CHAPTER 9

Introduction to C++

F9DFGC

Learning Objectives

After the completion of this chapter, the student will be able to

- Understand the basic building blocks of C++ programming language
- Able to construct simple C++ programs
- Execute and debug C++programs

9.1 Introduction

C++ is one of the most popular programming language which supports both procedural and Object Oriented Programming paradigms. Thus, C++ is called as a **hybrid language.** C++ is a superset (extension) of its predecessor C language. Bjarne Stroustrup named his new language as "C with Classes". The name C++ was coined by Rick Mascitti where ++ is the C language increment operator.

Bjarne Stroustrup
Inventor of C++ Programming Language

Bjarne is a Danish Computer Scientist born on 30th December 1950. He has a Master degree in Mathematics and Computer Science in 1975 from Aarhus University, Denmark and Ph.D in Computer Science in 1979 from the University of Cambridge, England.

History of C++

C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratory during 1979. C++ is originally derived from C language and influenced by many languages like Simula, BCPL, Ada, ML, CLU and ALGOL 68. Till 1983, it was referred "New C" and "C with Classes". In 1983, the name was changed as C++ by Rick Mascitti.

Benefits of learning C++

- C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development.
- C++ is an object-oriented programming language and includes **classes**, **inheritance**, **polymorphism**, **data abstraction** and **encapsulation**.
- C++ has a rich function library.
- C++ allows exception handling, inheritance and function overloading which are not possible in C.
- C++ is a powerful, efficient and fast language. It finds a wide range of applications – from GUI applications to 3D graphics for games to real-time mathematical simulations.

9.2 Character set

Character set is a set of characters which are used to write a C++ program. A character represents any alphabet, number or any other symbol (special characters) mostly available in the keyboard. C++ accepts the following characters.

Alphabets	A …. Z, a ….. z
Numeric	0 …. 9
Special Characters	+ - * / ~ ! @ # $ % ^& [ ] ( ) { } = >< _ \ | ? . , : ‘ “ ;
White space	Blank space, Horizontal tab (→), Carriage return (↲), Newline, Form feed
Other characters	C++ can process any of the 256 ASCII characters as data.

9.3 Lexical Units (Tokens):

C++ program statements are constructed by many different small elements such as commands, variables, constants and many more symbols called as operators and punctuators. These individual elements are collectively called as **Lexical units** or **Lexical elements** or **Tokens**. C++ has the following tokens:

- Keywords
- Literals
- Punctuators
- Identifiers
- Operators

TOKEN:

The smallest individual unit in a program is known as a Token or a Lexical unit

9.3.1 Keywords

Keywords are the reserved words which convey specific meaning to the C++ compiler. They are the essential elements to construct C++ programs. Most of the keywords are common to C, C++ and Java.

C++ is a case sensitive programming language so, all the keywords must be in lowercase.

Table 9.1 C++ Keywords

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

- With revisions and additions, the recent list of keywords also includes:

using, namespace, bal, static_cast, const_cast, dynamic_cast, true, false

- Identifiers containing a double underscore are reserved for use by C++ implementations and standard libraries and should be avoided by users.

9.3.2 Identifiers

Identifiers are the user-defined names given to different parts of the C++ program viz. variables, functions, arrays, classes etc., These are the fundamental building blocks of a program. Every language has specific rules for naming the identifiers.

Rules for naming an identifier:

- The first character of an identifier must be an alphabet or an underscore (_).
- Only alphabets, digits and underscore are permitted. Other special characters are not allowed as part of an identifier.
- C++ is case sensitive as it treats upper and lower-case characters differently.
- Reserved words or keywords cannot be used as an identifier name.

As per ANSI standards, C++ places no limit on its length and therefore all the characters are significant.

Identifiers	Valid / Invalid	Reason for invalid
Num	Valid	
NUM	Valid	
_add	Valid	
total_sales	Valid	
tamilMark	Valid	
num-add	Invalid	Contains special character (-)
this	Invalid	This is one of the keyword. Keyword cannot be used as identifier names.
2myfile	Invalid	Name must start begins with an alphabet or an underscore

- You may use an underscore in variable names to separate different parts of the name (eg: ***total_sales*** is a valid identifier where as the variable called ***total sales*** is an invalid identifier).
- You may use capital style notation, such as ***tamilMark*** ie. capitalizing the first letter of the second word.

9.3.3 Literals (Constants)

Literals are data items whose values do not change during the execution of a program. Therefore Literals are called as Constants. C++ has several kinds of literals:

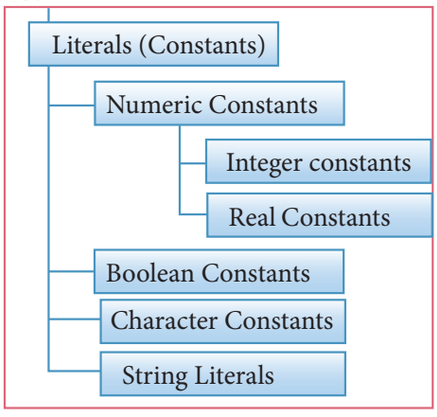


Figure 9.1 Types of Constants

Numeric Constants:

As the name indicates, the numeric constants are numeric values, which are used as constants. Numeric constants are further classified as:

1. Integer Constants (or) Fixed point constants.
2. Real constants (or) Floating point constants.

(1) Integer Constants (or) Fixed point constants

Integers are whole numbers without any fractions. An integer constant must have at least one digit without a decimal point. It may be signed or unsigned. Signed integers are considered as negative, commas and blank spaces are not allowed as part of it. In C++, there are three types of integer constants: (i) Decimal (ii) Octal (iii) Hexadecimal

(i) Decimal

Any sequence of one or more digits (0 …. 9)

Valid	Invalid
725	7,500 (Comma is not allowed)
-27	66 5(Blank space is not allowed)
4.56	9$ (Special Character not allowed)

If you assign **4.56** as an integer decimal constant, the compiler will accept only the integer portion of **4.56** ie. **4**. It will simply ignore **.56.**

Notes

If a Decimal constant declared with fractions, then the compiler will take only the integer part of the value and it will ignore its fractional part. This is called as "Implicit Conversion". It will be discussed later.

(ii) Octal

Any sequence of one or more octal values (0 …. 7) that begins with 0 is considered as an Octal constant.

Valid	Invalid
012	05,600(Commas is not allowed)
-027	04.56 (Decimal point is not allowed)**
+0231	0158 (8 is not a permissible digit in octal system)

Notes

** When you use a fractional number that begins with 0, C++ considers the number as an integer not an Octal.

(iii) Hexadecimal

Any sequence of one or more Hexadecimal values (0 …. 9, A …. F) that starts with 0x or 0Xis considered as an Hexadecimal constant.

Valid	Invalid
0x123	0x1,A5 (Commas is not allowed)
0X568	0x.14E (Decimal point is not allowed like this)

The suffix L or l and U or u added with any constant forces it to be represented as a long or unsigned constant respectively.

(2) Real Constants (or) Floating point constants

A real or floating point constant is a numeric constant having a fractional component. These constants may be written in fractional form or in exponent form.

Fractional form of a real constant is a signed or unsigned sequence of digits including a decimal point between the digits. It must have at least one digit before and after a decimal point. It may be prefixed with + or - sign. A real constant without any sign will be considered as positive.

Exponent form of real constants consists of two parts: **(1) Mantissa** and **(2) Exponent.** The mantissa must be either an integer or a real constant. The mantissa followed by a letter E or e and the exponent, should also be an integer.

For example, 58000000.00 may be written as 0.58×10^8 or 0.58E8.

Mantissa (Before E)	Exponent (After E)
0.58	8

Example:

5.864 E1 $10^1 \times 5.864$ ➔ 58.64 ➔

5864 E-2 $10^{-2} \times 5864$ ➔ 58.64 ➔

0.5864 E2 $10^2 \times 0.5864$ ➔ 58.64 ➔

Boolean Literals

Boolean literals are used to represent one of the Boolean values (True or false). Internally true has value 1 and false has value 0.

Character constant

A character constant in C++ is any valid single character enclosed within single quotes.

Valid character constants : 'A', '2', '$'

Invalid character constants : "A"

The value of a single character constant has an equivalent ASCII value. For example, the value of 'A' is 65.

Escape sequences (or) Non-graphic characters

C++ allows certain non-printable characters represented as character constants. Non-printable characters are also called as non-graphic characters. Non-printable characters are those characters that cannot be typed directly from a keyboard during the execution of a program in C++, for example: backspace, tabs etc. These non-printable characters can be represented by using escape sequences. An escape sequence is represented by a backslash followed by one or two characters.

Table 9.2 Escape Sequences

Escape sequence	Non-graphical character
\a	Audible or alert bell
\b	Backspace
\f	Form feed
\n	Newline or linefeed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quote
\"	Double quote
\?	Question Mark
\On	Octal number
\xHn	Hexadecimal number
\0	Null

Even though an escape sequence contains two characters, they should be enclosed within single quotes because, C++ consider escape sequences as character constants and allocates one byte in ASCII representation.

ASCII (American Standard Code for Information Interchange) was first developed and published in 1963 by the X3 committee, a part of the American Standards Association (ASA).

Evaluate Yourself

1. What is meant by literals? How many types of integer literals are available in C++?
2. What kind of constants are following?
   i) 26 ii) 015 iii) 0xF iv) 014.9
3. What is character constant in C++?
4. How are non graphic characters represented in C++?
5. Write the following real constants into exponent form:
   i) 32.179 ii) 8.124 iii) 0.00007
6. Write the following real constants in fractional form:
   i) 0.23E4 ii) 0.517E-3 iii) 0.5E-5
7. What is the significance of null (\0) character in a string?

String Literals

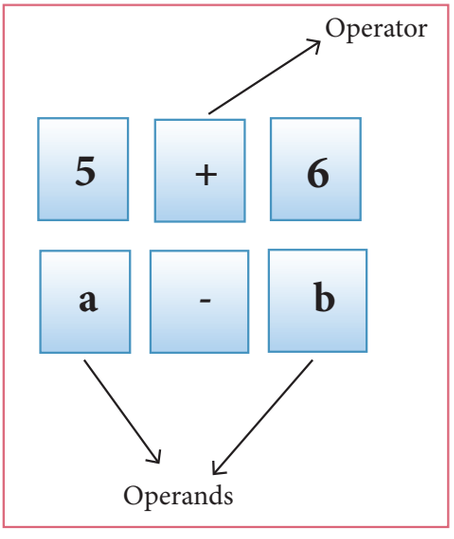
Sequence of characters enclosed within double quotes are called as String literals. By default, string literals are automatically added with a special character **'\0' (Null)** at the end. Therefore, the string "welcome" will actually be represented as "welcome\0" in memory and the size of this string is not 7 but 8 characters i.e., inclusive of the last character \0.

Valid string Literals : "A", "Welcome" "1234"

Invalid String Literals : 'Welcome', '1234'

9.3.4 Operators

The symbols which are used to do some mathematical or logical operations are called as **"Operators".** The data items or values that the oper ators act upon are called as **"Operands".**



In C++, The operators are classified on the basis of the number of operands.

(i) **Unary Operators** - Require only one operand

(ii) **Binary Operators** - Require two operands

(iii) **Ternary Operators** - Require three operands

C++ Binary Operators are classified as:

(1) Arithmetic Operators
(2) Relational Operators
(3) Logical Operators
(4) Assignment Operators
(5) Conditional Operator

(1) Arithmetic Operators

Arithmetic operators perform simple arithmetic operations like addition, subtraction, multiplication, division etc.,

Operator	Operation	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 – 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2 (Quotient of the division)
%	Modulus (To find the remainder of a division)	10 % 3 = 1(Remainder of the division)

- The above mentioned arithmetic operators are binary operators which requires minimum of two operands.

Increment and Decrement Operators

++ (Plus, Plus) Increment operator

-- (Minus, Minus) Decrement operator

An increment or decrement operator acts upon a single operand and returns a new value. Thus, these operators are unary operators. The increment operator adds 1 to its operand and the decrement operator subtracts 1 from its operand. For example,

- **x++ or ++ x** is the same as **x = x+1;**

  It adds 1 to the present value of x
- **x -- or -- x** is the same as to **x = x–1**;

  It subtracts 1 from the present value of x

The ++ or -- operators can be placed either as prefix (before) or as postfix (after) to a variable. With the prefix version, C++ performs the increment / decrement before using the operand.

(2) Relational Operators

Relational operators are used to determine the relationship between its operands. When the relational operators are applied on two operands, the result will be a Boolean value i.e 1 or 0 to represents True or False respectively. C++ provides six relational operators. They are,

Operator	Operation	Example
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b
==	Equal to	a == b
!=	Not equal	a != b

- In the above examples, the operand *a* is compared with *b* and depending on the relation, the result will be either 1 or 0. i.e., 1 for true, 0 for false.
- All six relational operators are binary operators.

(3)Logical Operators

A logical operator is used to evaluate logical and relational expressions. The logical operators act upon the operands that are themselves called as logical expressions. C++ provides three logical operators.

Table 9.3 Logical Operators

Operator	Operation	Description
&&	AND	The logical AND combines two different relational expressions in to one. It returns 1 (True), if both expression are true, otherwise it returns 0 (false).

||	**OR**	The logical OR combines two different relational expressions in to one. It returns 1 (True), if either one of the expression is true. It returns 0 (false), if both the expressions are false.
!	**NOT**	NOT works on a single expression / operand. It simply negates or inverts the truth value. i.e., if an operand / expression is 1 (true) then this operator returns 0 (false) and vice versa

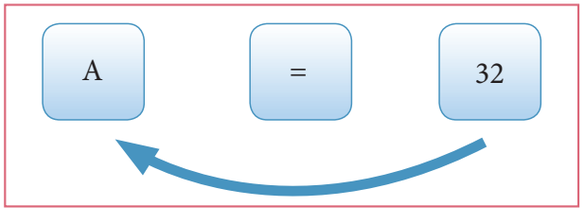
- AND, OR both are binary operators where as NOT is an unary operator.

Example: a = 5, b = 6, c = 7;

Expression	Result
(a<b) && (b<c)	1 (True)
(a>b) && (b<c)	0 (False)
(a<b) || (b>c)	1 (True)
!(a>b)	1 (True)

(4)Assignment Operator:

Assignment operator is used to assign a value to a variable which is on the left hand side of an assignment statement. = (equal to) is commonly used as the assignment operator in all computer programming languages. This operator copies the value at the right side of the operator to the left side variable. It is also a binary operator.



C++ uses different types of assignment operators. They are called as Shorthand assignment operators.

Operator	Name of Operator	Example
+=	Addition Assignment	a = 10; c = a += 5; (ie, a = a + 5) c = 15
-=	Subtraction Assignment	a = 10; c = a -= 5; (ie. a = a – 5) c = 5
*=	Multiplication Assignment	a = 10; c = a *= 5; (ie. a = a * 5) c = 50
/=	Division Assignment	a = 10; c = a /= 5; (ie. a = a / 5) c = 2
%=	Modulus Assignment	a = 10; c = a %= 5; (ie. a = a % 5) c = 0

Discuss the differences between = and == operators

(5) Conditional Operator:

In C++, there is only one conditional operator. **?:** is a conditional Operator which is also known as Ternary Operator. This operator is used as an alternate to if … else control statement. We will learn more about this operator in later chapters along with if …. else structure.

Other Operators:

The Comma operator	Comma (,) is an operator in C++ used to string together several expressions. The group of expression separated by comma is evaluated from left to right.
Sizeof	This is called as compile time operator. It returns the size of a variable in bytes.
Pointer	* Pointer to a variable & Address of

Component selection	. Direct component selector -> Indirect component selector
Class member operators	:: Scope access / resolution .* Dereference ->* Dereference pointer to class member

Precedence of Operators:

Operators are executed in the order of precedence. The operands and the operators are grouped in a specific logical way for evaluation. This logical grouping is called as an Association.

The order of precedence:

( ) [ ]	Operators within parenthesis are performed first	Higher
++, --	Postfix increment / decrement	
++, --	Prefix increment / decrement	
*, /, %	Multiplication, Division, Modulus	
+, -	Addition, Subtraction	
<, <=, >, >=	Less than, Less than or equal to, Greater than, Greater than or equal to	
==, !=	Equal to, Not equal to	
&&	Logical AND	
||	Logical OR	
?:	Conditional Operator	
=	Simple Assignment	
+=, -=, *=, /=	Shorthand operators	
,	Comma operator	Lower

9.3.5 Punctuators

Punctuators are symbols, which are used as delimiters, while constructing a C++ program. They are also called as **"Separators".** The following punctuators are used in C++; most of these symbols are very similar to C and Java.

Separator	Description	Example
Curly braces { }	Opening and closing curly braces indicate the start and end of a block of code. A block of code containing more than one executable statement. These statements together are called as "compound statement"	int main ( ) { int x=10, y=20, sum; sum = x + y; cout << sum; }
Parenthesis ( )	Opening and closing parenthesis indicate function calls and function parameters.	clrscr( ); int main ( )
Square brackets [ ]	It indicates single and multidimensional arrays.	int num[5]; char name[50];
Comma ,	It is used as a separator in an expression	int x=10, y=20, sum;

Semicolon ;	Every executable statement in C++ should terminate with a semicolon	int main ( ) { int x=10, y=20, sum; sum = x + y; cout << sum; }
Colon :	It is used to label a statement.	private:
Comments // /* */	Any statement that begins with // are considered as comments. Comments are simply ignored by compilers. i.e., compiler does not execute any statement that begins with a // **// Single line comment** **/* …….. */ Multiline comment**	**/* This is written by me to learn CPP */** int main ( ) { int x=10, y=20, sum; **// to sum x and y** sum = x + y; cout << sum; }

In C++, one or two operators may be used in different places with different meaning.

For example: Asterisk (*) is used for multiplication as well as for pointer to a variable.

Evaluate Yourself

1. What is the use of operators?
2. What are binary operators? Give examples of arithmetic binary operators.
3. What does the modulus operator % do?
4. What will be the result of 8.5 % 2?
5. Give that i = 8, j = 10, k = 8, What will be result of the following expressions?
   (i) i < k (ii) i < j (iii) i > = k (iv) i = = j (v) j ! = k
6. What will be the order of evaluation for the following expressions?
   (i) i + 3 >= j - 9 (ii) a +10 < p - 3 + 2 q
7. Write an expression involving a logical operator to test, if marks are 75 and grade is 'A'.

9.4 I/O Operators

9.4.1 Input operator

C++ provides the operator >> to get input. It extracts the value through the keyboard and assigns it to the variable on its right; hence, it is called as **"Stream extraction"** or **"get from"** operator.

It is a binary operator i.e., it requires two operands. The first operand is the pre-defined identifier cin (pronounced as C-In) that identifies keyboard as the input device. The second operand must be a variable.

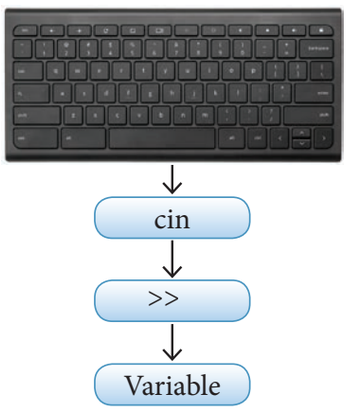


Figure 9.4 Working process of cin

To receive or extract more than one value at a time, >> operator should be used for each variable. This is called cascading of operator.

Example:

cin >> num ;	Pre-defined object cin extracts a value typed on keyboard and stores it in variable **num.**

cin >>x >> y;	This is used to extract two values. cin reads the first value and immediately assigns that to variable x; next, it reads the second value which is typed after a space and assigns that to y. Space is used as a separator for each input.

9.4.2 Output Operator

C++ provides << operator to perform output operation. The operator << is called the **"Stream insertion"** or **"put to"** operator. It is used to send the strings or values of the variables on its right to the object on its left. << is a binary operator.

The first operand is the pre-defined identifier cout (pronounced as C-Out) that identifies monitor as the standard output object. The second operand may be a constant, variable or an expression.

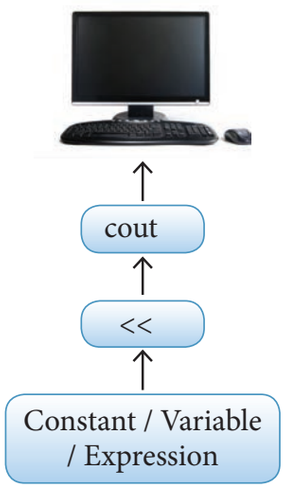


Figure 9.5 Working process of cout

To send more than one value at a time, << operator should be used for each constant/variable/expression. This is called **cascading of operator**.

Example:

cout << "Welcome";	Pre-defined object cout sends the given string "Welcome" to screen.
cout << "The sum = " << sum;	First, cout sends the string "The Sum = " to the screen and then sends the value of the variable sum; Usually, cout sends everything specified within double quotes or single quotes i.e., string or character constants, except non-graphic characters.
cout <<"\n The Area: " <<3.14*r*r;	First, cout sends everything specified within double quotes except \n to the screen, and then it evaluates the expression 3.14*r*r and sends the result to the screen. \n – is a non graphic character constant to feed a new line.
cout << a + b ;	cout sends the sum of a and b to the output console (monitor)

9.4.3. Cascading of I/O operators

The multiple use of input and output operators such as >> and << in a single statement is known as cascading of I/O operators.

Cascading cout:

```
int Num=20;
cout << "A=" << Num;
```

The Figure 9.6 is used to understand the working of Cascading cout statement

```
cout << "A=" << Num;
```

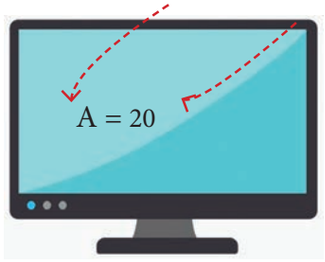


Figure 9.6 Cascading cout

Cascading cin - Example:

cout >> "Enter two number: ";
cin >> a >> b;

The Figure 9.7 is used to understand the working of Cascading cin statement

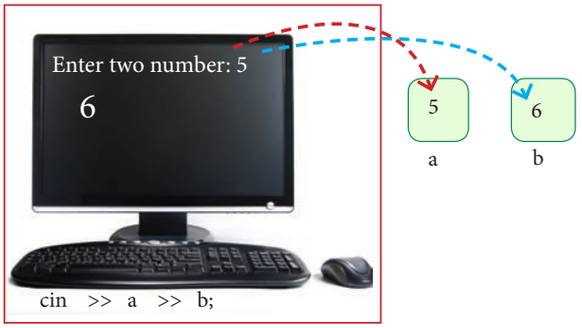


Figure 9.7 Cascading cin

9.5 Sample program – A first look at C++ program

Let us start our first C++ program that prints a string "Welcome to Programming in C++" on the screen.

Sample Prog 9_5.cpp

```
1  // C++ program to print a string
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6      cout << "Welcome to Programming in C++";
7      return 0;
8  }
```

The above program produces, the following output:

Welcome to Programming in C++

This is very simple C++ program which includes the basic elements that every C++ program has. Let us have a look at these elements:

1	// C++ program to print a string
This is a comment statement. Any statement that begins with // are considered as comments. Compiler does not execute any comment as part of the program and it simply ignores. If we need to write multiple lines of comments, we can use /* ……. */.	
2	**# include <iostream>**
Usually all C++ programs begin with include statements starting with a # (hash / pound). The symbol # is a directive for the preprocessor. That means, these statements are processed before the compilation process begins. **#include <iostream> statement tells the compiler's preprocessor to include the header file "iostream" in the program.** The header file iostream should included in every C++ program to implement input / output functionalities. In simple words, **iostream header file contains the definition of its member objects cin and cout.** If you fail to include iostream in your program, an error message will occur on cin and cout; and we will not be able to get any input or send any output.	
3	**using namespace std;**
The line using namespace std; tells the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. Namespaces provide a method of preventing name conflicts in large projects. It is a new concept introduced by the ANSI C++ standards committee.	

4 int main ( )

C++ program is a collection of functions. Every C++ program must have a main function. The main( ) function is the starting point where all C++ programs begin their execution. Therefore, the executable statements should be inside the main( ) function.

```
5  {
6      cout << "Welcome to Programming in C++";
7      return 0;
8  }
```

The statements between the curly braces (Line number 5 to 8) are executable statements. This is actually called as a block of code. In line 6, cout simply sends the string constant "Welcome to Programming in C++" to the screen. As we discussed already, every executable statement must terminate with a semicolon. In line 7, return is a keyword which is used to return the value what you specified to a function. In this case, it will return 0 to main( ) function.

9.6 Execution of C++ program:

For creating and executing a C++ program, one must follow four important steps.

(1) Creating Source code

Creating includes typing and editing the valid C++ code as per the rules followed by the C++ Compiler.

(2) Saving source code with extension .cpp

After typing, the source code should be saved with the extension .cpp

(3) Compilation

This is an important step in constructing a program. In compilation, compiler links the library files with the source code and verifies each and every line of code. If any mistake or error is found, it will throw error message. If there are no errors, it translates the source code into machine readable object file with an extension .obj

(4) Execution

This is the final step of a C++ Program. In this stage, the object file becomes an executable file with extension .exe. Once the program becomes an executable file, the program has an independent existence. This means, you can run your application without the help of any compiler or IDE.

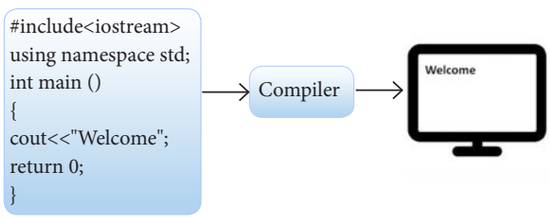


Figure 9.8 Execution

9.7 C++ Development Environment

There are lot of IDE programs available for C++. IDE makes it easy to create, compile and execute a C++ program. Most of the IDEs are open source applications (ie.) they are available free of cost.

9.7.1 Familiar C++ Compilers with IDE

Table 9.4 Open Source Compilers

Compiler	Availability
Dev C++	Open source
Geany	Open source
Code::blocks	Open source
Code Lite	Open source

Net Beans	Open source
Digital Mars	Open source
Sky IDE	Open source
Eclipse	Open source

9.7.2 Working with Dev C++

Among the dozens of IDEs, we take "Dev C++" compiler to create C++ programs. Programming techniques and illustrated programs of this book are based on "Dev C++" compiler.

Dev C++ is an open source, cross platform (alpha version available for Linux), full featured Integrated Development Environment (IDE) distributed with the GNU General Public License for programming in C and C++. It is written in Delphi. It can be downloaded from ***http://www.bloodshed.net/dev/devcpp.html***

1. After installation **Dev C++** icon is available on the desktop. Double click to open IDE. Dev C++ IDE appears as given below.

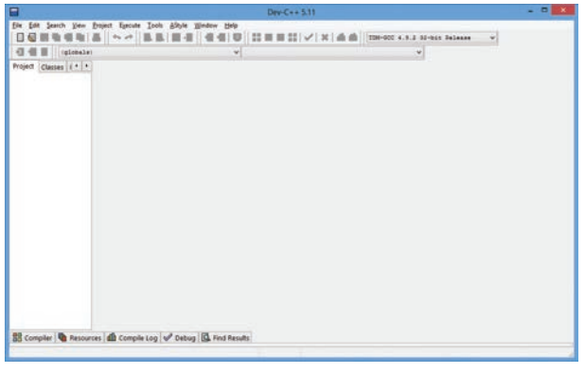


Figure 9.9 Dev C++ opening Window

2. To create a source file, Select **File → New → Source file** or **Press Ctrl + N**.
3. In the screen that appears, type your C++ program, and save the file by clicking **File → Save** or Pressing **Ctrl + S.** It will add .cpp by default at the end of your source code file. No need to type .cpp along with your file name.

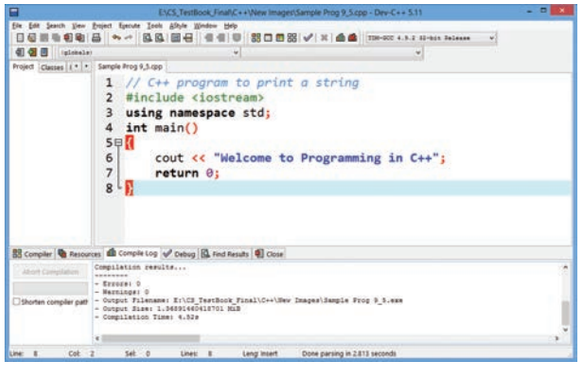


Figure 9.10 Dev C++ IDE with a program

4. **After save, Click Execute → Compile and Run or press F11 key.**

   If your program contains any error, it displays the errors under **compile log**. If your program is without any error, the display will appear as follows.

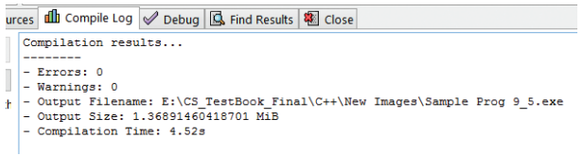


Figure 9.11 Dev C++ Compile Log

5. **After successful compilation, output will appear in output console, as follows**

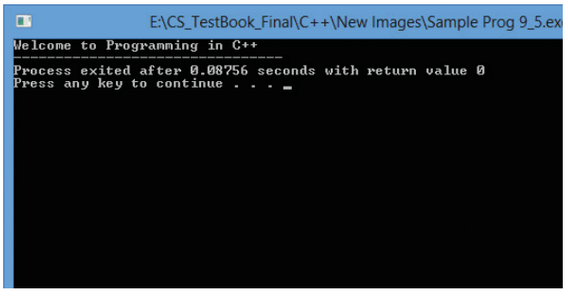


Figure 9.12 Dev C++ Output Window

9.8 Types of Errors

Some common types of errors are given below:

Type of Error	Description
Syntax Error	• Syntax is a set of grammatical rules to construct a program. Every programming language has unique rules for constructing the sourcecode. • Syntax errors occur when grammatical rules of C++ are violated. • Example: if you type as follows, C++ will throw an error. **cout << "Welcome to Programming in C++"** • As per grammatical rules of C++, every executable statement should terminate with a semicolon. But, this statement does not end with a semicolon.
Semantic Error	• A Program has not produced expected result even though the program is grammatically correct.It may be happened by wrong use of variable / operator / order of execution etc. This means, program is grammatically correct, but it contains some logical error. So, Semantic error is also called as **"Logic Error"**.
Run-time error	• A run time error occurs during the execution of a program. It occurs because of some illegal operation that takes place. • For example, if a program tries to open a file which does not exist, it results in a run-time error

Points to Remember:

- C++ was developed by Bjarne Stroustrup at AT & T Bell Labs during the year 1979.
- Character set is the set of characters which are allowed to write C++ programs.
- Individual elements are collectively called as Lexical units or Lexical elements or Tokens.
- Keywords are the reserved words that convey specific meaning to the C++ compiler.
- Identifiers are user-defined names given to different parts of the C++ program viz. variables, functions, arrays, classes etc.,
- Literals are data items whose values do not change during the execution of a program. Therefore, Literals are called as Constants.
- There are different kinds of literals used in C++ (Integer, Float, Character, String)
- The symbols which are used to do some mathematical, logical operations are called as "Operators".
- Punctuators are symbols, which are used as delimiters in constructing C++ programs. They are also called as "Separators".
- Extraction operator(>>) and Insertion operator (<<) are used to get input and send output in C++.

Hands on practice:

- Type the following C++ Programs in Dev C++ IDE and execute. if compiler shows any errors, try to rectify it and execute again and again till you get the expected result.

1. C++ Program to find the total marks of three subjects

```
#include <iostream>
using namespace std;
int main()
{
        int m1, m2, m3, sum;
        cout << "\n Enter Mark 1: ";
        cin >> m1;
        cout << "\n Enter Mark 2: ";
        cin >> m2;
        cout << "\n Enter Mark 3: ";
        cin >> m3;
        sum = m1 + m2 + m3;
        cout << "\n The sum = " << sum;
}
```

- Make changes in the above code to get the average of all the given marks.

2. C++ program to find the area of a circle

```
#include <iostream>
using namespace std;
int main()
{
        int radius;
        float area;
        cout << "\n Enter Radius: ";
        cin >> radius;
        area = 3.14 * radius * radius;
        cout << "\n The area of circle = " << area;
}
```

3. point out the errors in the following program:

```
Using namespace std;
int main( )
{
cout << “Enter a value ”;
cin << num1 >> num2
num+num2=sum;
cout >> “\n The Sum= ” >> sum;
```

4. point out the type of error in the following program:

```
#include <iostream>
using namespace std;
int main()
{
        int h=10; w=12;
        cout << "Area of rectangle " << h+w;
}
```

Evaluation

F9MBHZ

SECTION – A

Choose the correct answer:

1. Who developed C++?
   (a) Charles Babbage (b) Bjarne Stroustrup
   (c) Bill Gates (d) Sundar Pichai
2. What was the original name given to C++?
   (a) CPP (b) Advanced C (c) C with Classes (d) Class with C
3. Who coined C++?
   (a) Rick Mascitti (b) Rick Bjarne (c) Bill Gates (d) Dennis Ritchie
4. The smallest individual unit in a program is:
   (a) Program (b) Algorithm (c) Flowchart (d) Tokens
5. Which of the following operator is extraction operator in C++?
   (a) >> (b) << (c) <> (d) ^^
6. Which of the following statements is not true?
   (a) Keywords are the reserved words which convey specific meaning to the C++ compiler.
   (b) Reserved words or keywords can be used as an identifier name.
   (c) An integer constant must have at least one digit without a decimal point.
   (d) Exponent form of real constants consist of two parts
7. Which of the following is a valid string literal?
   (a) 'A' (b) 'Welcome' (c) 1232 (d) "1232"
8. A program written in high level language is called as
   (a) Object code (b) Source code (c) Executable code (d) All the above
9. Assume a=5, b=6; what will be result of a&b?
   (a) 4 (b) 5 (c) 1 (d) 0
10. Which of the following is called as compile time operators?
    (a) sizeof (b) pointer (c) virtual (d) this

SECTION-B

Very Short Answers

1. What is meant by a token? Name the token available in C++.
2. What are keywords? Can keywords be used as identifiers?
3. The following constants are of which type?
   (i) 39 (ii) 032 (iii) 0XCAFE (iv) 04.14
4. Write the following real constants into the exponent form:
   (i) 23.197 (ii) 7.214 (iii) 0.00005 (iv) 0.319
5. Assume n=10; what will be result of n++ and --n;?
6. Match the following

A	B
(a) Modulus	(1) Tokens
(b) Separators	(2) Remainder of a division
(c) Stream extraction	(3) Punctuators
(d) Lexical Units	(4) get from

SECTION-C

Short Answers

1. Describe the differences between keywords and identifiers?
2. Is C++ case sensitive? What is meant by the term "case sensitive"?
3. Differentiate "=" and "==".
4. What is the use of a header file?
5. Why is main function special?

SECTION - D

Explain in detail

1. Write about Binary operators used in C++.
2. What are the types of Errors?

References:

(1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
(2) The Complete Reference C++ (Forth Edition), Herbert Schildt.Mc.Graw Hills.
(3) Computer Science with C++ (A text book of CBSE XI and XII), Sumita Arora, Dhanpat Rai & Co.

Data Types, Variables and Expressions

9.10 Introduction

Every programming language has two fundamental elements, viz., data types and variables. They are very essential elements to write even the most elementary programs. C++ provides a predefined set of data types for handling the data items. Such data types are known as fundamental or built-in data types. Apart from the built-in data types, a programmer can also create his own data types called as User-defined data types. In this chapter, we are going to learn about built-in data types.

9.11 Concept of Data types

Let us look at the following example,

Name = Ram

Age = 15

Average_Mark = 85.6

In the above example, Name, Age, Average_mark are the fields which hold the values such as Ram, 15, and 85.6 respectively.

In a programming language, **fields** are referred as **variables** and the **values** are referred to as **data**. Each data item in the above example looks different. That is, "Ram" is a sequence of alphabets and the other two data items are numbers. The first value is a whole number and the second one is a fractional number. In real-world scenarios, there are lots of different kinds of data we handle in our day-to-day life. The nature or type of the data item varies, for example distance (from your home to school), ticket fare, cost of a pen, marks, temperature, etc.,

In C++ programming, before handling any data, it should be clearly specified to the language compiler, regarding what kind of data it is, with some predefined set of data types.

9.12 C++ Data types

In C++, the data types are classified as three main categories

(1) Fundamental data types

(2) User-defined data types and

(3) Derived data types.

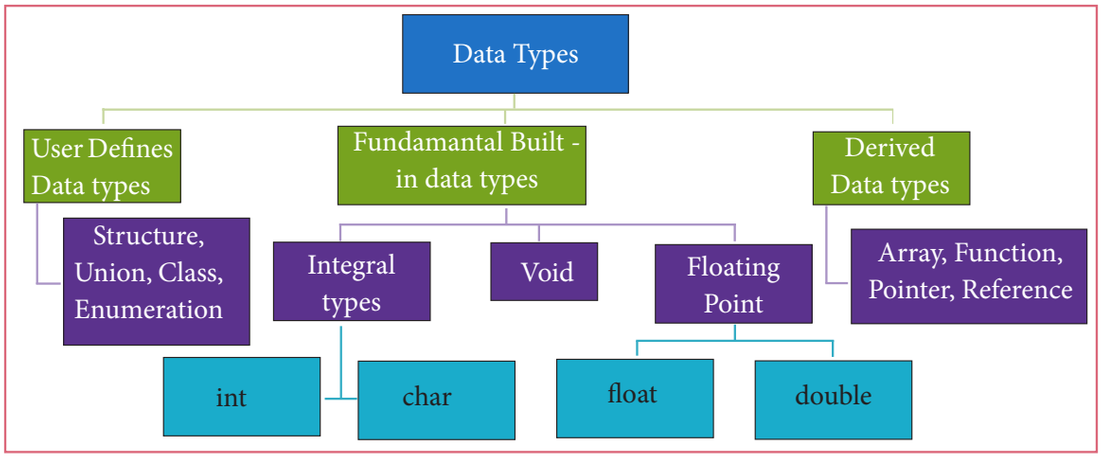


Figure 9.13 Data types in C++

In this chapter, we are going to learn about only the Fundamental data types.

In order to understand the working of data types, we need to know about variables. **T**he variables are the named memory locations to hold values of specific data types. In C++, the variables should be declared explicitly with their data types before they are actually used.

Syntax for declaring a variable:

<data type> <variable name>;

Example:

int num1;

To declare more than one variable which are of the same data type using a single statement, it can be declared by separating the variables using a comma.

Example:

int num1, num2, sum;

For example, to store your computer science marks first you should declare a variable to hold your marks with a suitable data type. Choosing an appropriate data type needs more knowledge and experience. Usually, marks are represented as whole numbers. Thus, the variable for storing the computer science marks should be of integer data type.

Example:

int comp_science_marks;

Now, one variable named **comp_science_marks** is ready to store your marks.

We will learn more about variables later in this chapter.

9.12.1 Introduction to fundamental Data types:

Fundamental (atomic) data types are predefined data types available with C++. There are five fundamental data types in C++: **char, int, float, double** and **void**. Actually, these are the keywords for defining the data types.

(1) int data type:

Integer data type accepts and returns only integer numbers. If a variable is declared as an **int**, C++ compiler allows storing only integer values into it. If you try to store a fractional value in an int type variable it will accept only the integer portion and the fractional part will be ignored.

For Example
int num=12;

num1 variable is declared as integer types. So, it can store integer value

(2) char data type:

Character data type accepts and returns all valid ASCII characters. Character data type is often said to be an integer type, since all the characters are represented in memory by their associated **ASCII Codes.** If a variable is declared as char, C++ allows storing either a character or an integer value.

Example 1:-

char c='A';

cout<<ch ;

In the above code, **ch** is declared as a char type variable to hold a character. It displays the character A

Example 2:-

char ch='A'

cout<<ch+1;

In the above statements, the value of **ch** is incremented by 1 and the new value is stored back in the same variable **ch**. (Remember that, arithmetic operations are carried out only on the numbers not with alphabets) so it displays B

Another program illustrates how **int** and **char** data types are working together.

Illustration 9.3: C++ program to get an ASCII value and display the corresponding character

```
#include <iostream>
using namespace std;
int main ()
{
        int n;
        char ch;
        cout << "\n Enter an ASCII code (0 to 255): ";
        cin >> n;
        ch = n;
        cout << "\n Equivalent Character: " << ch;
}
```

The output

```
Enter an ASCII code (0 to 255): 100
Equivalent Character: d
```

In the above program, variable **n** is declared as an int type and another variable **ch** as a char type. During execution, the program prompts the user to enter an ASCII value. If the user enters an ASCII value as an integer, it will be stored in the variable **n**. In the statement **ch = n**; the value of **n** is assigned into **ch**. Remember that, **ch** is a char type variable.

For example, if a user enters 100 as input; initially, 100 is stored in the variable **n**. In the next statement, the value of **n** i.e., 100 is assigned to **ch**. Since, **ch** is a char type; it shows the corresponding ASCII character as output. (Equivalent ASCII Character for 100 is d).

(3) float data type:

If a variable is declared as float, all values will be stored as floating point values.

There are two advantages of using float data types.

(1) They can represent values between the integers.

(2) They can represent a much greater range of values.

At the same time, floating point operations takes more time to execute compared to the integer type ie., floating point operations are slower than integer operations. This is a disadvantage of floating point operation.

For Example

```
float num=13.4;
```

In the above example, num variable is declared as float type .

(4) double data type:

This is for double precision floating point numbers. (precision means significant numbers after decimal point). The double is also used for handling floating point numbers. But, this type occupies double the space than float type. This means, more fractions can be accommodated in double than in float type. The double is larger and slower than type float. double is used in a similar way as that of float data type.

(5) void data type:

The literal meaning for void is 'empty space'. Here, in C++, the void data type specifies an empty set of values. It is used as a return type for functions that do not return any value.

Evaluate Yourself

1. What do you mean by fundemantal data types?
2. The data type char is used to represent characters. then why is it often termed as an integer type?
3. What is the advantage of floating point numbers over integers?
4. The data type double is another floating point type. Why is it treated as a distinct data type?
5. What is the use of void data type?

9.12.2 Memory representation of Fundamental Data types:

One of the most important reason for declaring a variable as a particular data type is to allocate appropriate space in memory. As per the stored program concept, every data should be accommodated in the main memory before they are processed. So, C++ compiler allocates specific memory space for each and every data handled according to

the compiler's standards.

The following Table 9.5 shows how much of memory space is allocated for each fundamental data type. Remember that, every data is stored inside the computer memory in the form of binary digits (See Unit I Chapter 2).

Table 9. 5 Memory allocation for Fundamental data types

Data type	Space in memory		Range of value
	in terms of bytes	in terms of bits	
char	1 byte	8 bits	-128 to 127
int	2 bytes	16 bits	-32,768 to 32,767
float	4 bytes	32 bits	3.4×10^{-38} to 3.4×10^{38} -1
double	8 bytes	64 bits	1.7×10^{-308} to 1.7×10^{308}-1

9.12.3 Data type modifiers:

Modifiers are used to modify the storing capacity of a fundamental data type except void type. Usually, every fundamental data type has a fixed range of values to store data items in memory. For example, int data type can store only two bytes of data. In reality, some integer data may have more length and may need more space in memory. In this situation, we should modify the memory space to accommodate large integer values. **Modifiers can be used to modify (expand or reduce) the memory allocation of any fundamental data type. They are also called as Qualifiers.**

There are four modifiers used in C++. They are:

(1) signed (2) unsigned (3) long (4) short

These four modifiers can be used with any fundamental data type. The following Table 9.6 shows the memory allocation for each data type with and without modifiers.

Integer type

Table 9.6 Memory allocation for Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
short	short is a short name for **short int**	2 bytes	16 bits	-32,768 to 32,767
unsigned short	an integer number without minus sign.	2 bytes	16 bits	0 to 65535
signed short	An integer number with minus sign	2 bytes	32 bits	-32,768 to 32,767
Both short and signed short are similar				
int	An integer may or may not be signed	2 bytes	16 bits	-32,768 to 32,767

unsigned int	An integer without any sign (minus symbol)	2 bytes	16 bits	0 to 65535
signed int	An integer with sign	2 bytes	16 bits	-32,768 to 32,767
Both short and int are similar				
long	long is short name for long int	4 bytes	32 bits	-2147483648 to 2147483647
unsigned long	A double spaced integer without any sign	4 bytes	32 bits	0 to 4,294,967,295
signed long	A double spaced integer with sign	4 bytes	32 bits	-2147483648 to 2147483647

The above table clearly shows that an integer type accepts only 2 bytes of data whereas a long int accepts data that is double this size i.e., 4 bytes of data. So, we can store more digits in a long int. (long is a modifier and int is a fundamental data type)

char type

Table 9.7 Memory allocation for char Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
char	Signed ASCII character	1 byte	8 bits	-128 to 127
unsigned char	ASCII character without sign	1 byte	8 bits	0 to 255
signed char	ASCII character with sign	1 byte	8 bits	-128 to 127

Floating point type

Table 9.8 Memory allocation for floating point Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
float	signed fractional value	4 bytes	32 bits	3.4×10^{-38} to $3.4\times10^{38}-1$
double	signed more precision fractional value	8 bytes	64 bits	1.7×10^{-308} to $1.7\times10^{308}-1$
long double	signed more precision fractional value	10 bytes	80 bits	3.4×10^{-4932} to $1.1\times10^{4932}-1$

Memory allocation is subjected to vary based on the type of compiler that is being used. Here, the given values are as per the **Turbo C++** compiler. **Dev C++** provides some more space to **int** and **long** double types. Following Tables 9.9 shows the difference between Turbo C++ and Dev C++ allocation of memory.

Table 9.9 Memory allocation of Turbo C++ and Dev C++

Data type	Memory size in bytes	
	Turbo C++	Dev C++
short	2	2
unsigned short	2	2
signed short	2	2
int	2	**4**
unsigned int	2	**4**
signed int	2	**4**
long	4	4
unsigned long	4	4
signed long	4	4
char	1	1
unsigned char	1	1
signed char	1	1
float	4	4
double	8	8
long double	10	**12**

Since, Dev C++ provides 4 bytes to int and long, any one of these types can be used to handle bigger integer values while writing programs in Dev C++.

Note: sizeof( ) is an operator which gives the size of a data type.

Number Suffixes in C++

There are different suffixes for integer and floating point numbers. Suffix can be used to assign the same value as a different type. For example, if you want to store 45 in int, long, unsigned int and unsigned long int, you can use suffix letter **L** or **U** (either case) with 45 i.e. **45L** or **45U**. This type of declaration instructs the compiler to store the given values as long and unsigned. 'F' can be used for floating point values, example: 3.14F

9.13 Variables

Variables are user-defined names assigned to specific memory locations in which the values are stored. Variables are also identifiers; and hence, the rules for naming the identifiers should be followed while naming a variable. These are called as symbolic variables because these are named locations.

There are two values associated with a symbolic variable; they are **R**-value and **L**-value.

- R-value is data stored in a memory location
- L-value is the memory address in which the R-value is stored.

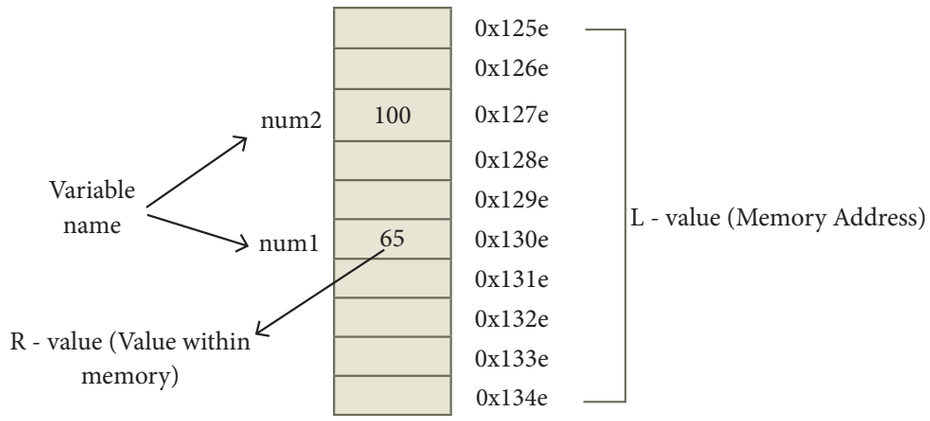


Figure 9.14 Memory allocation of a variable

Remember that, the memory addresses are in the form of Hexadecimal values

9.13.1 Declaration of Variables

Every variable should be declared before they are actually used in a program. Declaration is a process to instruct the compiler to allocate memory as per the type that is specified along with the variable name. For example, if you declare a variable as int type, in Dev C++, the compiler allocates 4 bytes of memory. Thus, every variable should be declared along with the type of value to be stored.

Declaration of more than one variable:

More than one variable of the same type can be declared as a single statement using a comma separating the individual variables.

Syntax:

<data type> <var1>, <var2>, <var3> …… <var_n>;

Example:

int num1, num2, sum;

In the above statement, there are three variables declared as int type. Which means, in **num1, num2** and **sum**, you can store only integer values.

For the above declaration, the C++ compiler allocates 4 bytes of memory (i.e. 4 memory boxes) for each variable.

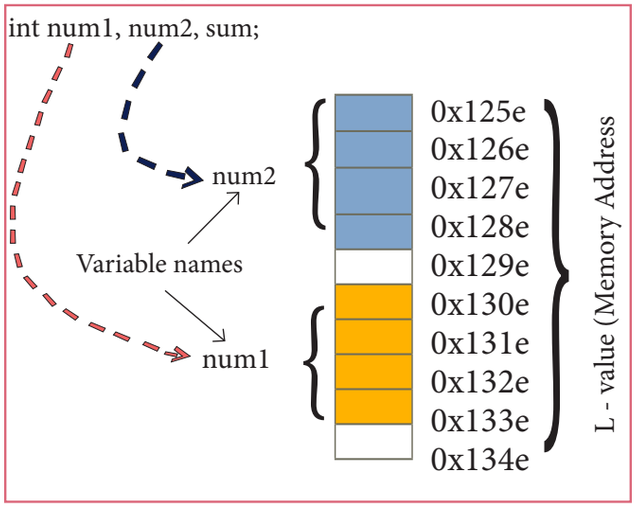


Figure 9.15 Memory allocation of int type variables

If you declare a variable without any initial value, the memory space allocated to that variable will be occupied with some unknown value. These unknown values are called as **"Junk"** or **"Garbage"** values.

```
#include <iostream>
using namespace std;
int main()
{
        int num1, num2, sum;
        cout << num1 << endl;
        cout << num2 << endl;
        cout << num1 + num2;
}
```

In the above program, some unknown values will be occupied in memory that is allocated for the variables **num1** and **num2** and the statement **cout << num1 + num2;** will display the sum of those unknown junk values.

9.13.2 Initialization of variables

Assigning an initial value to a variable during its declaration is called as "Initialization".

Examples:

int num = 100;

float pi = 3.14;

double price = 231.45;

Here, the variables num, pi, and price have been initialized during the declaration. These initial values can be later changed during the program execution.

Illustration 9.6 C++ Program to find the Curved Surface Area of a cylinder (CSA) (CSA = 2 pi r * h)

```
#include <iostream>
using namespace std;
int main()
{
   float pi = 3.14, radius, height, CSA;
   cout << "\n Curved Surface Area of a cylinder";
   cout << "\n Enter Radius (in cm): ";
   cin >> radius;
   cout << "\n Enter Height (in cm): ";
   cin >> height;
   CSA = (2*pi*radius)*height;
   system("cls");
   cout << "\n Radius: " << radius <<"cm";
   cout << "\n Height: " << height << "cm";
   cout << "\n Curved Surface Area of a Cylinder is " << CSA <<" sq. cm.";
}
```

Output:

```
Curved Surface Area of a cylinder
Enter Radius (in cm): 7
Enter Height (in cm): 20
Radius: 7cm
Height: 20cm
Curved Surface Area of a Cylinder is 879.2 sq. cm.
```

Variables that are of the same type can be initialized in a single statement.

Example:

int x1 = -1, x2 = 1, x3, n;

9.13.3 Dynamic Initialization

A variable can be initialized during the execution of a program. It is known as "**Dynamic initialization**". For example,

int num1, num2, sum;

sum = num1 + num2;

The above two statements can be combined into a single one as follows:

int sum = num1+num2;

This initializes sum using the known values of num1 and num2 during the execution.

Illustration 9.7 C++ Program to illustrate dynamic initialization

```
#include <iostream>
using namespace std;
int main()
{
        int num1, num2;
        cout << "\n Enter number 1: ";
        cin >> num1;
        cout << "\n Enter number 2: ";
        cin >> num2;
        int sum = num1 + num2; // Dynamic initialization
        cout << "\n Average: " << sum /2;
}
```

Output:

```
Enter number 1: 78
Enter number 2: 65
Average: 71
```

In the above program, after getting the values of num1 and num2, sum is declared and initialized with the addition of those two variables. After that, it is divided by 2.

Illustration 9.8: C++ program to find the perimeter and area of a semi circle

```
#include <iostream>
using namespace std;
int main()
{
        int radius;
        float pi = 3.14;
        cout << "\n Enter Radius (in cm): ";
        cin >> radius;
        float perimeter = (pi+2)*radius;   // dynamic initialization
        float area = (pi*radius*radius)/2; // dynamic initialization
        cout << "\n Perimeter of the semicircle is " << perimeter << " cm";
        cout << "\n Area of the semicircle is " << area << " sq.cm";
}
```

Output:

```
Enter Radius (in cm): 14
Perimeter of the semicircle is 71.96 cm
Area of the semicircle is 307.72 sq.cm
```

9.13.4 The Access modifier const

const is the keyword used to declare a constant. You already learnt about constant in the previous chapter. const keyword modifies / restricts the accessibility of a variable. So, it is known as Access modifier.

For example,

int num = 100;

The above statement declares a variable num with an initial value 100. However, the value of num can be changed during the execution. If you modify the above definition as **const int num = 100;** the variable num becomes a constant and its value will remain 100 throughout the program, and it can never be changed during the execution.

```
#include <iostream>
using namespace std;
int main()
{
        const int num=100;
        cout << "\n Value of num is = " << num;
        num = num + 1; // Trying to increment the constant
        cout << "\n Value of num after increment " << num;
}
```

In the above code, an error message will be displayed as **"Cannot modify the const object"** in Turbo compiler and **"assignment of read only memory num"** in Dev C++.

9.13.5 References

A reference provides an alias for a previously defined variable. Declaration of a reference consists of base type and an & (ampersand) symbol; reference variable name is assigned the value of a previously declared variable.

Syntax:

<type> <& reference_variable> = <original_variable>;

Illustration 9.9: C++ program to declare reference variable

```
#include <iostream>
using namespace std;
int main()
{
        int num;
        int &temp = num; //declaration of a reference variable temp
        num = 100;
cout << "\n The value of num = " << num;
cout << "\n The value of temp = " << temp;
}
```

The output of the above program will be

The value of num = 100

The value of temp = 100

Evaluate Yourself

1. What are modifiers? What is the use of modifiers?
2. What is wrong with the following C++ statement:
   long float x;
3. What is a variable ? Why is a variable called symblolic variable?
4. What do you mean by dynamic initialization of a variable? Give an exmple.
5. What is wrong with the following statement?
   const int x;

9.14 Formatting Output

Formatting output is very important in the development of output screens for easy reading and understanding. Manipulators are used to format the output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.

C++ offers several input and output manipulators for formatting. Commonly used manipulators are: **endl, setw, setfill, setprecision and setf**. In order to use these manipulators, you should include the appropriate header file. **endl** manipulator is a member of iostream header file. **setw, setfill, setprecision and setf** manipulators are members of iomanip header file.

endl (End the Line)

endl is used as a line feeder in C++. It can be used as an alternate to '\n'. In other words, endl inserts a new line and then makes the cursor to point to the beginning of the next line. There is a difference between endl and '\n', even though they are performing similar tasks.

- endl – Inserts a new line and flushes the buffer (Flush means – clean)
- '\n' - Inserts only a new line.

Example:

```
cout << "\n The value of num = " << num;
cout << "The value of num = " << num <<endl;
```

Both these statements display the same output.

setw ( )

setw manipulator sets the **width of the field** assigned for the output. The field width determines the minimum number of characters to be written in output.

Syntax:

setw(number of characters)

Illustration 9.10: Program to Calculate Net Salary

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        float basic, da, hra, gpf, tax, gross, np;
        char name[30];
        cout << "\n Enter Basic Pay: ";
        cin >> basic;
        cout << "\n Enter D.A : ";
        cin >> da;
        cout << "\n Enter H.R.A: ";
        cin >> hra;
        gross = basic+da+hra; // sum of basic, da and hra
        gpf = (basic+da) * 0.10; // 10% 0f basic and da
        tax = gross * 0.10; //10% of gross pay
        np = gross - (gpf+tax); //netpay = earnings - deductions
        cout << setw(25) << "Basic Pay : " << setw(10)<< basic<< endl;
        cout << setw(25) << "Dearness Allowance : "<< setw(10)<<da<< endl;
        cout<<setw(25)<<"House Rent Allowance : "<<setw(10)<< hra<<endl;
        cout << setw(25) << "Gross Pay : " << setw(10) << gross << endl;
        cout << setw(25) << "G.P.F : " << setw(10) << gpf << endl;
        cout << setw(25) << "Income Tax : " << setw(10)<< tax << endl;
        cout << setw(25) << "Net Pay : " << setw(10) << np << endl;
}
```

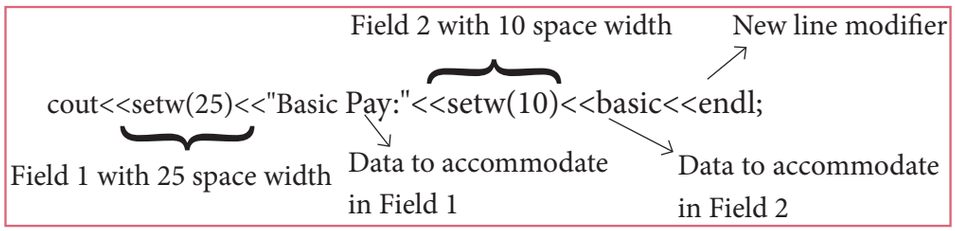
The output will be,

```
Enter Basic Pay: 12000
Enter D.A : 1250
Enter H.R.A : 1450
                    Basic Pay  :      12000
           Dearness Allowance  :       1250
       House Rent Allowance    :       1450
                    Gross Pay  :      14700
                        G.P.F  :       1325
                   Income Tax  :       1470
                      Net Pay  :      11905
```

(HOT: Try to make multiple output statements as a single cout statement)

In the above program, every output statement has two setw( ) manipulators; first setw (25) creates a filed with 25 spaces and second setw(10) creates another field with 10 spaces. When you

represent a value to these fields, it will show the value within the field from right to left.



In field1 and field 2, the string "Basic Pay: " and the value of basic pay are shown as given in Figure 9.16 below.

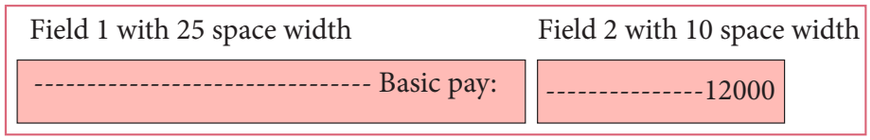


Figure 9.16 setw( ) function

setprecision ( )

This is used to display numbers with fractions in specific number of digits.

Syntax:

setprecision (number of digits);

Example:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{     float hra = 1200.123;
      cout << setprecision (5) << hra; }
```

In the above code, the given value 1200.123 will be displayed in 5 digits including fractions. So, the output will be **1200.1**

setprecision ( ) prints the values from left to right. For the above code, first, it will take 4 digits and then prints one digit from fractional portion.

setprecision can also be used to set the number of decimal places to be displayed. In order to do this task, you will have to set an ios flag within **setf()** manipulator. This may be used in two forms: (i) **fixed** and (ii) **scientific**

These two forms are used when the keywords fixed or scientific are appropriately used before the setprecision manipulator.

Example:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{     cout.setf(ios::fixed);
      cout << setprecision(2)<<0.1; }
```

In the above program, ios flag is set to **fixed** type; it prints the floating point number in fixed notation. So, the output will be, 0.10

cout.setf(ios::scientific);

cout << setprecision(2) << 0.1;

In the above statements, ios flag is set to **scientific type;** it will print the floating point number in scientific notation. So, the output will be, 1.00e-001

9.15 Expression

An expression is a combination of operators, constants and variables arranged as per the rules of C++. It may also include function calls which return values. (Functions will be learnt in upcoming chapters).

An expression may consist of one or more operands, and zero or more operators to produce a value. In C++, there are seven types of expressions, and they are:

(i) Constant Expression (ii) Integer Expression
(iii) Floating Expression (iv) Relational Expression
(v) Logical Expression (vi) Bitwise Expression
(vii) Pointer Expression

SN	Expression	Description	Example
1	Constant Expression	Constant expression consist only constant values	int num=100;
2	Integer Expression	The combination of integer and character values and/or variables with simple arithmetic operators to produce integer results.	sum=num1+num2; avg=sum/5;
3	Float Expression	The combination of floating point values and/or variables with simple arithmetic operators to produce floating point results.	Area=3.14*r*r;
4	Relational Expression	The combination of values and/or variables with relational operators to produce bool(true means 1 or false means 0) values as results.	x>y; a+b==c+d;
5	Logical Expression	The combination of values and/or variables with Logical operators to produce bool values as results.	(a>b)&& (c==10);
6	Bitwise Expression	The combination of values and/or variables with Bitwise operators.	x>>3; a<<2;
7	Pointer Expression	A Pointer is a variable that holds a memory address. Pointer variables are declared using (*) symbol.	int *ptr;

Table 9.10 : Types of Expressions

9.16 Type Conversion

The process of converting one fundamental data type into another is called as "Type Conversion". C++ provides two types of conversions.

(1) Implicit type conversion

(2) Explicit type conversion.

(1) Implicit type conversion:

An Implicit type conversion is a conversion performed by the compiler automatically. So, implicit conversion is also called as **"Automatic conversion".**

This type of conversion is applied usually whenever different data types are intermixed in an expression. If the type of the operands differ, the compiler converts one of them to match with the other, using the rule that the "smaller" type is converted to the "wider" type, which is called as **"Type Promotion".**

For example:

```
#include <iostream>
using namespace std;
int main()
{
        int a=6;
        float b=3.14;
        cout << a+b;
}
```

In the above program, operand **a** is an int type and **b** is a float type. During the execution of the program, int is converted into a float, because a float is wider than int. Hence, the output of the above program will be: **9.14**

The following Table 9.11 shows you the conversion pattern.

RHO / LHO	**char**	**short**	**int**	**long**	**float**	**double**	**long double**
char	int	int	int	long	float	double	long double
short	int	int	int	long	float	double	long double
int	int	int	int	long	float	double	long double
long	long	long	long	long	float	double	long double
float	float	float	float	float	float	double	long double
double	double	double	double	double	double	double	long double
long double	long double	long double	long double	long double	long double	long double	long double

(RHO – Right Hand Operand; LHO – Left Hand Operand)

Table 9.11: Implicit conversion of mixed operands

(2) Explicit type conversion

C++ allows explicit conversion of variables or expressions from one data type to another specific data type by the programmer. It is called as **"type casting".**

Syntax:

(type-name) expression;

Where type-name is a valid C++ data type to which the conversion is to be performed.

Example:

```
#include <iostream>
using namespace std;
int main( )
{
        float varf=78.685;
        cout << (int) varf;
}
```

In the above program, variable **varf is** declared as a **float** with an initial value 78.685. The value of **varf** is explicitly converted to an **int** type in cout statement. Thus, the final output will be 78.

During explicit conversion, if you assign a value to a type with a greater range, it does not cause any problem. But, assigning a value of a larger type to a smaller type may result in loosing or loss of precision values.

S.No	Explicit Conversion	Problem
1	double to float	Loss of precision. If the original value is out of range for the target type, the result becomes undefined
2	float to int	Loss of fractional part. If original value may be out of range for target type, the result becomes undefined
3	long to short	Loss of data

Table 9.12 – Explicit Conversion Problems

Example:

```
#include <iostream>
using namespace std;
int main()
{
        double varf=178.25255685;
        cout << (float) varf << endl;
        cout << (int) varf << endl;
}
```

Output:

```
178.253
178
```

Evaluate Yourself

1. What is meant by type conversion?
2. How implicit conversion is different from explicit conversion?
3. What is the difference between endl and \n?
4. What is the use of references?
5. What is the use of setprecision ( ) ?

Hands on practice:

1. Write C++ programs to interchange the values of two variables.
   a. Using the third variable
   b. Without using third variable
2. Write C++ programs to do the following:
   a. To find the perimeter and area of a quadrant.
   b. To find the area of triangle.
   c. To convert the temperature from Celsius to Fahrenheit.
3. Write a C++ to find the total and percentage of marks you secured from 10th Standard Public Exam. Display all the marks one-by-one along with total and percentage. Apply formatting functions.

Points to Remember

- Every programming language has two fundamental elements, viz., data types and variables.
- In C++, the data types are classified as three main categories (1) Built-in data types (2) User-defined data types (3) Derived data types.
- The variables are the named space to hold values of certain data type.
- There are five fundamental data types in C++: char, int, float, double and void.
- C++ compiler allocates specific memory space for each and every data handled according to the compiler's standards.
- Variables are user-defined names assigned to a memory location in which the values are stored.
- Declaration is a process to instruct the compiler to allocate memory as per the type specified along with the variable name.
- Manipulators are used to format output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.
- An expression is a combination of operators, constants and variables arranged as per the rules of C++.
- The process of converting one fundamental data type into another is called as "Type Conversion". C++ provides two types of conversions (1) Implicit type conversion and (2) Explicit type conversion.

Evaluation

SECTION – A

F9W7JM

Choose the correct answer

1. How many categories of data types are available in C++?
   (a) 5 (b) 4 (c) 3 (d) 2
2. Which of the following data types is not a fundamental type?
   (a) signed (b) int (c) float (d) char
3. What will be the result of following statement?
   char ch= 'B';
   cout << (int) ch;
   (a) B (b) b (c) 65 (d) 66
4. Which of the character is used as suffix to indicate a floating point value?
   (a) F (b) C (c) L (d) D
5. How many bytes of memory is allocated for the following variable declaration if you are using Dev C++? short int x;
   (a) 2 (b) 4 (c) 6 (d) 8
6. What is the output of the following snippet?
   char ch = 'A';
   ch = ch + 1;
   (a) B (b) A1 (c) F (d) 1A
7. Which of the following is not a data type modifier?
   (a) signed (b) int (c) long (d) short
8. Which of the following operator returns the size of the data type?
   (a) sizeof( ) (b) int ( ) (c) long ( ) (d) double ( )
9. Which operator is used to access reference of a variable?
   (a) $ (b) # (c) & (d) !
10. This can be used as alternate to endl command:
    (a) \t (b) \b (c) \0 (c) \n

SECTION-B

Very Short Answers

1. Write a short note on const keyword with an example.
2. What is the use of setw( ) format manipulator?
3. Why is char often treated as integer data type?
4. What is a reference variable? What is its use?
5. Consider the following C++ statement. Are they equivalent?
   char ch = 67; char ch = 'C';

6. What is the difference between 56L and 56?

7. Determine which of the following are valid constant? And specify their type.

   (i) 0.5 (ii) 'Name' (iii) '\t' (iv) 27,822

8. Suppose x and y are two double type variable that you want add as integer and assign to an integer variable. Construct a C++ statement to do the above.

9. What will be the result of following if num=6 initially.

   (a) cout << num;

   (b) cout << (num==5);

10. Which of the following two statements are valid? Why? Also write their result.

    (i) int a; a = 3,014; (ii) int a; a=(3,014);

SECTION-C

Short Answers

1. What are arithmetic operators in C++? Differentiate unary and binary arithmetic operators. Give example for each of them.

2. How relational operators and logical operators are related to one another?

3. Evaluate the following C++ expressions where x, y, z are integers and m, n are floating point numbers. The value of x = 5, y = 4 and m=2.5;

   (i) n = x + y / x;

   (ii) z = m * x + y;

   (iii) z *= x * m + x;

Reference:

(1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.

(2) The Complete Reference C++ (Forth Edition), Herbert Schildt. Mc.Graw Hills.

(3) Computer Science with C++ (A text book of CBSE XI and XII), Sumita Arora, Dhanpat Rai & Co.