expression is true, the NOT operation of true is false and vice versa.

Unary operators

C++ provides two unary operators for which only one variable is required.

Example 3

  a = – 50;
  a = –b;
  a = + 50;
  a = + b;

Here plus sign (+) and minus sign (-) are unary because they are not used between two variables.

Assignment operator

The assignment operator '=' stores the value of the expression on the right hand side of the equal sign to the operand on the left hand side.

Example 4

  int m = 5, n = 7;

  int x, y, z;

  x = y = z = 0;

> '=' assignment operator. If you write int a = 5 means it will assign value 5 to an integer variable 'a'.

In addition to standard assignment operator shown above, C++ also supports compound assignment operators. C++ provides two special operators viz '++' and '- -' for incrementing and decrementing the value of a variable by 1. The increment/decrement operator can be used with any type of variable but it cannot be used with any constant. With the prefix version of these operators, C++ performs the increment or decrement operation before using the value of the operand. For instance, the following code:

  int sum, ctr;
  sum = 12;
  ctr = 4;
  sum = sum + (++ctr);

will produce the value of sum as 17 because ctr will be first incremented and then added to sum producing value 17.

Similarly, the following code

  sum = 12;

  ctr = 4;

  sum = sum + (– – ctr);

Notes

Notes

will produce the value of sum as 15 because ctr will be first decremented and then added to sum producing value 15.

With the postfix version of these operators, C++ first uses the value of the operand in evaluating the expression before incrementing or decrementing the operand's value. For example, the following code

>   sum = 12;
>
>   ctr = 4;
>
>   sum = sum + (ctr ++);

will produce the value of sum as 16 because ctr will be first used in the expression producing the value of sum as 16 and then increments the value of ctr by 1 (ctr becomes now 5).

Similary, the following code

>   sum = 12;
>
>   ctr = 4;

sum = sum + (ctr - -) will produce the value of sum as 16 because ctr will be first used with its value 4 producing value of sum as 16 and then decrements the value of ctr by 1 (ctr becomes 3).

Let us study the use of compound assignment operators in the following table:

Table 12.7: Compound assignment operators

Operator	Example	Equivalent to
+ =	A + = 2	A = A + 2
- =	A - = 2	A = A - 2
% =	A % = 2	A = A % 2
/ =	A / = 2	A = A / 2
* =	A * = 2	A = A * 2

Example 5

>   int x = 2; / / first
>
>   x + = 5; / / second

In the second statement, the value of x is 7.

Conditional operator

The conditional operator ?: is called ternary operator as it requires three operands. The format of the conditional operator is: Conditional_expression? expression1 : expression2;

If the value of conditional_expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

Example 6

>   int a = 5;
>   int b = 6;
>   big = (a > b) ? a : b;

The condition evaluates to false, therefore the variable big gets the value from b and it becomes 6.

The comma operator

The comma operator gives left to right evaluation of expressions. It enables to put more than one expression separated by comma on a single line.

Example 7

```
#include <iostream.h>
int main()
{
    int i, j;
    j = 10;
    i = (j++, j+100, 999+j);
    cout << i;
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

>   1010

12.4 THE SIZEOF OPERATOR

As we know that different types of variables, constant, etc., require different amounts of memory to store them. The size of operator can be used to find how many bytes are required for an object to store in memory.

Example 8

>   size of (char) returns 1
>   size of (int) returns 2
>   size of (float) returns 4

If k is an integer variable, the sizeof (k) returns 2.

Notes

The size of operator determines the amount of memory required for an object at compile time rather than at run time.

12.5 THE ORDER OF PRECEDENCE

The order in which the arithmetic operators (+, -, *, / , %) are used in a given expression is called the order of precedence. The following table 12.7 shows the order of precedence.

Table 12.8: Precedence of Arithmetic operators

Order	Opereators
First	( )
Second	*, /, %
Third	+, –

The expression in the following example is calculated from left or right.

Example 9

(20 + 10)* 15 / 5

= 30 * 15 / 5

= 30 * 3

= 90

The table 12.9 shows the precedence of operators.

Table 12.9: Precedence of Operator

++, – – (post increment/decrement)	Highest
+ + (Pre increment – – (Pre decrement), size of ( ), ! (not), – (unary), + (unary)	↑
*, /, %	
+ , _	
<, <=, >, >=	
= = , !, =	
&&	
? :	
=	↓
Comma operator	Lowest

 INTEXT QUESTIONS 12.1

1. Fill in the blanks.

   (a) A keyword is a .... word of C++.

   (b) Pascal is a .... language whereas C++ is an .... language

   (c) An identifier must not start with a ....

   (d) .... constants are whole numbers without any fractional part.

   (e) A sequence of characters enclosed within double quotes is called a .... literal.

2. State whether the following statements are true or false:

   (a) An identifier name can start with the underscore '–'.

   (b) A token is a group of characters that logically belong together.

   (c) Literals are data items that generally change their value during program execution.

   (d) An escape sequence represents a single character.

   (e) The symbol # is used as pre-processor directive.

3. List out the various types of tokens used in C++.

4. List out the data types supported by C++.

12.6 TYPE CONVERSION

The process in which one pre-defined type of expression is converted into another type is called conversion. There are two types of conversion in C++.

- Implicit conversion
- Explicit conversion

12.6.1 Implicit conversion

Data type can be mixed in the expression.

Example 10

```
double a;
int b = 5;
float c = 8.5;
a = b * c;
```

Notes

Notes

When two operands of different types are encountered in the same expression, the lower type variable is converted to the higher type variable. The following table 12.10 shows the order of data types.

<div align="center">

Table 12.10: order of data types

</div>

Order of data types	
Data type	**order**
long double	(highest)
double	
float	↑
long	
int	↓
char	lowest

In the example 10, the int value of b is converted to type float and stored in a temporary variable before being multiplied by the float variable c. The result is then converted to double so that it can be assigned to the double variable a.

12.6.2 Explicit conversion

It is also called type casting. It temporarily changes a variable data type from its declared data type to a new one. It may be noted here that type casting can only be done on the right hand side of the assignment statement.

```
int i1 = 10;
int i2 = 4;
float f = (float)i1 / i2;
```

In the above program, we use a float cast to tell the compiler to promote i1 to a floating point value. Because i1 is a floating point value, i2 will then be promoted to a floating point value as well, and the division will be done using floating point division instead of integer division.

12.7 CONSTANTS

A number which does not change its value during execution of a program is known as a **constant**. Any attempt to change the value of a constant will result in an error message. A constant in C++ can be of any of the basic data types.

Const qualifier can be used to declare constant as shown below:

const float Pi = 3.1415;

The above declaration means that Pi is a constant of float types having a value 3.1415.

Examples of valid constant declarations are:

const int rate = 50;

const float Pi = 3.1415;

const char ch = 'A';

12.8 VARIABLES

A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and is given a symbolic name for easy reference. The variables can be used to hold different values at different times during the execution of a program.

To understand more clearly let us consider the the following statements:

Total = 20.00;   ... (i)

Net = Total - 12.00;   ... (ii)

In statement (i), a value 20.00 has been stored in a memory location Total. The variable Total is used in statement (ii) for the calculation of another variable Net. The point worth noting is that the variable Total is used in statement (ii) by its name not by its value.

Before a variable is used in a program, it has to be defined. This activity enables the compiler to make available the appropriate type of location in the memory. The definition of a variable consists of the type name followed by the name of the variable. For example, a variable Total of type float can be declared as shown below:

float Total;

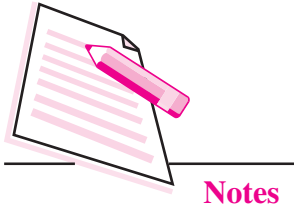Similarly the variable Net can also be defined as shown below:

float Net;

Examples of valid variable declarations are:

(i)   int count;

(ii)   int i, j, k;

(iii)   char ch, first;

(iv)   float total, Net;

(v)   long int sal;

Notes

Notes

12.9 INPUT / OUTPUT (I/O)

C ++ supports input/output statements which can be used to feed new data into the computer or obtain output on an output device such as: VDU, printer etc. It provides both formatted and unformatted stream I/O statements. The following C++ streams can be used for the input/output purpose.

Stream	Description
cin	console input
cout	console output

In addition to the above I/O streams, two operators << and >> are also used. The operator << is known as put to or **bit wise shift operator**. The operator >> is known as **extraction** or **get from operator**.

Let us consider a situation where the user desires to display a message "My first computer" on the screen. This can be achieved through the following statement:

cout << "My first computer";

Once the above statement is carried out by the computer, the message "My first computer" will appear on the screen.

Similarly the following program segment defines a variable sum of integer type, initializes it to value 100 and displays the contents of the variable on the screen.

```
        .
        .
        .
        int sum;
        sum = 100;
        cout << "The value of variable sum =";
        cout << sum;
```

Once the above program segment is executed, the following output is displayed on the screen:

The value of variable sum = 100

From the above discussion we see that cout is the standard output stream of C++ by virtue of which output can be displayed on the screen. However the put to operator << is also required to hand over the contents of the variable to cout as shown in Fig. 12.1.
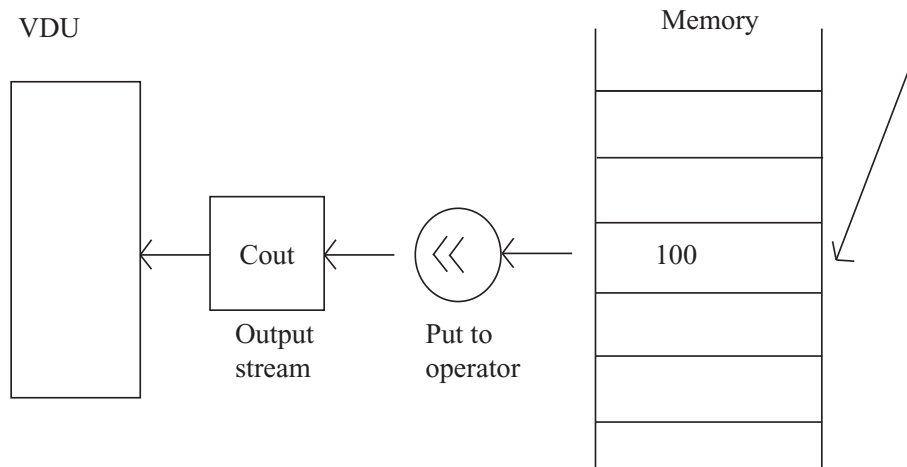
VDU

Memory

Fig: 12.1: Usage of cout and <<

cin is the standard input stream (keyboard) and it can be used to input a value entered by the user from the keyboard. However, the get from operator >> is also required to get the typed value from cin as shown in Fig. 12.2 and store it in the memory location.
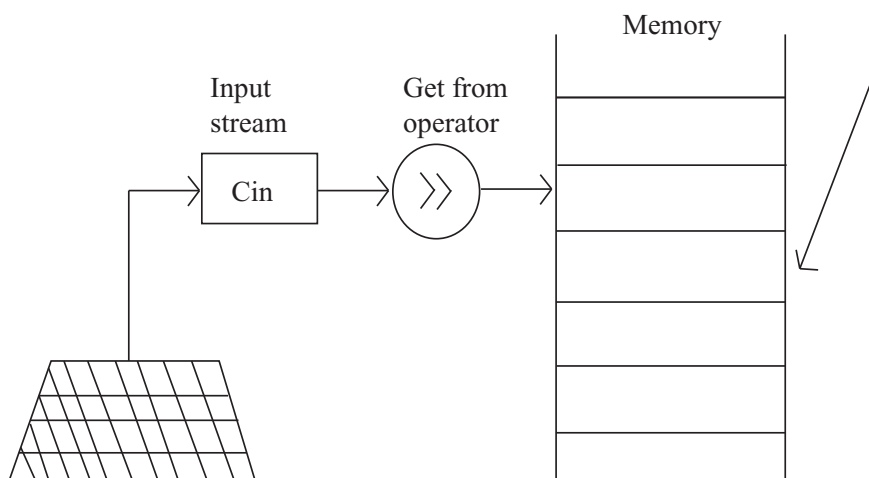
Memory

Fig: 12.2: Usage of cin and >>

Let us consider the following program segment:

```
        .
        .
        .
    int marks;
    cin>> marks;
        .
        .
        .
```

Notes

Notes

In the above segment, the user has defined a variable marks of integer type in the first statement and in the second statement user is trying to read a value from the keyboard. Once this set of statements is obeyed by the computer, whatever is typed on the keyboard (say 87) is received by the cin standard input stream. cin then hands over the typed value to get from operator >> which ultimately stores it in memory location called marks. The contents of this variable can be displayed on the screen by the statements given below:

cout << "marks=";

cout << marks;

We can use more than one output or put to operators within the same output statement as shown below. The activity is known as **cascading of operators**:
cout << "marks =" << marks;   The output of this statement will be

cout << "marks="; << marks;

The output of this statement will be marks = 87.

12.10 STRUCTURE OF C++

The structure of a C++ program is given below:

include <header file>

main ( )

{

....

....

....

}

A C++ program starts with function called main( ). The body of the function is enclosed between curly braces. The program statements are written within the braces. Each statement must end by a semicolon (statement terminator). A C++ program may contain as many functions as required. However, when the program is loaded in the memory, the control is handed over to function main( ) and it is the first function to be executed.

Let us now write our first program:

// This is my first program in C++

/* this program will illustrate different components of a
    simple program in C++*/

include <iostream.h>

void main ( )

{

cout <<"This is my first program in C++";

cout << "\n....................................";

}

When the above program is compiled, linked and executed the following output is displayed on the VDU screen.

This is my first program in C++

....

Various components of this program are discussed below:

(i) Comments

First three lines of the above program are comments and are ignored by the compiler. Comments are included in a program to make it more readable. If a comment is short and can be accommodated in a single line, then it is started with double slash (//)sequence in the first line of the program. However, if there are multiple lines in a comment, it is enclosed between the two symbols /* and */. Everything between /* and */ is ignored by the complier.

(ii) include <iostream.h>

The lines in the above program that start with symbol '#' are called directives and are instructions to the compiler. The word include with '#' tells the compiler to include the file iostream.h into the file of the above program. File iostream.h is a header file needed for input/output requirements of the program. Therefore, this file has been included at the top of the program.

(iii) void main ( )

The word main is a function name. The brackets( ) with main tells that main ( ) is a function. The word void before main( ) indicates that no value is being returned by the function main( ). Every C++ program consists of one or more functions. However, when program is loaded in the memory, the control is handed over to function main( ) and it is the first function to be executed.

Notes

Notes

(iv) The curly brackets and body of the function main( )

A C ++ program starts with function called main( ). The body of the function is enclosed between curly braces. The program statements are written within the brackets. Each statement must end by a semicolon, without which an error message is generated.

Example 11

Write a C ++ program that reads two values x and y, exchanges their contents and prints the output.

Solution: Let us assume x and y are of type int. We will use a variable temp of type int as a temporary location. Let us write a program.

```
# include <iostream.h>
main ( )
{
int x, y temp;
// ... Read the values x and y
cout << "Enter the values:";
cin >> x >> y;
// ... Exchange the contents
temp = x;
x = y;
y = temp;
// display the output now
cout <<"the exchanged values are:" << x << y; }
```

INTEXT QUESTIONS 12.2

1. Fill in the blanks:

   (a) .... operators are used to test the relation between two values.

   (b) C++ provides two .... operators for which only one variable is required.

   (c) The process in which one pre-defined type of expression is converted into another type is called ....

(d) A .... does not understand the contents of a comment and ignores it.

(e) The operator ++ is known as .... operator.

2. State whether the following statements are true or false:

(a) The computer value of an arithmetic expression is always a numerical value.

(b) '\n' is the remainder operator.

(c) Arithmetic operations on characters is allowed in C++.

(d) The output of logical AND operation is true if one of its operand is true.

(e) + = is a compound assignment operator.

3. Evaluate the following expressions.

(a) 5/3 * 6

(b) 6.0 * 5/3

(c) 6 * 5/3

Notes

WHAT YOU HAVE LEARNT

● The built in or basic data types supported by C++ are integer, floating point and character type.

● The identifier is a sequence of characters taken from C++ character set.

● The sizeof operator can be used to find how many bytes are required for an object to store in memory.

● The process in which one pre-defined type of expression is converted into another type is called conversion. There are two types of conversion in C++. i.e., implicit conversion and explicit conversion.

● A number which does not change its value during execution of a program is known as a constant.

● The variables can be used to hold different values at different times during the execution of a program.
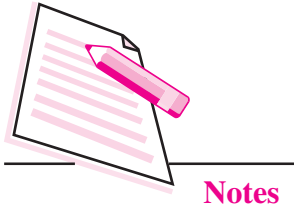
TERMINAL EXERCISE

1. What is the difference between a keyword and an identifier?

2. Explain the following terms:

(i) Literals (ii) Implicit conversion

Notes

3. How many ways are there in C++ to represent an integer constant?

4. State the rules for comment statement.

5. What is an escape sequence? Describe any two backslash character constant?

6. Explain briefly about standard input/output stream of C++.

7. Write the equivalent C++ expression for the following algebraic expression:

   (i) $\dfrac{2AB + 2BC + 2CA}{2A}$

   (ii) $\dfrac{4}{3}x^2$

   (iii) $\dfrac{b^2 - 4ac}{2a}$

8. Evaluate the following C++ expression:

   int a, b = 2, k = 4;

   a = b * 3/4 + k/4 + 8 – b + 5/8;

9. Write an appropriate C++ statement for each of the following:

   (i)   Read the values of a, b and c.

   (ii) Write the values of a and b in one line followed by the value of c on another line.

   (iii) Write the values of a and b in one line separated by blanks and value of c after two blank lines.

10. Write a program that will find out the square of a given number.

ANSWERS TO INTEXT QUESTIONS

12.1

1. (a) reserved   (b) structured, object-oriented

    (c) digit   (d) Integer

    (e) string

2. (a) True   (b) True   (c) False   (d) True   (e) True

3.   Various types of tokens used in C++ are:

(i) Keywords   (ii) Identifiers

(iii) Literals   (iv) Punctuators

(v) Operators

4.   Data types are:

(i)   Integer type (int)

(ii) Floating point type (float)

(iii) Character type (char)

Notes

12.2

1.   (a)   Relational

(b)   Unary

(c)   Conversion

(d)   Compiler

 (e)   Increment

2.   (a)   True   (b) False   (c)   True

(d)   False   (e) True

3.   (a)   6   (b) 10.0   (c)   10