COMPUTER SCIENCE

Higher Secondary - Second Year

Volume 2 : Object Technology

Government of Tamilnadu

Untouchability is a sin
Untouchability is a crime
Untouchability is inhuman

TAMILNADU
TEXTBOOK CORPORATION
College Road, Chennai - 600 006.

Government of Tamilnadu
First Edition – 2005

Chairman Syllabus Committee
Dr. Balagurusamy E, Former Vice Chancellor, Anna University, Chennai

Co-Ordinator Textbook Writing
Dr. Sankaranarayanan V, Director, Tamil Virtual University, Chennai

Authors
Dr. Gopal T V, Anna University, Chennai
Ms. Rukmini K, Padma Seshadri Bala Bhavan, Chennai
Dr. Sankaranaraynan V, Director, Tamil Virtual University, Chennai

Reviewers
Dr. Gopal T V, Anna University, Chennai
Ms. Vasanthi Krishnakumar, M/s Sify Limited, Chennai

Copy-Editor
Ms. Subha Ravi, Director, M/s Digiterati Consultancy Pvt. Ltd,Chennai

Price Rs. :

This book has been prepared by the Directorate of School Education on behalf of the Government of Tamilnadu

This book has been printed on 70 G.S.M. Paper

FOREWORD

"In human affairs we have reached a point where the problems that we must solve are no longer solvable without the aid of computers. I fear not computers but the lack of them"

-**Issac Asimov**

Computers are machines that can help us solve complex scientific, business and administrative problems. They have helped automation of many industrial and business systems. However, we must remember that they are machines, created and managed by men. They have no brain of their own. Anything they do is the result of human instructions. They carry out the instructions obediently as long as the instructions can be executed using the available hardware, no matter whether they are right or wrong. That is, computers lack common sense.

Computers need clear instructions to tell them what to do, how to do, and when to do. The way of providing such instructions to computers is called programming. The language used in construction and communication of these instructions is known as a programming language.

There are over 200 programming languages currently in use. Some were designed for scientific use, some for commercial applications while some others were meant for more general-purposes . A programming language should have features that would facilitate programmers in making and designing the solution steps easily.

We have already learned the C language, which is a procedure-oriented language. As the name implies, the emphasis was on solution procedures. C is a powerful general-purpose language. This volume presents the advanced version of C know as C++. C++ supports a totally new concept of object-oriented programming (OOP) and therefore it is classified as an object-oriented technology. We chose C++ because it has become an industry-standard OOP language today.

Continued

In OOP languages such as C++, the emphasis is on the entities of the physical world called objects. These objects may represent a person, a car, a table of data, or any item that the program must handle. We human beings normally look at real-life problems as a collection of distinct objects and try to solve them taking into account the relationship among the objects. In a similar way, in C++, programming problems are analyzed in terms of objects and the nature of communications between them.

Numerous examples and illustrative programs presented in the volume are meant to be both simple and educational. It is hoped that the material provided will help the students to quickly move into the world of C++ and object-oriented programming.

This volume also presents many IT enabled applications with a visual support in the form of animated content (in a separate CD). Ethical use of IT has been highlighted in all applications.

I would like to place on record our sincere appreciation and thanks to all the authors, reviewers and the Directorate of School Education officials for their excellent work and co-operation.

E. BALAGURUSAMY
Chairman
Syllabus Review Committee

CONTENTS

CHAPTER I

OBJECT ORIENTED CONCEPTS USING C++

1.1 Object Oriented Programming Language

A computer is a tool to solve a wide range of problems. The solutions to the problems are in the form of computer programs or application software. These programs are written using a chosen programming language.

A computer program operates on a set of known input data items. The program transforms this input data into a set of expected data items. Only this set of expected data items must be the output of the computer program.

In the early programming languages the input and output data items were represented as variables. Data types categorized these input data items. Control statements provided a way of instructing the computer on the operations that need to be performed on the data items.

Programming languages have another use. They help us in organizing our ideas about the solution of the problem. As the problems being solved or the applications being developed became complex, this aspect of programming languages became very important. Many programming languages emerged to address this issue along with the ease of instructing the computer.

It was realized that viewing the solution of a problem as two separate segments 'data' and 'operations' does not resemble the way human beings solve the real life problems.

Object oriented programming languages such as C++ are based on the way human beings normally deal with the complex aspects of real life. It has been observed that human beings normally solve real life problems by identifying distinct objects needed for the solution. Human beings then recognize the relationships amongst these objects. The relationships are like 'part of the whole' or are 'a type of'. Simple abilities such as recognizing that one object is a part of the bigger object and one object is a type of another object are proving to be very important in solving problems in real life. Object Oriented programming facilitates this way of problem solving by combining 'data' and 'operations' that are to be performed on the data.

In other words, the set of data items is split into smaller groups such that a set of operations can be performed on this group without calling any other function. This group of data and the operations together are termed - 'object'. The operations represent the behavior of the object. An object attempts to capture a real world object in a program.

For example, take a look at any calculator, it has both state and behaviour. Its state refers to its physical features like its dimensions, buttons, display screen, operators and the like. Its behaviour refers to the kind of functions it can perform like addition, subtraction, storing in memory, erasing memory and the like.

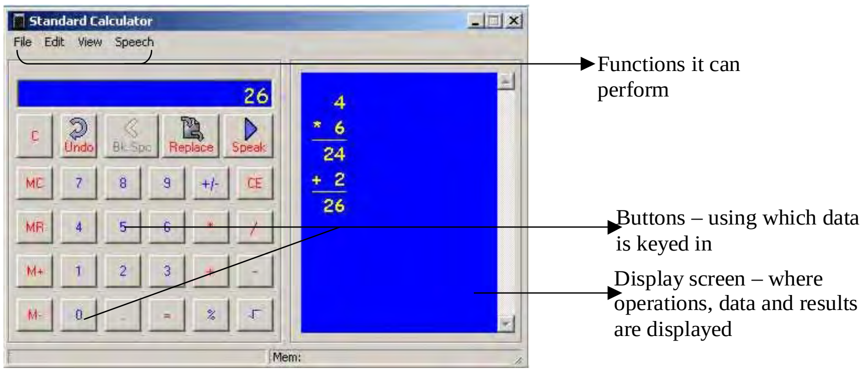


Fig. 1.1 Standard Calculator

In an object oriented programming language, a calculator is viewed as follows :

Object – calculator
Data :
Number1,result, operator, Number_backup

Functions :
Additon()
Subtraction()
Erase_Memory()
Display_Result()

- ✓ **An object is a group of related functions and data that serves those functions.**
- ✓ **An object is a kind of a self-sufficient "subprogram" with a specific functional area.**

The process of grouping data and its related functions into units called as objects paves way for **encapsulation**.

The mechanism by which the data and functions are bound together within an object definition is called as ENCAPSULATION.

It is easy to see how a bank-account, a student, a bird, a car , a chair etc., embodies both state and behaviour. It is this resemblance to real things that gives objects much of their power and appeal. Objects make it easy to represent real systems in software programs.

Examples of objects – BANK ACCOUNT & STUDENT

BANK ACCOUNT	STUDENT
Data :	**Data :**
Account number – long int	Date_of_birth – char[10]
Name – char[15]	Name – char[15];
Opening balance –float;	Class, sec char[4];
Account type – char	Marks float
Functions :	**Functions :**
Accept_Details()	Accept_Details()
Display_Details()	Display_Details()
Update_opening_balance()	Total()
Withdrawls()	Average()
Deposit()	Grading()

1.2 Polymorphism

Now let us consider the job of drawing different shapes like a rectangle, square, circle and an arc. We tend to define different functions to draw these different shapes. The definitions may be like this :

Draw_Square() Read side Draw required lines	Draw_Rectangle() Read length,breadth Draw required lines	Draw_Circle() Read radius Draw	Draw_Arc() Read Start_angle, End_angle,radius draw

Now look at the following function :

Draw(side) – is defined to draw a square
Draw (length, breadth) - is defined to draw a rectangle
Draw(radius) - is defined to draw a circle
Draw(radius,start_angle,end_angle) – to draw an arc

The function draw() accepts different inputs and performs different functions accordingly. As far as the user is concerned, he will use the function **draw()** to draw different objects with different inputs. This differential response of the function draw() based on different inputs is what is called as **polymorphism**.

The ability of an object to respond differently to different messages is called as **polymorphism**.

1.3 Inheritance

The data type **Class** conventionally represents an object in the real world. Class is a template for entities that have common behaviour. For example animals form a group of living beings, or in other words **animals** is a class. We know that animals are divided into mammals, reptiles, amphibians, insects, birds and so on. All animals share common behaviour and common attributes. Eyes, skin, habitat, food refer to the features or attributes of the animals, while reproduction, living_style, prey_style etc refers to the behaviour of the animals. Every sub group of animals has its own unique features or styles apart from the common behaviour and features. The sub groups do share the properties of the parent class – "ANIMALS" apart from its own sub classes viz ., mammals, reptiles, amphibians, insects, birds. This may be pictorially depicted as follows :

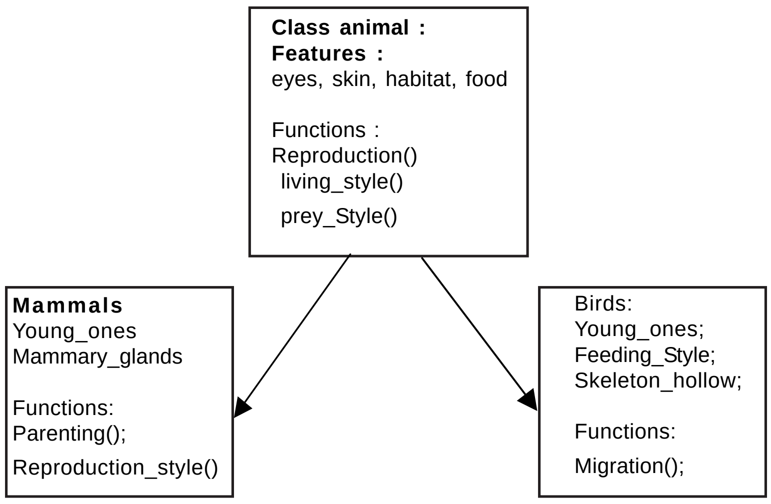


Fig. 1.1 Inheritance

Animals is called the base class, and Mammals and Birds are called derived classes. The derived classes are power packed, as they include the functionality of the base class along with their own unique features. This process of acquiring the Base class properties is what is called **inheritance** .

Inheritance increases the functionality of a derived class and also promotes reusability of code (of the base class).

Advantages of Object Oriented Programming –

- ✓ Class data type allows programs to organize as objects that contain both data and functions .
- ✓ Data hiding or Abstraction of data provides security to data, as unrelated member functions(functions defined outside the class) cannot access its data, or rather it reveals only the essential features of an object while curtailing the access of data
- ✓ Polymorphism reduces software complexity, as multiple definitions are permitted to an operator or function
- ✓ Inheritance allows a class to be derived from an existing class , thus promoting reusability of code, and also promote insertion of updated modules to meet the requirements of the dynamic world

1.4 A Practical Example : Domestic Waterusage

Fig.1.2 Domestic Waterusage

For example, let us consider developing a program that modeled home water usage. The objective of this program is to compute the water consumed by each outlet in a building and also total consumption. All that we require for this program is the number of taps installed in the building, amount of water that flowed through each tap, and finally the amount of water consumed. Each tap may be viewed as an object. The functions associated would be to start and stop the flow of water, return the amount of water consumed in a given period, and so on. To do this work, the tap object would need instance variables to keep track of whether the tap is open or shut, how much water is being used, and where the water is coming from. The object may be visualised as:

Data
Tap_open, Qty_water, Water_source

Functions
Start()
Stop()

Fig.1.3 Tap as an Object

The program that models water usage will also have WaterPipe objects that delivers water to the taps . There could be a Building object to coordinate a set of WaterPipes and taps. When a Building object is asked as to how much water is being used, it might call upon each tap and pipe to report its current state. The project may be visualised as shown in Fig.1.4.

Now the total_amount of water consumed would be calculated as t1.water_consumed() + t2.water_consumed+t3.water_consumed() and water consumption by each outlet would be given away individually by t1.water_consumed, t2.water_consumed() and so on.

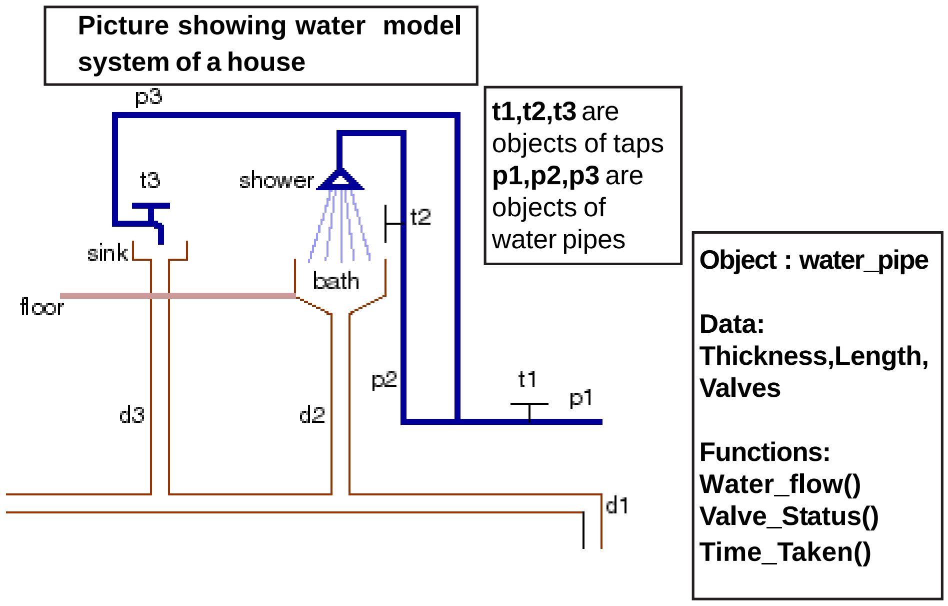


Fig.1.4 Water Distribution System in a House

t1.water_consumed() would in turn communicate with p1 to get the amount of water flowed through that pipe, as tap-1s(t1) water consumption is determined by pipe1(p1). Thus a program consists of objects that call each other to compute. Each object has a specific role to play, and the co-ordinated working of all the modules produces the end result of a program. Objects communicate with one another by sending data as inputs.

Everyday programming terminology is filled with analogies to real-world objects like tables, students, managers, bank accounts, and the like. Using such entities as programming objects merely extends the comparison in a natural way. This line of thinking about functions and object behaviour is the key factor of object-oriented programming.

Exercises

I. Fill in the blanks

a. _____________ model entities in the real world

b. Binding of data and member functions together is called as _____________

c. The ability of an object to respond differently to different messages is called as ________________

d. The process of creating new data types from existing data type is called as ___________________

II. Answer the following briefly

1. What is the significance of an object ?
2. What is Encapsulation?
3. How is polymorphism different from inheritance?

III Design data type for the following project

A company wishes to prepare a data model for its activities. The company stores information of all its employees. The common details of all employees are : Name, date_of_birth,language and nativity.

Additional details of employees based on their placement are stored as :

a. Stores – date of joining, dept, salary

b. Scientist – area of specialisation, current project details, paper_presentations

c. Technician – Height, Weight, ailments, risk factor, department, wages.

CHAPTER 2

OVERVIEW OF C++

2.1 Introduction

C++ was developed at AT & T Bell laboratories in the early 1980s by ***Bjarne Stroustrup.*** The name C++ (pronounced as C plus plus) was coined by Rick Mascitti where "++" is the C increment operator.

2.2 C++ character set

Like the C language, C++ also comprises a character set from which the tokens (basic types of elements essential for programming coding) are constructed. The character set comprises of "A" .. "Z" , "a" .. "z", 0 .. 9, +, -, /, *,\, (,), [,], {, }, =, !=, <, >, **.** , ' " ; : %, ! , &, ?, _, #, <=, >=, @, white space, horizontal tab, carriage return and other characters.

A quick recap of the basic types : The basic types are collectively called as **TOKENS**. A token is the smallest individual unit in a program. Tokens are classified as shown in Fig 2.1.

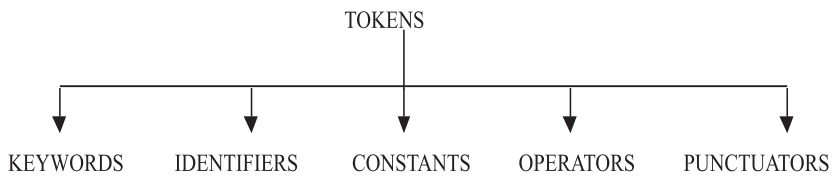


Fig. 2.1 Classification of Tokens

2.2.1 Keywords

Keywords have special meaning to the language compiler. These are reserved words for special purpose. These words cannot be used as normal identifiers. Table 2.1 shows the list of keywords used in C++.

auto	break	case	const	class	continue
default	delete	do	else	enum	for
friend	goto	if	inline	new	operator
private	protected	public	return	signed	sizeof
static	struct	switch	this	unsigned	virtual
while					

Table 2.1 Keywords

2.2.2 Identifiers

Identifiers are also called as variables. Variables are memory boxes that hold values or constants. A variable name must begin with an alphabet or underscore followed by alphabets or numbers. For example *_test ; test ; sum*12 are some valid identifiers. We shall see more about variables after dealing with data types

2.2.3 Constants

Constants are data items whose values cannot be changed. A constant is of numeric or non-numeric type. Numeric constants consist of only numbers, either whole numbers or decimal numbers. Integer, floating-point are numeric constants.

2.2.4 Integer Constant

- Integer Constant must have at least one digit and must not contain any fractional part.
- May be prefixed with a + or – sign
- A sequence of digits starting with **0** (zero) is treated as Octal constant Ex. 010 = 8 ($[8]_{10} = [10]_{8}$)
- A sequence of digits starting with **0x** is treated as hexadecimal integer. Ex. 0xF = 15 ($[15]_{10} = [F]_{16}$)

2.2.5 Floating Point Constant

Floating Point Constant is a signed real number. It includes an integer portion, a decimal point, a fractional portion and an exponent. While representing a floating point constant the integer portion or the decimal portion can be omitted but never both. For example 58.64 is a valid floating point (Real) constant. It can be represented in exponent form as follows :

- 5.864E1 => 5.864 x 10^1 => 58.64
- 5864E-2 => 5864 x 10^{-2} => 58.64
- 0.5864E2 => 0.5864x10^2 => 58.64

The letter E or e is used to represent the floating-point constant exponent form.

2.2.6 Character Constant

Character constant is a constant that contains a single character enclosed within single quotes. It can be any character as defined in the character set of C++ language (alphabet, numeral, mathematical, relational or any other special character as part of the ASCII set). Certain special characters like tab, backspace, line feed, null, backslash are called as non-graphic character constants. These characters are represented using escape sequences. Escape sequences are represented using characters prefixed with a backslash. Table 2.2 shows the escape sequences.

Escape Sequence	Nongraphic Character
\a	Bell
\b	Back space
\n	New line/ line feed
\t	Horizontal tab
\v	Vertical tab
\\	Back slash
\’ or \”	Single / double quotes
\o	Octal number
\x	Hexadecimal number
\0	Null

Table 2.2 Escape Sequences

2.2.7 String Literal

String Literal is a sequence of characters surrounded by double quotes. String literals are treated as array of characters. Each string literal is by default added with a special character ‘\0’ which marks the end of a string. For example “testing”

2.2.8 Operator

Operator specifies an operation to be performed that yields a value. An operand is an entity on which an operator acts. For example :

RESULT = NUM1 + NUM2

NUM1 and NUM2 are operands. **+** is the additional operator, that performs the addition of the numbers. The result (value) generated is stored in the variable RESULT by virtue of “=” (Assignment) operator. Table 2.3 shows the operators in C++.

[ ]	*	%	==	=	>=
( )	+	<<	!=	*=	&=
.	-	>>	^	/=	^=
->	~	<	|	+=	|=
++	!	>	&&	-=	, --
	size of	<=	||	%=	#
&	/	>=	?:	<<=	##

Table 2.3 Operators in C++

The following operators are specific to C++.

::   .*   ->*

The operators # and ## are used only by the preprocessor.

Operators are classified as

- Arithmetic
- Assignment
- Component Selection
- Conditional
- Logical
- Manipulator
- Member dereferencing
- Memory Management
- Preprocessor
- Relational
- Scope Resolution
- Shift
- Type Cast

Based on operand requirements, operators are also classified as unary, binary and ternary operators.

For example :

Unary operators require one operand
Binary operator requires two operands
Ternary operator requires three operands.

Table 2.4b Binary Operators

Additive	+	Binary Plus
	-	Binary minus
Multiplicative	*	Multiply
	/	Divide
	%	Remainder (Modulus)
Shift	<<	Shift Left
	>>	Shift Right
Bitwise	&	AND
	|	OR
	^	XOR
Logical	&&	Logical AND
	||	Logical OR
Assignment	=	Assignment
	/=	Assign quotient
	+=	Assign sum
	*=	Assign product
	%=	Assign remainder
	-=	Assign difference
	<<=	Assign left shift
	>>=	Assign right shift
	&=	Assign bitwise AND
	^=	Assign bitwise XOR
	|=	Assign bitwise OR
Relational	<	Less Than
	>	Greater than
	<=	Less than or Equal to
	>=	Greater that or Equal to
Equality	= =	Equal to
	! =	Not Equal to
Component Selection	.	Direct Component Selection
	->	Indirect Component Selection
Class member	::	Scope access/Resolution operator
	. *	Dereference Operator
	-> *	Dereference pointer to class member
Conditional	? :	Ternary operator
Comma	,	Evaluate

Table 2.4a
Unary Operators

&	Address of
!	Logical Not
*	Indirection
++	Increment
~	Bitwise
--	Decrement
-	Unary minus
+	Unary plus

2. 2.7.1 Arithmetic Operators

Arithmetic Operators are used to perform mathematical operations. The list of arithmetic operators are :

- +
- -
- * multiplication operator
- / division operator
- % modulus operator - gives the remainder of an integer divison
- += , -=, *= , /= , %=

Arithmetic expressions are formed using arithmetic operators, numerical constants/variables, function call connected by arithmetic operators.

Examples :

- a = -5;
- a = +b;
- a /= 5; (a = a/5)
- a++; (Post increment operator . Equivalent to a = a+1)
- a— ; (Post decrement operator. Equivalent to a = a-1)
- ++a; (Pre increment operator. Equivalent to a = a+1)
- —a ; (Pre decrement operator. Equivalent to a = a – 1)
- a *= 2 (a = a * 2)
- a %= 5 (a = a/5)
- a = b + pow(x,y) (a = b + x^y)

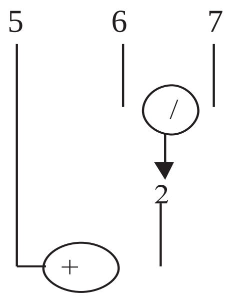
Operators are executed in the order of precedence. The operands and the operators are grouped in a specific logical way for evaluation. This logical grouping is called as association. Table 2.5 indicates the Mathematical Operators, its Type, and Association.

Operator Precedence	Type	Associativity
() []	Mathematical	Left to right
Postfix ++, -- ,	- Unary	Left to right
prefix ++, --		Right to left
+unary , - unary * / %	mathematical Mathematical –binary	Right to left Right to left Left to right
+ -	Mathematical– binary	Left to right

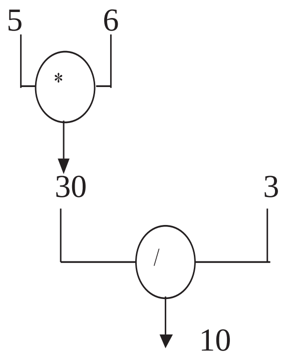
Table 2.5 Mathematical Operator Precedence

The following examples demonstrate the order of evaluation in arithmetic expressions :

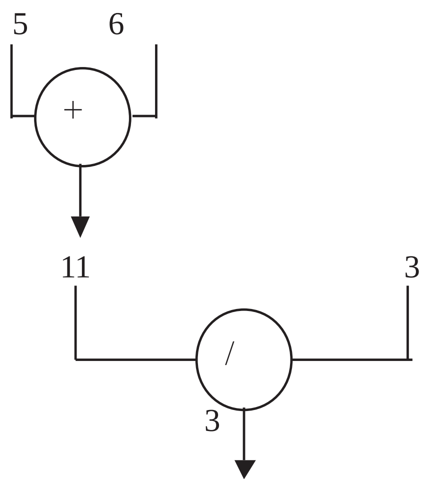
5 + 6/3 will yield the result as 7



5 * 6 / 3



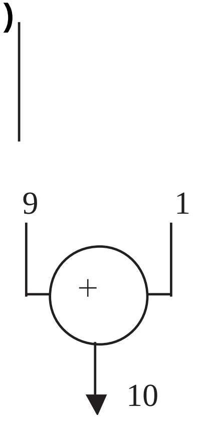
(5 + 6) / 3

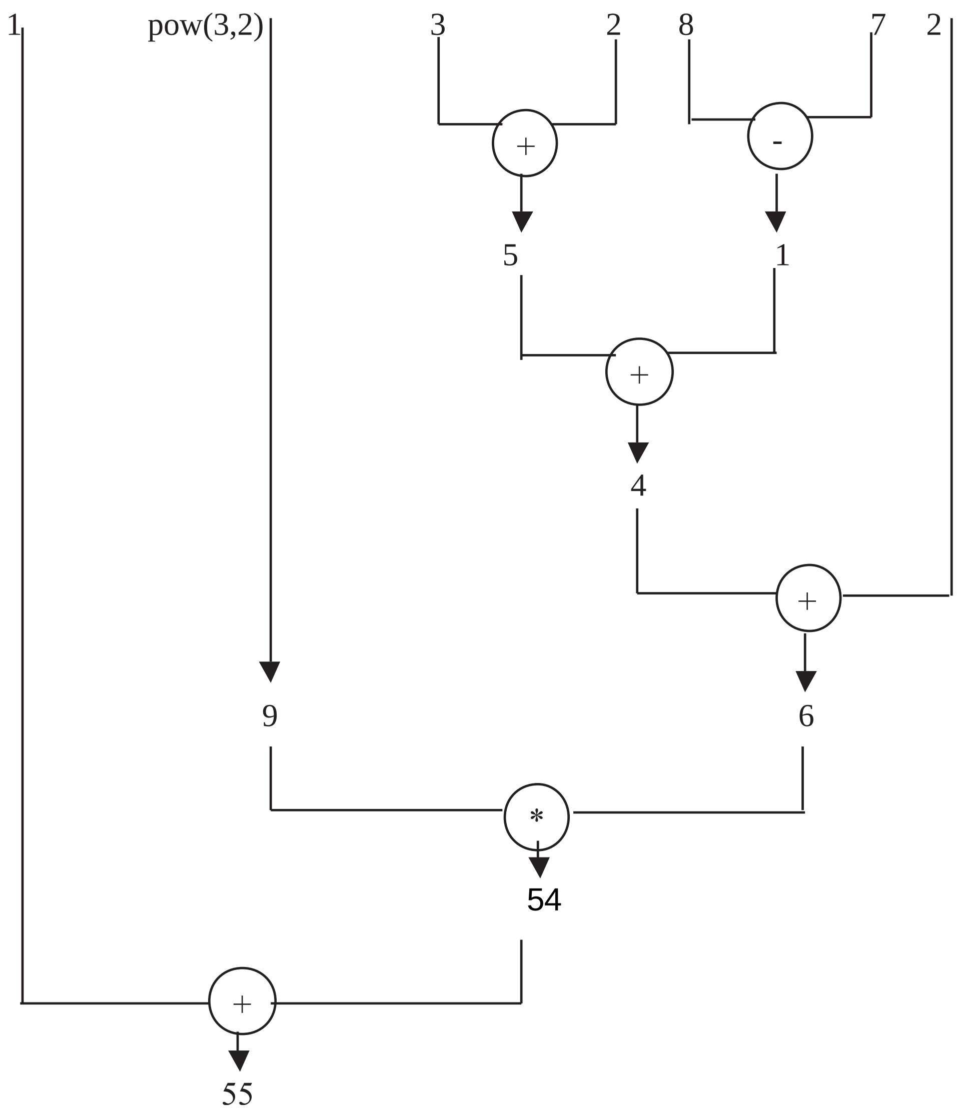


The result is 3, as all the inputs are of integer type.
The result will be 3.66 if any of the inputs are of float type.

1 + pow (3 , 2)

pow(3 , 2)





Increment and Decrement operators are unique to C++. Evaluation of expressions using these operators are indicated in the Table 2.6 .

Expression	Operation	Example

a++	Get the value of a, then increment the value of the variable by 1	a =5;
		c = a++
		Execution :
		c = a;
		a = a+ 1;
		Hence the value stored in the variable c is 5.

++a	Increment the value of the variable a by 1, and then get the value	a = 5;
		c = ++a;
		Execution:
		a=a+1
		c = a;
		Hence the value of c will be 6

a--	Get the value of a , then decrement it by 1	a = 5;
		c = a--;
		Execution :
		c = a;
		a = a –1
		What will be the value of c ?

--a	Decrement the value of a by 1, then get the value of a	a = 5
		c= --a;
		Execution :
		a = a – 1
		c = a;
		What will be the value of c ?

Table 2.6 Increment and Decrement Operator

What will be the values stored in the variables of the following snippets as shown in Table 2.7?

1. a = 5 b = 5 a = a + b++ Value stored in the variable a is _______	2. x = 10 f = 20 c = x++ + ++f Value stored in the variable c is _________ x is __________ f is ___________	3. fun = 1 sim = 2; final = —fun + ++sim – fun - Value stored in the variable fun is __________ sim is __________ final is __________

Table 2.7 Simple Problems

2.2.7.2 Relational Operators

Relational Operators are used to compare values. The list of relational operators are :

- = = equal to
- > greater than
- < lesser than
- >=, <= greater than or equal to , lesser that or equal to
- != not equal to

Relational operators are used to compare numeric values. A relational expression is constructed using any two operands connected by a relational operator. For example the relational operators are used to construct conditions such as

- 10 > 20
- 500.45 <= 1005
- 99 != 99.5
- 9 = = 9

The result of a relational operation is returned as true or false. The numeric constant zero (0) represents False value, and any non-zero constant represents true value. The above expressions output will be

- 0 (10 > 20 is false) ;
- 1 (500.45 < 1005 is evaluated to True hence any non zero constant) ;
- 1 (99 != 99.5 will be evaluated to True hence non zero constant)
- 1 (9 = = 9 will be evaluated to true, hence the output will be non zero constant)

What will be the value of the following expression ?

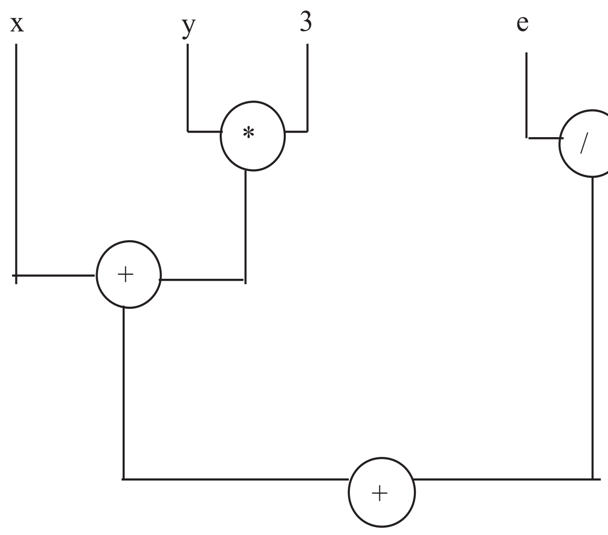
```
( num1 + num2 – num3 )/5 * 2 < ( num1 % 10)
where num1 = 99 , num2 = 20, num3 = 10
```

Evaluate the relational expressions shown in the following Table 2.8.

Operator	**Expression**	**Result**
= =	5 = = 6	0
! =	‘a’ = = ‘a’	
>	5 > 6	
	‘a’ > ‘A’	
<	5 < 6	
	‘a’ < ‘A’	
>=	‘a’ >= ‘z’	
	5 >= 5	
<=	‘a’ <= ‘z’	
	5 <= 6	

Table 2.8 Evaluate Relational Expressions

Relational operators have lower precedence than the arithmetic operators. For example the expression **x + y * z < e / f** will be evaluated as follows :



2.2.7.3.Logical Operators (Boolean Operators)

Logical operators combines the results of one or more conditions. The various logical operators are && (AND) , || (OR) , ! (NOT)

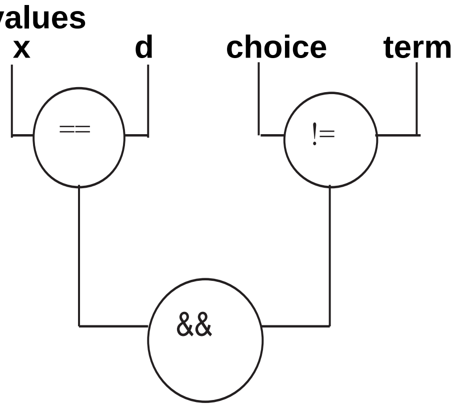
Example : c = 5 , d = 6 , choice = ‘y’, term = ‘2’ (Assume True is indicated as 1 and False as 0)

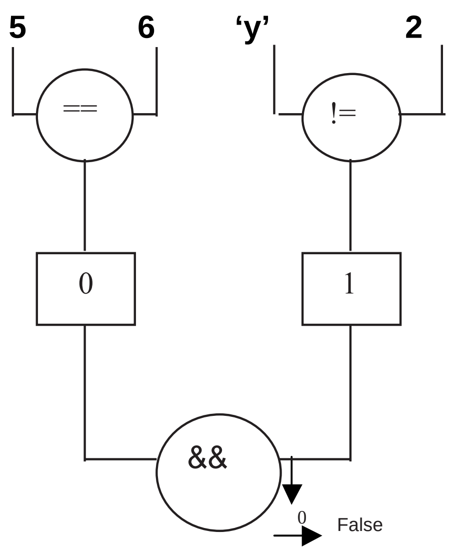
Result_1 = (c = = d) && (choice != term)
Result_2 = (‘y’ = = ‘Y’) || (term != ‘0’)

Result_3 = (c = = d) && ‘(y’ = = ‘Y’)|| choice != term
Result_4 = (c = = d) || (‘y’ = = ‘Y’) && choice ! = term

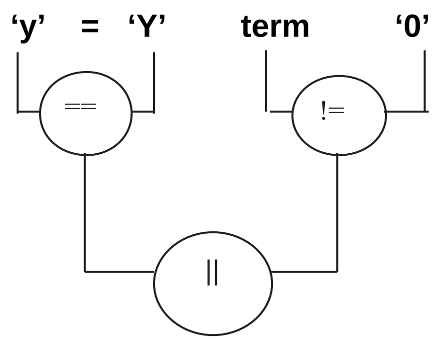
What will be the values stored in Result_1 and Result _2 ??
The values stored in Result_1 is 0 (False) ; Result_2 is 1 (True) , Result_3 is 1 (True) and Result_4 is 0 (False).

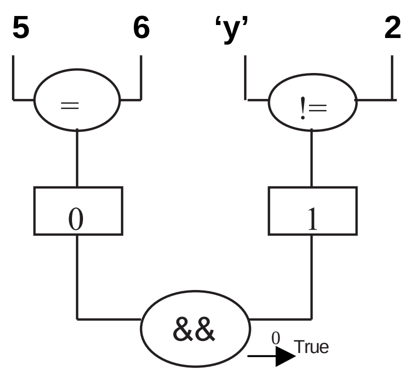
Result_1= (c==d) && (choice != term) By substituting values



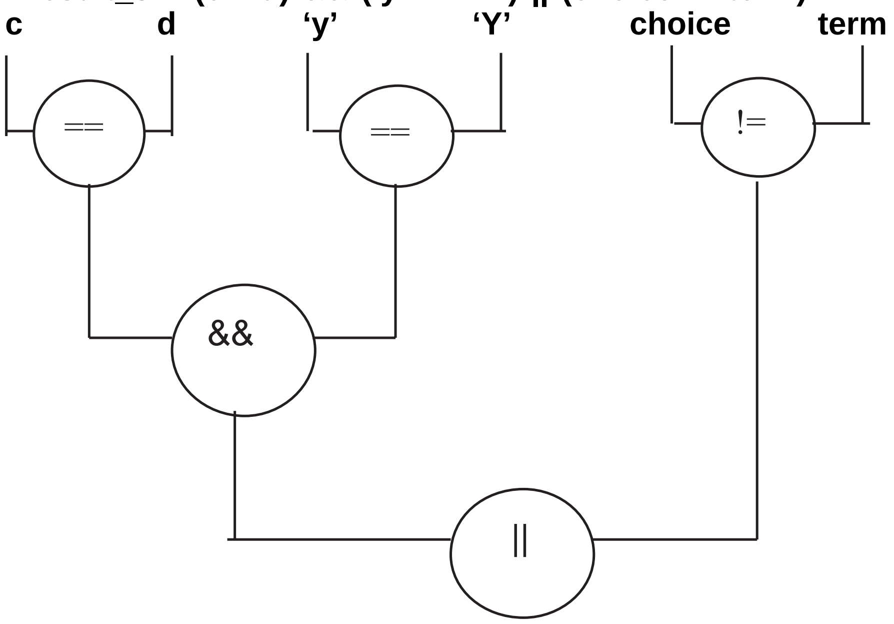


Result_2 = ('y'=='Y') && (term != 0) By substituting values

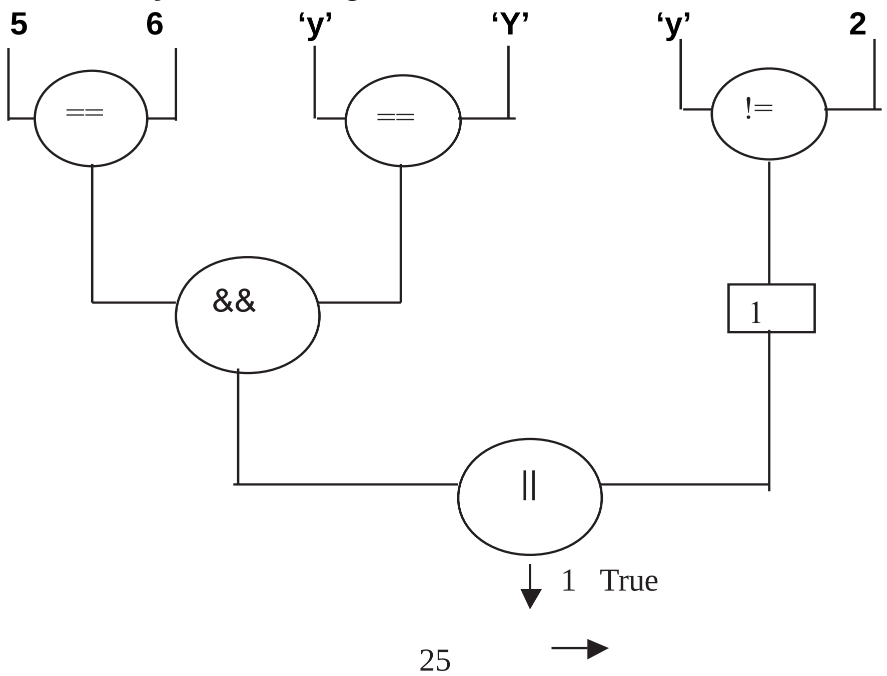




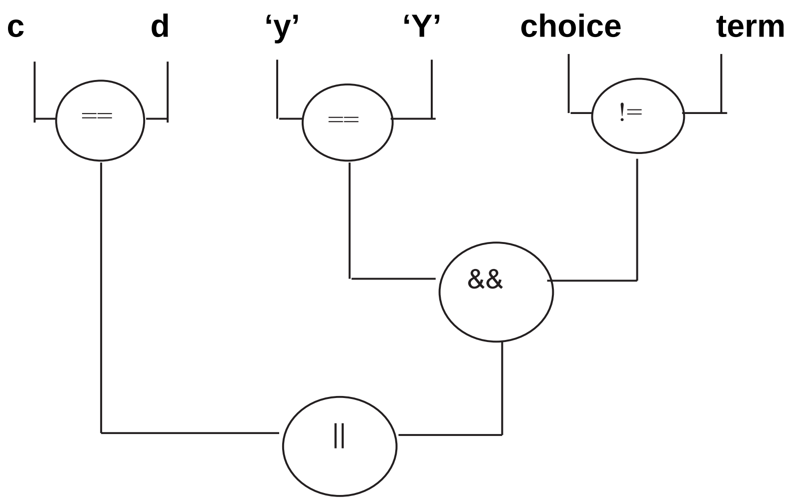
Result_3 = (c==d) && (‘y’ == ‘Y’) || (choice != term)



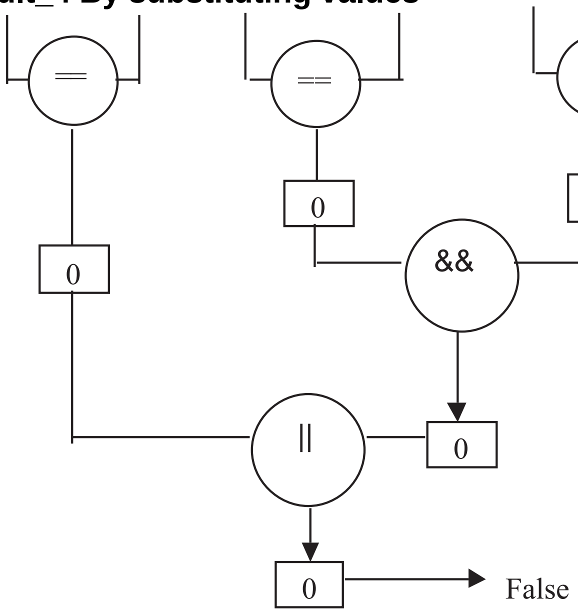
Result_3   By substituting values



Result_4 = (c == d) || ('y' == 'Y') && choice != term

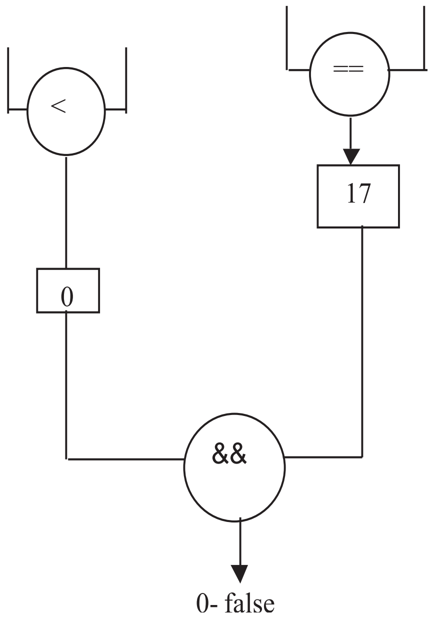


Result_4 By substituting values



The Logical operators have lower precedence to relational and arithmetic operators. Can you evaluate the value of the following expression ?

5 < 4 && 8 + 9



2.2.7.4. Conditional Operator (?:)

(num1 > num2) ? “true”:”else” - ?: Is a ternary operator – (num1>num2,”true”,”false” are the operands. A ternary operator (?:) is also called as conditional operator. The general syntax is E1 ? E2 : E3 where E1,E2,E3 are operands. E1 should essentially be of scalar type, E2 and E3 are values or statements. For example to assign the maximum value of the two values one can express it as :

> **max = (num1 > num2) ?** num1 : num2; The variable **max** will take the value of num1 if num1 is greater than num2, otherwise max will be assigned with the value of num2.

Can you write out what will be the value stored in **x** of the following snippet ?

```
a = 10
b = 10
x = ( a < b) ? a*a : b % a;
```

2.2.7.5. Assignment Operators

= is the simple assignment operator. It is used to assign the result of an expression (on the right hand side) to the variable (on the left hand side of the operator). In addition to the simple assignment operator, there are 10 'shorthand' assignment operators . Refer to the Table 2.9 for all assignment operators.

Expression	**Working**	**Result**
A = 5	The value 5 is assigned to the variable A.	The variable takes the value 5.
A += 2	A+= 2 is interpreted as A = A +2	The value stored in Ais 7
A *= 4	A = A * 4	The value stored in A is 20
A /= 2	A = A / 2	The value stored in A is 2
A - = 2	A = A – 2	The value stored in A is 3
A %= 2	A = A % 2	The value stored in A is 1
Evaluate the following expressions where a = 5, b = 6, c = 7		

A += b*c
 	
C *= a + a / b
 	
B += a % 2 * c		

Table 2.9 Assignment Operators

Table 2.10 gives the complete Operator precedence of all operators used in C++

Operator Precedence	Type	Associativity
() []		
Postfix ++, — , prefix ++, — ! –logical not + unary , - unary	Mathematical-Unary Logical – unarymathematical	Left to right Left to right Right to left Right to left Right to left Left to right
* / %	Mathematical –binary	Left to right
+ -	Mathematical– binary	Left to right
< <= > >=	Relational-binary	Left to right
= = !=	Relational-binary	Left to right
&& (AND)	Logical – binary	Left to right
|| (OR)	Logical – binary	Left to right
?:	Logical – ternary	Left to right
= *= /= %= += -= <<= >>= &= ^= |=	Assignment	Right to left

Table 2.10 Operator Precedence

(note : operators specific to C++ will be dealt with in their relevant topics)

2.2.8 Punctuators

Punctuators are characters with a specific function**.** Refer to the Table 2.11 for Punctuators and their Purpose.

Punctuators	**Purpose**
;	Terminates a C++ statement
//	Treats statements prefixed with this as comments
/* */	Blocks enclosed within these characters are treated as comment
{ }	Used to group a set of c++ statements. Coding for a function is also enclosed within these symbols
[ ]	Index value for an element in an array is indicated within these brackets
‘ ’	Is used to enclose a single character
“ “	Is used to enclose a set of characters

Table 2.11 Punctuators and their Purpose

2.3 Data Types

Data Types are the kind of data that variables hold in a programming language. The ability to divide data into different types in C++ enables one to work with complex objects. Data is grouped into different categories for the following two reasons :

- The compiler may use the proper internal representation for each data type

- The programmer designing the programs may use appropriate operators for each data type. They can be broadly classified into the following three categories.
  - User defined type
  - Built-in type
  - Derived type

The broader classification is indicated in the Fig. 2.2

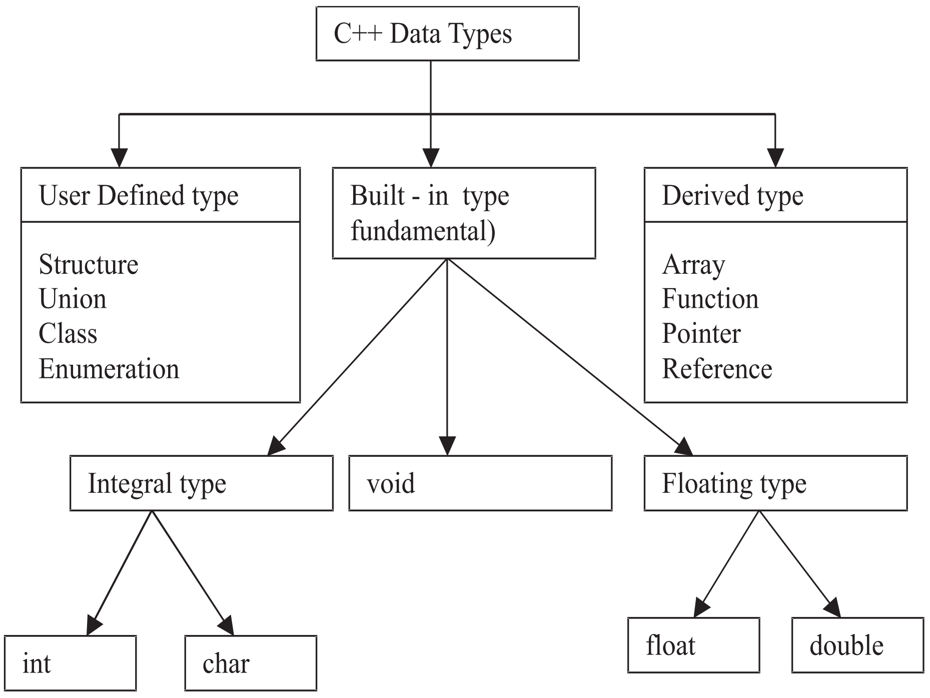
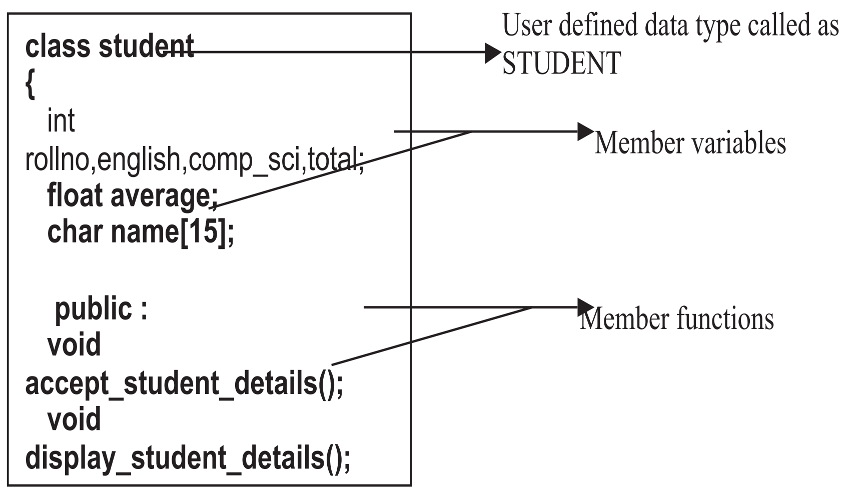


Fig. 2.2 C++ Data Types

2.3.1 User Defined Data Type

User Defined Data Type enables a programmer to invent his/her own data type and define values it can assume. This helps in improving readability of the program.

For example consider the following user defined data type



```
class student
{
   int
rollno,english,comp_sci,total;
   float average;
   char name[15];

    public :
   void
accept_student_details();
   void
display_student_details();
```

Fig. 2.3 User Defined Data Type

student is a user defined data type of **class**. This data type defines the features of a student in terms of member variables, and the associated functions like accepting data for a student, displaying details, and also calculating their respective totals and averages. Thus the **class student improves the credibility and readability of the program by combining** the data requirements and its associated functions in the form of a data type for a student.

Users can define a variable that would represent an existing data type. "**Type definition**" allow users to define such user defined data type identifier. The syntax :

typedef data_type user_defined_data_type_identifier;

For example:
typedef int **marks**;
typedef char **grade**;

The data type identifiers **marks** and **grade** are user defined identifiers for int and char respectively. Users can define variables of int and char as follows:

marks eng_marks, math_marks;
grade eng_grade, math_grade ;

typedef helps in creating meaningful data type identifiers, that would increase the readability of the program.

Another user defined data type is they enumerated data type. As the name suggests, enumerated data type helps users in creating a list of identifiers, also called as symbolic numeric constants of the type int.

The syntax :
enum data type identifier (value 1, value 2, … value n);

Examples :
enum **working_days** (Monday, Tuesday, Wednesday, Thursday, Friday);
enum **holidays** (Sunday, Saturday);

The identifiers **working_days** , **holidays** are user defined data type. Monday, Tuesday … is the list of values also called as **enumeration constants** or **numeric constants.**

Users can declare variables of this enumerated data type using the syntax :

enum identifier variable1, variable2 …,variable n;

For example the variables first_workingday and last_workingday of the type working_days may be declared as follows:

working_days **first_workingday**, **last_workingday**;

These variables can take only one of the values defined for working_days.

```
   first_workingday = Monday ;
   last_workingday = Friday;
```

The enumeration constants (Monday, Tuesday, Wednesday…) are given integer constants starting with 0 (zero) by the compiler. The above assignment statements can also be rewritten as:

```
   first_workingday = 0 ;
   last_workingday = 4;
```

Users can also redefine these integer constants by assigning explicit values to the enumeration constants as

enum **working_days** (Monday = 1, Tuesday, Wednesday, Thursday, Friday);

Here, the constant Monday is assigned the value of 1. The remaining constants are assigned successive integer constants.

2.3.2. Storage Class

Storage Class is another qualifier (like long or unsigned) that can be added to a variable declaration. The four storage specifiers are **auto**, **static**, **extern** and **register**. static and register variables are automatically intialized to zero when they are declared. Auto variables are not initailized with appropriate values based on their data type. These variables get undefined values known as garbage. The following Table 2-12 gives the meaning and relevant examples.

Storage	Meaning	Example

auto	Defines local variable known to the block in which they are defined. By default the local variables are auto hence rarely used.	void main()
		{
		autofloat ratio;
		int kount;
		}
		
		The variables ratio and kount defined within the function main() have the storage specifier as auto.

static	Variables defined within a function or a block cease to exist , the moment the function or the block looses its scope. **Static** modifier allows the variable to exist in the memory of the computer, even if its function or block within which it is declared looses its scope. Hence the variable also retains the last assigned value.	void fun(){
		static int x;
		x++;
		}

extern	Global variable known to all functions in the current program. These variables are defined in another program.	extern int filemode;extern void factorial();

register	The modifier register instructs the compiler to store the variable in the CPU register to optimize access.	void fun(){
		register int I;}

Table 2.12 Storage Classes

2.3.4 Built in Data Types

Built in Data Types are also called as Fundamental or Basic data types. They are predefined in the compiler. Integral, Float and Void are the three fundamental data types.

Integral type is further divided into **int** and **char.** Int is the Integer data type. It cannot hold fractional values. **char** is character data type that can hold both the character data and the integer data. For example consider the declaration and initialization of the variable **ch -** char ch = ‘A’. The statement char ch = 65 would also yield the same result of storing the value ‘A’ in the variable ch as character data type can hold both character and integer values.

Floating type is further divided into **float** and **double.** Floating type can store values with fractional part (Refer to floating point constants representation)

Void type has two important purposes :

- To indicate the a function does not return a value
- To declare a generic pointer

For example consider the following functions defined in C++ (Program void.cpp & fun.cpp).

```
Program  void.cpp
#include<iostream.h>
#include<conio.h>
void  fun(void)
{
  int a,b;
  cin >> a >> b;
  cout << a+b;
}

void main()
{
  fun();
}
```

```
Program  fun.cpp
#include<iostream.h>
#include<conio.h>
int fun(int a, int b)
{
  return  a+b;
}
void main()
{
   int a = 5 , b = 6, sum = 0;
   sum =  fun(a,b);
   cout << sum;
}
```

In the example void.cpp the prototype void fun(void) indicates that the function does not return any value, nor does it receives values(in the form of parameters). Hence the call statement in the main() function is given as **'fun()'** . In the example fun.cpp, the prototype **int fun(int a, int b)** , instructs the compiler that the function fun() returns an integer value. Hence the call statement in the main() function is given as '**sum = fun(a,b)'** The variable sum receives the value from the return statement (return a+b)

✓ **void** data type indicates the compiler that the function does not return a value, or in a larger context void indicates that it holds nothing.

Basic data types have several modifiers. These modifiers have a profound effect in the internal representation of data. signed, unsigned, long and short are some of the modifiers. Table 2.13 gives a list of the data types, memory allocation and range of values.

2.3.4. Derived Data Type

These are built from the basic integer and floating type (built in type) or user defined data types. For example

```
int num_array[5]      =   {1,2,3,4,5};
chardayname[7][3]     =  {Sun","Mon","Tue","Wed","Thu",
                           "Fri","Sat"};
```

num_array stores 5 values. Each element is accessed using the positional value of the element in the array. The position numbering commences from zero. num_array[0] stores value 1 and num_array[4] stores value 5.

Can you write as to what is stored in dayname[0],dayname[5] and dayname[3][2] ?

Type	**Byte**	**Range**
char 1	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int, unsigned short int	2	0 to 65535
signed int,short int, signed short int	2	-32768 to 32767
long int,signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float 4	4	3.4e-38 to 3.4e+38
double	8	1.7e-308 to 1.7e+308
long double	10	3.4e-4932 to 1.1e+4932

Table 2.13 Data Types Size & Range of Values

2.3.4.1 Pointers

A pointer is a variable that holds a memory address. Pointers provide the means through which the memory locations of a variable can be directly accessed. Every byte in the computer's memory has an address. Addresses are numbers just as our house numbers. The address number starts at NULL and goes up from there.. 1, 2 , 3…..

For example a memory size of 640 KB will have addresses commencing from NULL and goes up to 655, 358 as shown in Fig. 2.4.

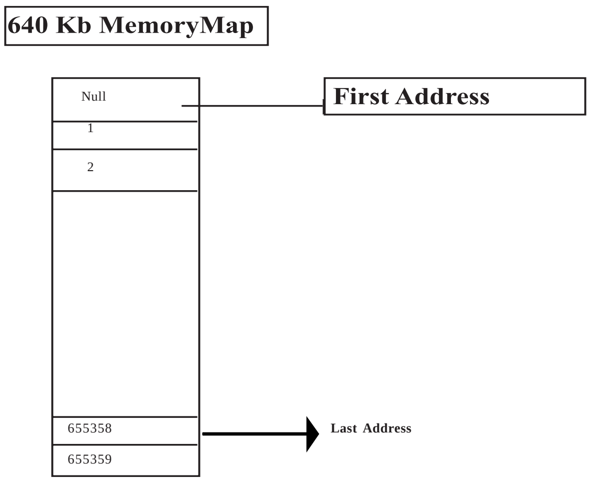
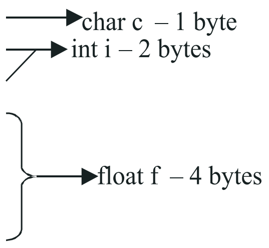


Fig. 2.4 640 Kb Memory Map

When a program is compiled, some memory is allocated to the variables by the compiler. The amount of memory allocated to each variable depends on the data type of the variable.

For example consider the declarations :
char c ; int i; float f;



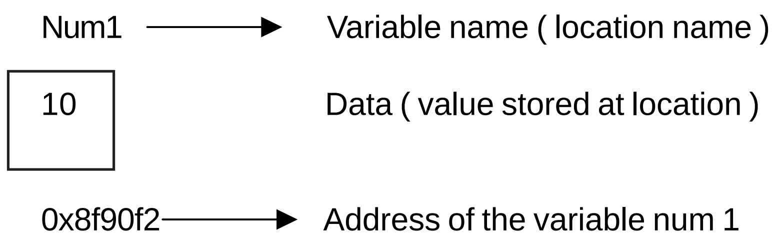
Every variable will be referred by its address. From our example the address of the variables c,i and f will be 1,2 and 4 respectively. Addressing is done using the hexadecimal system.

When dealing with pointer data type one needs to know about the **address of (&)** operator and the **value at operator (*).**

The ' &' operator : When we type int num1=10;

the C++ compiler performs the following operations / actions :zz

1) Reserves space in the memory to hold the integer value
2) Associates the variable name num1 with a memory location
3) Stores the value 2 at this location in the memory


Num1
Variable name (location name)
10
Data (value stored at location)
0x8f90f2
Address of the variable num 1

```
// Program – 2.1
// to demonstrate use of & operator
#include<iostream.h>
#include<conio.h>
void main()
{     clrscr();
      int i = 10;

cout << "\n Address of the variable... " <<&i;
cout << "\nValue stored in the variable .." << i;

      getch();
}
```

Now consider this

```
int *x , num1;
num1=10;
x=&num1;
cout<<*x;
```

Note :

The asterix (*) is

1) Used to declare a pointer variable
2) Used to display the contents stored at a location (value at the address operator)
3) It is a unary operator

2.4 Variables

The name assigned to a data field that can assume any of a given set of values is defined as the variable. For example consider the following group of statements

```
int num;
num  =  5;
```

The statement **int num**; may be interpreted as " num is a variable of the type integer ". The assignment statement **num = 5** may be interpreted as the value 5 is stored in the variable num.

Variables are user defined named entities of memory locations that can store data.

Variable names may contain letters, numbers and the underscore character(_). Names must begin with a letter or underscore. (However names beginning with an underscore are reserved for internal system variables). Names are case sensitive, which means that it differentiates between lower case and upper case letters.

Complete the Table – 2.14.

Variable	**Valid/Invalid**	**Reasons if invalid**
A_b	Valid	
1a_b	Invalid	Variables must begin with an alphabet or an underscore only.
_Test		
Balance$		
#One		
Include		

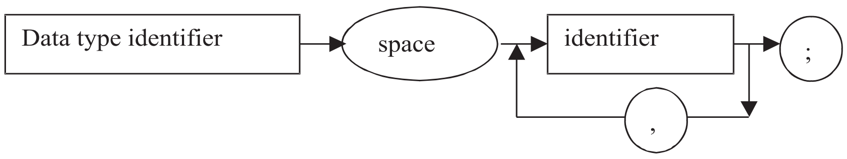
Table 2.14 Validity of Variable Names

2.4.1 Declaration of Variables

Variables are allocated memory to store data. Compiler allocates memory, based on the data type of the variable. Hence variables must be declared before they are used.

Example : int a;
float f1,f2;
char name[10],choice;

Syntax :



Consider the declaration **int side, float hypotenuse , area ;** This is an erroneous declaration because the compiler interprets this statement as follows :

- The variables side, float, hypotenuse and area will be treated as instances of the data type **int**. Hence it throws an error stating that " **comma is expected after float**"
- The intention was to declare the variable side of int data type and the vairbales hypotenuse & area of the data type float.
- Hence the above declaration statement should be rewritten as follows :

```
int side ;
float hypotenuse , area ;
```

```
int side ; float hypotenuse,area ;
```

✓ **More than one variable of the same data type can be declared in a single declaration statement. But every variable should be separated by comma.**

There are nine words for data types such as **char , int , double , float, void, short, signed, long and unsigned .**

long, short, signed and unsigned are qualifiers or modifiers that modify a built – in data type with the exception of void.

The internal representation for the integer value 15 is **0**0000000 00001111 . Integer values are stored in 16 bit format in binary form. Starting from right extreme, 15 bits are used to store data. Maximum value stored in an integer variable is +32767 and the minimum value is –32768, as 2^{15} **is 32767 on the positive side and –32768 on the negative side. 16th bit, also called as the Most Signigicant Bit or sign bit. It is used to store sign. The 16th bit will have a value 1 if**

negative value is stored. When the modifier unsigned is used the integer data type will store only positive values, the sign bit is also used to store data. Therefore the range to store data goes upto 2^{16} , hence the maximum value will be 65535.

✓ **The modifier alters the base data type to yield new data type.**

The impact of modifiers :

- **unsigned** modifies the range of the integer values as the sign bit is also used to store data.
- **long** increases the bytes for a particular data type, thus increasing the range of values.

The base data type should be prefixed with the modifiers at the time of declaring a variable. For example :

unsigned int registration_number;
long unsigned int index;
short signed char c;

✓ **Prefix the data type with modifiers at the time of declaring variables.**

The **const** qualifier specifies that the value of a variable will not change during the run time of a program. Any attempt to alter the value of a variable defined with this qualifier will throw an error message by the compiler. The const qualifier is used like any other modifier where the variable is prefixed with the keyword const followed by data type .

For example :

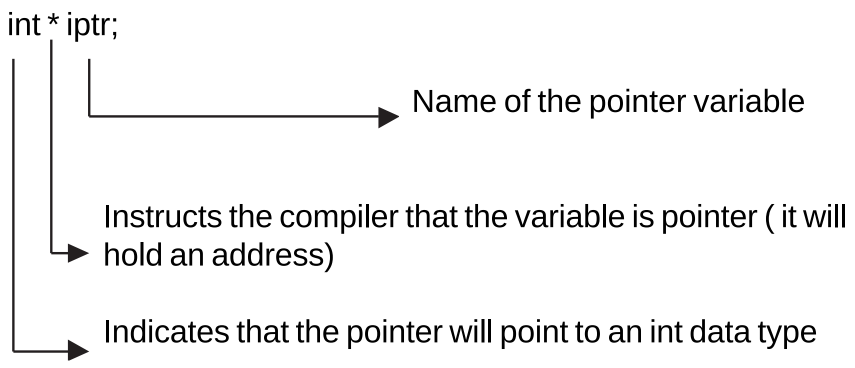
const float pi = 3.14;

The Table 2.15 shows the **data types** with altered lengths and range of values when the qualifiers or modifiers are used

Data Types		
Type	Length	Range
unsigned char	8 bits	0 to 255
char	8 bits	-128 to 127
enum	16 bits	-32,768 to 32,767
unsigned int	16 bits	0 to 65,535
short int	16 bits	-32,768 to 32,767
int	16 bits	-32,768 to 32,767
unsigned long	32 bits	0 to 4,294,967,295
long	32 bits	-2,147,483,648 to 2,147,483,647
float	32 bits	3.4 *(10**-38)to 3.4 * (10**+38)
double	64 bits	1.7 *(10**-308)to 1.7*(10**+308)
long double	80 bits	3.4 * (10**-4932) to 1.1 * (10**+4932)

Table 2.15 Data Types with Modifiers

Declaring pointer variables


int * iptr;
Name of the pointer variable
Instructs the compiler that the variable is pointer (it will hold an address)
Indicates that the pointer will point to an int data type

The declaration statement int *ptr may be read as ptr is a pointer variable of the type int. The variable ptr can only store addresses that hold integer values.

Examples of pointer variable declarations:

char * cptr float * fptr	declaring a pointer to character type a pointer to float type
void *v ptr	a pointer that can point to any data type a generic pointer is declared in this way
const int * ptr	ptr is a pointer to a constant integer (cannot modify the value stored at the address pointed by ptr)
char * const cp	cp is a constant pointer. The address stored in cp cannot be modified

Table 2.16 Examples of Pointer Variables

2.4.2 Initialization of variables

Variables are initialized to a specific value at the time of declaration. Initialization is done only once. For example :

```
int  num = 10;
int fun(5)
```

In statement (1) the variable num is initialized to 10, whereas in the second statement the variable fun is initialized to 5 through a constructor.

Implicit conversions: refers to data type changes brought about in expressions by the compiler. For example consider the following snippet:

```
float f = 7.6;
int  x = f;
```

The value stored in the variable x is 7, as float is converted to int. The compiler does this conversion automatically.

Rules for implicit conversion :

Consider a term, having a pair of operands and an operator. The conversions takes place as follows :

1. If one operand is of type **long double** , then the other value is also converted to long double.
2. If one operand is of type **double,** then the other value is also converted to double.
3. If one of the operands is a **float**, the other is converted to a float.
4. If one of the operands is an **unsigned long int**, the other is converted to unsigned long int.
5. If one of the operands is a **long int**, then the other is converted to long int.
6. If one of the operands is an **unsigned int**, then the other is converted to an unsigned int.

```
// demonstrating implicit type conversions
        // Program – 2.2
# include <iostream.h>
# include <conio.h>
# include <iomanip.h>

void main()
{
 clrscr();

 int i;
 float f;
 double d;
 long double ld;
```

```
  unsigned int ui;
  unsigned long int uli;
  i = -5;
  f = 2;
  d = 3;
  ld = 3;
  ui = 6;
  uli = 4;
cout <<"\nSizeof long double.."<<sizeof(ld*d)<<'\t'<<ld*d;
cout << "\nSizeof double..." << sizeof(d*f)<<'\t'<<d*f;
cout << "\nSizeof float..." << sizeof(f * i)<<'\t' << f*i;
cout << "\nSizeof unsigned long int ..."
     << sizeof(uli* f)<<'\t'<< uli * f;
cout << "\nSizeof unsigned int..." << sizeof(ui * i)
     <<'\t'<< ui * i;
getch();
}
```

Note : **sizeof** is an operator . It returns the size (memory requirement) in terms of bytes, of the given expression or data type.

```
Output displayed by the above program:
Sizeof long double       ...10  9
Sizeof double            ...8   6
Sizeof float             ...4   -10
Sizeof unsigned long int ...4   8
Sizeof unsigned int      ...2   65506
```

Complete the following Table 2.17 based on the sample program 2.2 , write answers as shown in the reason column for the first value.

Sno.	Size of the result - in terms of bytes	Expression	Reason
1.	10	ld*d	The value generated is of long double type as the variable ld is long double. As long double data type requires 10 bytes to store a value, 10 is displayed.
2.	8	d*f	The value generated is of double type. …….
3.	4	f*l	
4.	4	uli * f	
5.	2	ui * i	

Table 2.17 Exercise based on Program 2.2

Initialization of pointer variables

Pointer variables can store the address of other variables. But the addresses stored in pointer variables are not of the same data type as this pointer variable is pointing to. For example :

```
int *iptr, num1;
num1 = 10;
iptr = &num1; // initializing a pointer variable
```

The following initalization is invalid.

```
int *iptr;
float num1 = 10.5;
iptr = &num1 // initializing  pointer variable pointing to integer
                     data type with the address of float variable would
                      throw an error.
```

Pointer variables are sensitive to the data type they point to.

Type cast : Type cast refers to the process of changing the data type of the value stored in a variable . The statement **(float) 7** , converts the numeric constant 7 to float type. Type cast is achieved by prefixing the variable or value with the required data type. The syntax is : (data type) <varaible/value> or data type (variable/constant) . Type cast is restricted only to fundamental or standard data types. The statement **x = 8 % 7.7** will throw an error on compilation as, modulus operator % operates on integer data type only. This erroneous statement can be corrected as x = 8 % **(int) 7.7** - the float constant 7.7 is converted to integer constant by type casting it.

Complete the following Table 2.18

int x;
x =7 / 3;
	What is the value stored in X?

float x;
x = 7 / 3;
	What is the value stored in X?

float x;
x = 7.0 / 3.0;
	What is the value stored in X?

float x;
x = (float) 7 / 3;
	What is the value stored in X?

float x;
int a = 7 , b = 3;
x = a/b;
	What is the value stored in X?

float x;
int a = 7 , b = 3;
x = a/ (float) b;
	What is the value stored in X?

Table 2.18 Find the value of X

Exercises

1. Determine the order of evaluation of the following expressions :

   i. a + pow(b,c) * 2
   ii. a || b && c
   iii. a<b && c || d > a
   iv. (c>=50)||(!flag)&&(b+5 == 70)
   v. (a+b)/(a-b)
   vi. (b*b) – 4 * a * c

2. Identify errors in the following programs ..

a.
```
# include <iostream.h>
void main()
{
 float f = 10.0;
 x = 50;
 cout << x << f;
}
```

b.
```
 # include <iostream.h>
 void main()
{
 float f = 10.0;
 x = 50;
 cout << x << f;
}
```

c.
```
# include <iostream.h>
void main()
{
 int x,y,k,l;
 x =  y + k——l;
 cout << x;
}
```

3. Predict the output

```
a.
# include <iostream.h>
# include <conio.h>

void main()
{
  int i=20;
  cout << i << i++ << ++i;
  getch();
}
```

```
b.
# include <iostream.h>
# include <conio.h>
void main()
{
  clrscr();
  int i = 1, a= 3;
  i = a++;
  cout << i;
  getch();
}
```

```
c.
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int i = 3,x;
  x = i ? i++ : ++i;
  cout << x;
  getch();
}
```

```
d.
# include <iostream.h>
# include <conio.h>
void main()
{
  int z,x = 3, y = 2;
  z = --x + y++;
  cout << z;
  getch();
}
```

```
e.
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  char ch = 'a';
  ch = (ch == 'b') ? ch :' b';
  cout << ch;
  getch();
}
```

4. Evaluate the following C++ expressions

Assume a = 5, b =3, d =1.5, c is integer and f is float.

a. f = a+b/a

b. c = d * a+b

c. x = a++ * d + a;

d. y = a – b++ * —b;

e. (x >= y) ||(!(z==y) && (z < x)) where

- ✓ x = 10, y = 5, z = 11 (all are integers)
- ✓ x = 10, y = 10, z = 10
- ✓ x = 9, y = 10, z = 2

5. Write the C++ equivalent expressions using the conditional operator. Where

- ✓ f = 0.5 if x = 30, otherwise f = 5
- ✓ f n = 0.9 if x >= 60, otherwise .7

6. What are pointer variables ?

7. Write a declarative statement to declare ‘name‘ as a pointer variable that stores the address pointing to character data type.

CHAPTER 3

BASIC STATEMENTS

Basic Statements in C++ are constructed using tokens. The different statements are

- Input/output
- Declaration
- Assignment
- Control structures
- Function call
- Object message
- Return

3.1 Input/output statements

Input /Output statements such as reading data, processing data and displaying information are the essential functions of any computer program. There are two methods for assigning data to the variables. One method is by assignment statement which we have already seen in the earlier section, and the other method is to read data during the runtime of a program. Data is read from the keyboard during runtime by using the object **cin (**pronounced as C in**). cin** is a predefined object that corresponds to a standard input stream. Input stream represents the flow of data from the standard input device – the keyboard. cin can read data from other sources also which will be dealt later. The declarations for the object cin are available in a **header file** called as **istream.h** The basic input/output operations are managed by a set of declarations available in the istream.h and ostream.h header files. Iostream.h file comprises the combined properties of istream and ostream.

- A header file comprises of all standard declarations and definitions for predefined functions.
- One can include the header file in the program by using a preprocessor directive
- A preprocessor directive starts with # , which instructs the compiler to do the required job.
- # include <iostream.h> is a typical preprocessor directive, that instructs the compiler to include the header file iostream.h In order to use cin / cout objects one has to include iostream.h in the program.
- The other header files are iomanip.h, stdio.h, ctype.h, math.h, fstream.h etc.

The >> is the extraction or get from operator. It takes the value from the stream object to its left and places it in the variable to its right. For example consider the following snippet :

```
float temperature;
cin >> temperature;
```

The extraction operator (>>) extracts data from the input stream object (cin) and places the value in the variable(temperature) to its right. Multiple values can be read from the input stream and placed in the corresponding variables, by cascading the extraction operator. For example, to read the values for temperature and humidity one can perform it as follows :

```
cin >> temperature >> humidity;
```

cout pronounced as (C out) is a predefined object of standard output stream. The standard output stream normally flows to the screen display – although it can be redirected to several other output devices. The operator **<<** is called the insertion operator or put to operator. It directs the contents of the variable to its right to the object to its left. For example consider the following statements;

```
int marks = 85;
cout << marks;
cout << “\n Marks   obtained is : “ << marks;
```

The value stored in marks is directed to the object cout, thus displaying the marks on the screen.

The second statement **cout << “\n Marks   obtained is : “ << marks;** directs both the message and the value stored in the variable to the screen. Cascading of insertion operator facilitates sending of multiple output via a single statement.

Examples :

```
cout << “\n The sum of the variables a,b ..” << a+b;
cout << a+b << ‘\t’ << a-b << ‘\t’ << a/b;
cout << “\n The difference of numbers ….” << a-b
    << “\n The sum of two numbers  …. “ << a+b;
```

3.2 My first C++ program - Structure of a C++ Program

```
// My first program – Program 3.1
# include <iostream.h>//preprocessor directive
# include <conio.h>
   float fact = 1; // declaration of variables
   int term;
int main()  // function header
{
   clrscr(); // predefined function
   cout << “\n This program computes factorial of a
number”;
   cout << ‘\n’ << “Enter a number ...”;
   cin >> term;
   for(int x = 2; x <= term;fact *= x,x++);// looping
statement
   cout << “\nThe factorail of the given number ..”
         << term << “  is ..” << fact;
   return 0;
}
```

A C++ program has primarily three sections viz.,

- Include files
- Declaration of variables , data type , user defined functions.
- main() function

On successful compilation, when the program is executed the main() function will be automatically executed. It is from this block, that one needs to give call statements to the various modules that needs to be executed and the other executable statements.

3.3 Declaration Statements

Variables used in the declaration statements need to be declared and defined before they are used in a program.

```
int *iptr;   //  declares a pointer variable to int
iptr = new int;//fetches memory to store data – hence pointer
variable gets defined
*iptr = 5; // stores data 5 only after fetching memory
```

Declaration of a variable introduces a variable's name and its associated data type. For example consider the declaration int *iptr; This statement may be read as iptr is a pointer variable to integer. All pointer variables are defined only when memory is fetched to store data .

Declaration statements are used to declare user defined data type identifiers, function headers, pointer variables and the like. Recall the declaration of user defined data types dealt in **2.3**

However, if a declaration also sets aside memory for the variable it is called as definition. For example consider the declaration statement - **int num;** This statement is called as definition statement because memory is set aside to store data. Now consider the following snippet :

```
int num;   //  declares and defines an integer variable
num = 5; // The data 5 is stored .
```

Have you noticed , there is no explicit request for memory. That is because memory is set aside at the time of declaring the variable.

- ✓ **Declaration statement introduces a variable name and associates it with a specific data type**
- ✓ **A variable gets defined when memory is set aside .**
- ✓ **Some variables also get defined when they are declared**
- ✓ **Pointer variables get defined only when memory is fetched. For example by using new memory operator**

3.4 Assignment Statements

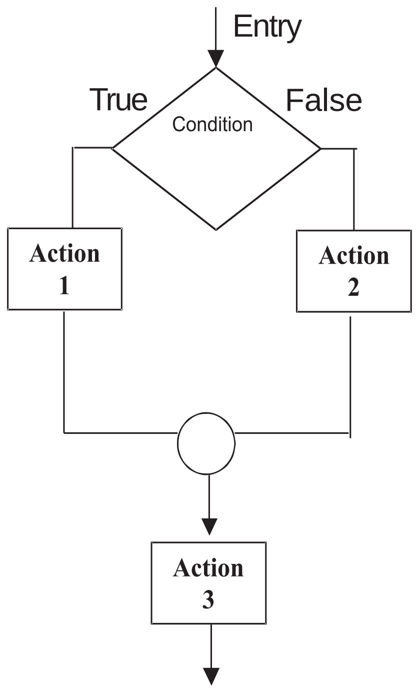
An assignment statement , assigns value on the right hand side of an expression to the variable on the left hand side of the assignment operator. '=' is the assignment operator . For example the different style of assigning values to the variables are as follow :

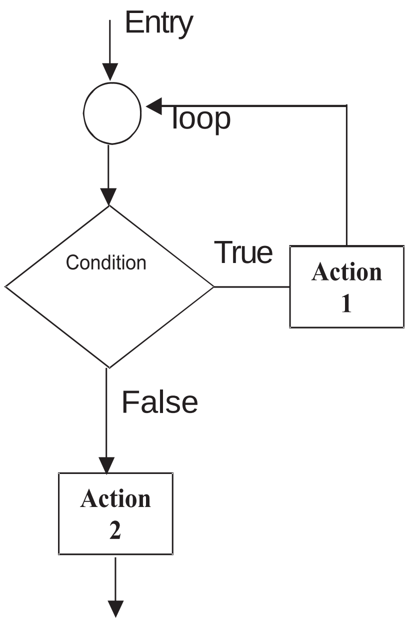
```
num = 5;
total = english+maths;
sum += class_marks;
```

During assignment operation , C++ compiler converts the data type on the right hand side of the expression to the data type of the variable on the left hand side of the expression. Refer to implicit conversions and Type cast of 2.4.2.

3.5 Control Structures

Statements in a program need not necessarily be executed in a sequential order. Some segments in a program are executed based on a condition. In such situations the flow of control jumps from one part of the program to another. Program statements that cause such jumps are called as control statements or control structures. Now look at the following flow charts (Flow chart I & II).





Selection **loop**

1. Trace out the steps to accept an integer and if it is odd add 1 to it. If it is even do nothing. Print the integer as depicted in Flow Chart I. The steps are executed in a sequential manner.

2. Trace out the steps to accept a integer, and print the message "EVEN" /"ODD" based on the divisibility of 2. Here the control

branches to statement " M = ODD" if there is remainder other wise branches to the statement "M = EVEN". This is depicted in Flow Chart 2.

> ✓ **Program statements that cause a jump of control from one part of a program to another are called Control Structures**

The two major categories of control structures are Decision making statements and Looping statements. The control structures are implemented in C++ using control statements as indicated in the following figure Fig. 4.1

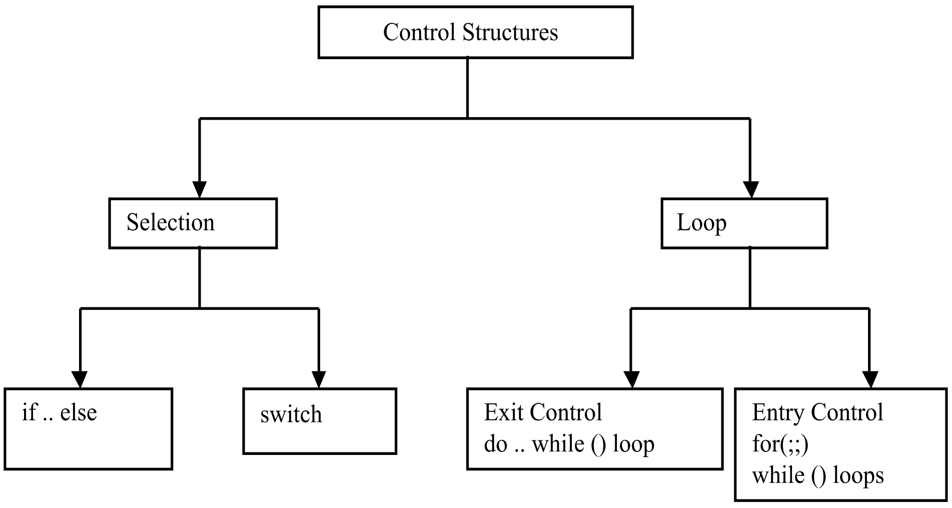


Fig. 4.1 Control Structures in C++

3.5.1 Selection Statements

In a program a decision causes a one time jump to a different part of a program. Decisions in C++ are made in several ways, most importantly with **if .. else …** statement which chooses between two alternatives. Another decision statement , **switch** creates branches for multiple alternatives sections of code, depending on the value of a single variable.

if statement : is the simplest of all the decision statements. It is implemented in two forms

- Simple if statement
- if .. else statement

```
if ( condition / expression)
{
    action block
}
```

```
If (condition / expression )
{
    action block 1
}
else
{
    action block 2
}
```

The following Program - 3.2 demonstrates **if statement** :

Syntax :

```
// Program - 3.2
# include <iostream.h>
# include <conio.h>
// Demonstrates the use and syntax of if statement
void main()
{
  int a;
  clrscr();
  cout << "\nEnter a number ";
  cin >> a;
  if ( a%2 == 0)
      cout << "\nThe given number " << a << "is even";
  getch();
}
```

In the above program the message “The given….” gets printed if the condition is evaluated to true, otherwise the control jumps to getch(); statement directly by passing the statement cout << “\nThe given ….

The following Program -3.3 demonstrates **if .. else ..** statement :

```
// Program – 3.3
// Demonstrates the use and syntax of if else
statement

# include <iostream.h>
# include <conio.h>
void main()
{
int a;
clrscr();
cout << “\nEnter a number “;
cin >> a;
if ( a%2 == 0)
     cout << “\nThe given number “ << a << “is
even”;
else
      cout << “\nThe given number “ << a << “is
odd”;
getch();
}
```

In the above program **“The given number 10 is even”** is printed if the expression is evaluated to true, otherwise statement following else option will be executed.

Examples of if constructs where conditions/expressions are given in different styles :

Condition is expressed using the variable *branch*

```
int branch = 10 > 20;
if  (branch )
{
    action block 1;
}
else
{
     action block 2;
}
```

Condition is expressed as 1, as any positive integer indicates TRUE state

```
if  (1)
{
    action block 1;
}
else
{
     action block 2;
}
```

Expression is used for condition. If the value of the expression is evaluated to > 0 then action block 1 is executed other wise action 2 is executed.

```
if  ( a % 2 )
{
   action block 1;
}
else
{
 action block 2;
}
```

Can you predict as to what will be printed when the following program is executed ?

```
// Program - 3.4
# include <iostream.h>
# include <conio.h>
void main()
{
   int count = 1;
   if (count > 0)
   {
     cout << “\nNegating count ....”;
     count *= -1;
   }
   else
   {
     cout << “\nResetting count ...”;
     count *= 1;
   }
   getch();
}
```

Output displayed will be **Negating count …** Why do you think block associated with else option is not executed since count was multiplied by –1 ?

Answer to this is that , once the true block is executed in an if .. else statement, then the else block will not be executed.

Else block is executed only if True block is not executed.

✓ **if .. else …** statement which chooses between two alternatives , executes the chosen block based on the condition.

The following if constructs are invalid because :

Sno	Invalid construct	Why invalid ?

1.	if a> b
	cout << "True";
		Condition should always be enclosed in a pair of brackets . The correct form is **if (a>b)** cout << "Condition should True";

2.	if (a> b)
	a—; cout<<"\nVariable is decremented";
	else
	a++;
	cout <<
	"Variable is incremented .."
		Error thrown by the compiler is "**Misplaced else**" . If the action block is compound statements, then it should be enclosed in curly braces .

		The correct form is : if (a> b) { a--; cout<<"\nVariable is decremented"; }else { a++; cout << "Variable is incremented .." }
3.	if (a > b); cout << "Greater.. ";else cout << "Lesser ..”;	The semicolon placed after condition nullifies the effect of if statement , the compiler throws an error "**Misplaced else**" .The correct form :if (a > b) cout << "Greater..";else cout << "Lesser ..";

Table 3.1 if construct

Write appropriate if constructs for the tasks mentioned in table **3.2**

Sno	**Task**	**If construct**
1.	Set Grade to ‘A’ if marks are above 90.	
2.	Set Grade to ‘A’ if marks are above 90, otherwise set grade to ‘B’	
3.	Print the message • “Accelerate – traffic to flow “ if speed is less than 30 kmph, • “Moderate – accelerate by 10kmph” if speed is between 31– 40 kmph, other wise • “Good – be careful ..”	

Table 3.2 Using if Constructs

Nested if statement : The statement sequence of if or else may contain another if statement ie., the if .. else statements can be nested within one another as shown below :

```
if  (expression 1)
   if (expression 2)
   {
       action 1;
   }
    else
    {
       action 2;
     }
else
   {
        action 3;
   };
```

In an nested if .. else statement, “**Each else matches with the nearest unmatched preceding if**”

For example

```
if (grade = = 'A')
   if (basic > 5500)
        incentive = basic * 10/100;
   else
         incentive = basic * 5/100;
else
    cout << "Try to attain Grade A";
```

Working of the above example :

- Grade = 'A' and basic == 5501, then incentive gets the value 550.
- Grade = 'A' and basic = = 5000, then incentive gets the value 250.

- Grade <> 'A' – the inner if will not be executed , the outer else will be executed and thus prints "Try to attain Grade A.

Do you think this if construct is equivalent to the above construct ? Write your answers in the Reason it out box.

```
if (grade = = 'A'  && basic > 5500)
        incentive = basic * 10/100;
else if (grade = = 'A' && basic <5501)
         incentive = basic * 5/100;
else
    cout << "Try to attain Grade A";
```

Reason it out ……………

switch Statement : This is a multiple branching statement where, based on a condition, the control is transferred to one of the many possible points.

This is implemented as follows :

```
switch (expression)
{
   case 1 :  action block 1;
               break;
   case 2 :  action block 2;
               break;
    case 3 : action block 3;
               break;
     default :
                action block 4;
}
```

```
switch (remainder)
{
   case  1 : cout << “remanider 1”;
                break;
   case  2 : cout << “remanider 2”;
               break;

     default :
                cout << “Divisible by 3”;
}
```

The following program demonstrates the use of switch statement.

```
// Program - 3.5
// to demonstrate the use of switch statement

# include <iostream.h>
# include <conio.h>

void main()
{
   int a, remainder;
   cout << “\nEnter a number ...”;
   cin >> a;
   remainder = a % 3;
   switch (remainder)
   {
      case 1 : cout << “\nRemainder is one”;
                break;
      case 2 : cout << “\nRemainder is two”;
                break;
      default: cout << “\nThe given number is divisible by 3”;
                break;
   }
   getch();
}
```

The above program displays

- Remainder is two if a = 5 or so
- The given number is divisble by 3, if a = 9 or so

Or in other words the above program checks for divisibility by 3 and prints messages accordingly.

What do you think will be the output of the following program ?

```
// Program - 3.6
// to demonstrate the use of switch statement

# include <iostream.h>
# include <conio.h>

void main()
{
   int rank = 1;
   char name[] = “Shiv”;
   switch (rank)
   {
      case 1 : cout << ‘\n’ << name << “ secured 1st
rank”;
      case 2 : cout << ‘\n’ << name << “ secured 2nd
rank”;
   }
   getch();
}
```

Output displayed will be :

```
Shiv secured 1st rank
Shiv secured 2nd rank
```

Why do you think both the action blocks of case 1 and case 2 are executed ? Compare the action blocks of Program -3 .5 & Program-3.6. What do you think is missing in Program-3.6 ? Yes it is the **break;** statement.

What do we infer ?

Every action block should be terminated with a break statement. Otherwise all action blocks are executed sequentially from the point where the control has been transferred based on the condition.

In the above example(Program- 3. 6), control was transferred to case 1, as Rank is 1, hence action blocks of case 1 and case 2 are executed sequentially.

✓ Include **break;** in action block,
in order to exit from switch statement.

The following switch constructs are invalid because :

1.	**char name[] = "Shiv";**
	switch (name)
	{
	case "Shiv" : cout << '\n'
	<< name << "
	secured 1st rank";
	case "Rama" : cout << '\n'
	<< name << "
	secured 2nd rank";
	}
		Compiler throws an error. "Switch selection expression must be of integral type "which means that switch expression should be evaluated to an integer constant only (char, enum,int)

2.	**float value;**
	switch (value)
	{
	case 1.5 : cout << '\n'
	<< value – 0.5;
	case 2.9 : cout << '\n'
	<< value + 0.1;
	}
		Value is of float type , hence not a valid switch expression.

3.

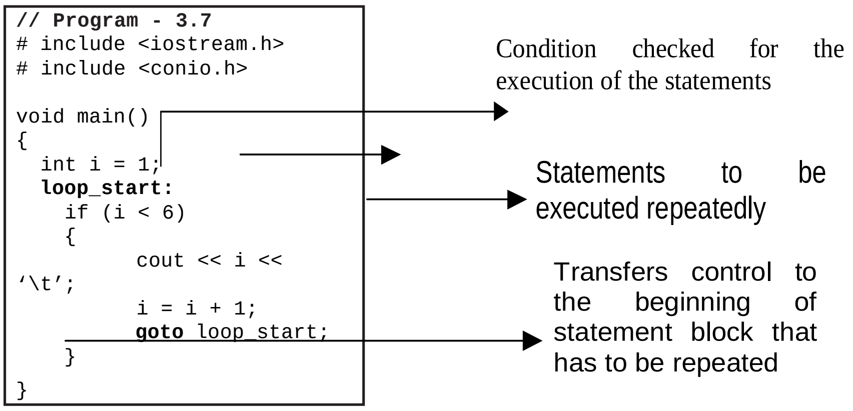
```
switch (rank)
{
case 1 to 2  : cout << '\n'
<< "Best rank";
break;
case 3 to 4 : cout << '\n'
<< "Good rank";
}
```

Case 1 to 2 is an invalid case statement, as case label should have only one integral value. In order to use more than one value for a particular action block one may rewrite the code as :

```
switch (rank)
{case 1 :
  case 2 : cout << "Best
                rank";
                break;
case 3 :
case 4 : cout << "Good
                rank";
                break;
                }
```

3.5.2. Loops

Loops execute a set of instructions repeatedly for a certain number of times. For example consider the following Program – 3.7.



```
// Program - 3.7
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 1;
  loop_start:
    if (i < 6)
    {
          cout << i <<
'\t';
          i = i + 1;
          goto loop_start;
    }
}
```

Condition checked for the execution of the statements
Statements to be executed repeatedly
Transfers control to the beginning of statement block that has to be repeated

The above program on execution will print numbers between 1 and 5, as the action block of **if statement** is executed 5 times.

The Program - 3. 7 works as follows :

1. Declares and initializes the variable i
2. Checks the relational expression i<6
3. If True then executes the action block (cout << i; i = i + 1) and transfers the control back to the loop_start (**goto** loop_start). This enables the program to execute a set of instructions repeatedly, based on the condition of the relational expression. The variable **i** is referred to as the control variable, as the iterations of the block is totally controlled by this variable.

A looping block therefore consists of two segments viz., the body of the loop and the control statement. The control statement checks the condition, based on which directs the control back to the body of the loop to execute the segment repeatedly. Now look at the following snippets.

```
//Program - 3.7 A
void main()
{
  int i = 6;
  loop_start:
    if (i < 6)
    {
        cout << i << '\t';
        i = i + 1;
        goto loop_start;
    }
   cout << i;
}
```

```
//Program - 3.7 B
void main()
{
  int i = 6;
  loop_start:
     {
        cout << i << '\t';
        i = i + 1;
       if (i < 6)
          goto loop_start;
    }
 cout << i;
}
```

What do you think will be the output generated by the above snippets ?

The Program -3.7 A will display 6, where as Program -3.7 B will display 7. Why do you think the loop is executed in Program-3.7 B? In this program the condition is placed after the statements (cout << i; i = i + 1;),hence these statements are executed once, after which the condition is checked. Since the variable i takes the value as 7, the control is not transferred to loop_start. So what do we infer ??

- ✓ **Loops are unconditionally executed at least once, if the condition is placed at the end of the body of the loop**
- ✓ **Based on the position of the condition, the loops are classified as Entry-Check loop (as in Program-3.7 A) and Exit Check Loop (as in Program-3.7 B)**

In general, a looping process would work in the following manner :

1. Initializes the condition variable
2. Executes the segment of the body
3. Increments the value of the condition variable as required
4. Tests the condition variable in the form of a relational expression. Based on the value of the relational expression the control is either transferred to the beginning of the block, or it quits the loop.

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

do .. while Loop : The construct of a do .. while loop is :

```
do
{
action block
} while <(condition)>
```

Look at the following program

```
// Program - 3.8
# include <iostream.h>
# include <conio.h>

// to print the square of numbers
// between 2 to 5

void main()
{
   clrscr();
   int num = 2;
   do
   {
       cout << num * num << '\t';
       num += 1;
   }
   while (num < 6);
   getch();
}
```

Answer the following questions based on the Program - 3.8

A. Identify the
   1. control variable used .
   2. Identify the statements that form the body of the loop
   3. The test expression

B. How many times will the loop be executed ?

C. What is the output of the program?

D. What type of loop is this ?

A.

1. **The control variable is num**
2. **Statements forming the body of the loop are :**

```
 cout << num * num << '\t';
       num += 1;
```

3. **num < 6 is the test expression**

B. 4 times

C. 4   9   16   25

D. Exit – check loop

1. Enters the loop
2. Prints the square of num
3. Increments the control variable by 2
4. Evaluates the condition , based on which the control is transferred to step 2
5. End

do … while <(condition)> is called as exit- check loop, as the condition(test expression) marks the last statement of the body of the loop. The following snippets show the various styles of constructing conditions.

```
Int ctr = 1, sum = 0, check =
1;
do
{
  cout << ctr;
   sum = sum + ctr;
   ctr = ctr + 2;
   check = (ctr < 11);
}while(check);
```

```
Int ctr = 5, sum = 0;
do
{

  cout << ctr;
   sum = sum + ctr;
   ctr = ctr -  2;

}while(ctr);
```

```
int ctr = 5,sum = 0,c=1;
do
{

   cout << ctr;
    sum = sum + ctr;
    ctr = ctr -  2;

}while(ctr >= 1);
```

What is the output displayed by the following snippets A and B ?

```
// snippet A
# include <iostream.h>
# include <conio.h>

void main()
{
   int i = 10;
   do
   {
        cout << i;
        i--;
   } while (i <= 10);
   getch();
}
```

```
//snippet B
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 10; choice = 1;
  do
  {
    cout << i;
    i++;
  }while (choice);
  getch();
}
```

Snippet A – the loop will be executed till the variable i gets the value as –32768, and the snippet B will result in infinite loop, as the value stored in the variable **choice** is 1 thus rendering the test expression to be TRUE all the time in both the snippets . It is very important to construct appropriate conditions that would evaluate to false at some point of time, and also incrementing/updating the control variable that is linked to the test expression in the while loop.

while <(condition)>{ … } loop : is called as the **entry-check** loop. The basic syntax is :

```
while <(condition)>
{
action block
}
```

The body of the while loop will be executed only if the test expression results true placed in the while statement. The control exits the loop once the test expression is evaluated to **false**. Let us rewrite all the programs that were discussed under do..while loop (Program - 3.9)

```
// Program - 3.9
# include <iostream.h>
# include <conio.h>

// to print the square of numbers
// between 2 to 5

void main()
{
   clrscr();
   int num = 2;
   while (num < 6)
   {
         cout << num * num << '\t';
         num += 1;
   }
   getch();
}
```

Condition (test expression) is placed at the entry of the body of the loop

The working of the above loop as follows :

1. Initialises the control variable num to 2
2. The test expression num < 2 is evaluated, control is transferred to step 3, only if the test expression is TRUE
3. Prints the square of the value stored in num
4. Increments num by 1
5. Control is transferred to step 2
6. End

Answer the following questions based on the Program - 3.10

```
//Program-3.10
# include <iostream.h>
# include <conio.h>

void main()
{
  int x = 3, y = 4, ctr = 2,res = x;
  while(ctr <= y)
  {
      res *= x;
      ctr += 1;
  }
  cout << "x to the power of y is : "
        << res;
  getch();

}
```

Answer the following questions based on the Program - 3.10

A. Identify the
   1. Control variable used .
   2. Statements that form the body of the loop
   3. The test expression

B. How many times will the loop be executed ?

C. What is the output of the program?

D. What type of loop is this ?

Answers :
1. Control variable used is ctr
2. res *= x; ctr += 1;
3. ctr <= y

B. 3 times

C. 81

Entry- check or entry – controlled loop

What will be the output of the following Program - 3.11 if the values read for choice is y,y,y,y,n?

```
// Program - 3.11
# include <iostream.h>
# include <conio.h>
void main()
{
  clrscr();
  int counter = 0;
  char choice = ‘y’;
  while (choice == ‘y’)
  {
       cout << “Continue <y/n> ...”;
       cin >> choice;
       counter = counter + 1;
   }
   cout << “\The loop is executed ..”
         << counter << “  times”;
   getch();
}
```

The following snippets are invalid. Why are they invalid ? Correct the code for proper execution.

```
//Program - 12 A
# include <iostream.h>
# include <conio.h>
//to print numbers between
5&10
void main()
{
    int start = 5,end = 10;
    while (start >= end)
          cout << start++;
    getch();
}
```

```
//Program - 12 B
# include <iostream.h>
# include <conio.h>
//to print numbers between 5&10
void main()
{
     int start = 5,end = 10;
    while (start <= end)
          cout << start++;
    getch();
}
```

```
//Program – 13 A
# include <iostream.h>
# include <conio.h>
// to print numbers between
10&5
void main()
{
    clrscr();
    int start = 5,end = 10;
    while (start <= end)
          cout << start--;
    getch();
}
```

```
//Program – 13 B
# include <iostream.h>
# include <conio.h>
// to print numbers between 10&5
void main()
{
    clrscr();
    int start = 5,end = 10;
    while (start <= end)
          cout << end--;
    getch();
}
```

```
//Program – 14 A
# include <iostream.h>
# include <conio.h>
// to print numbers
       between 1 & 5
void main()
{
    clrscr();
    int start = 1;
    while (Start <=5)
          cout << start++;
    getch();
}
```

```
//Program – 14 B
# include <iostream.h>
# include <conio.h>
// to print numbers
       between 1 & 5
void main()
{
    clrscr();
    int start = 1;
    while (1)
          cout << start++;
    getch();
}
```

for (; ;) .. loop : is an entry controlled loop and is used when an action is to be repeated for a predetermined number of times. The syntax is

for(intial value ; test-condition ; increment)

```
{
    action block;
}
```

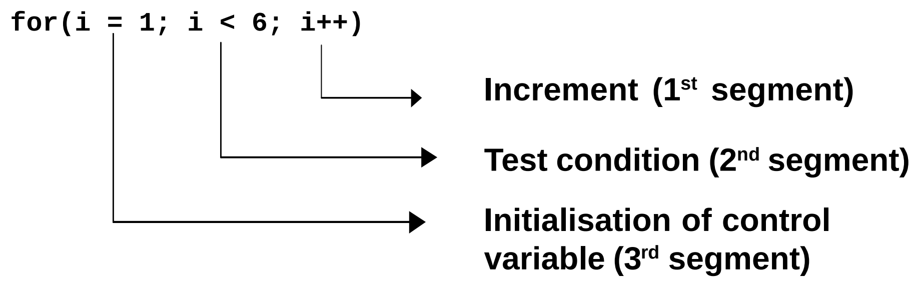
The general working of for(;;)loop is :

1. The control variable is initialized the first time when the control enters the loop for the first time
2. Test condition is evaluated. The body of the loop is executed only if the condition is TRUE. Hence for(;;) loop is called as entry controlled loop.

3. On repetition of the loop, the control variable is incremented and the test condition will be evaluated before the body of the loop is executed.
4. The loop is terminated when the test condition evaluates to false.

The following program illustrates for(;;) loop :

```
//Program - 3.15
# include <iostream.h>
# include <conio.h>

void main()
{
  int i,fact = 1;
  for(i = 1; i < 6; i++)
      fact *= i;
  cout << "\nThe factorial of the number is .." << fact;

}
```


for(i = 1; i < 6; i++)
Increment (1st segment)
Test condition (2nd segment)
Initialisation of control variable (3rd segment)

- ✓ **Initialisation is executed only once, ie., when the loop is executed for the first time**
- ✓ **Test condition is evaluated before the commencement of every iteration**
- ✓ **Increment segment is executed before the commencement of new iteration.**

Now look at the following programs and write out as to what will be displayed?

```
//  Program – 3.16
# include <iostream.h>
# include <conio.h>

void main()
{
    int ctr = 10;
    for(; ctr >= 6; ctr—)
       cout << ctr << '\n';
}
```

```
Output
10
 9
 8
 7
 6
```

```
//Program – 3.17
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  for(int i=2,fact =
1;i<6;fact*=i,i++);
  cout << "\nThe factorial  ..” <<
fact;
  getch();
}
```

```
Output displayed..

The factorial.. 120
```

Have you noticed the for statement, comprising of more than one statement in segments incrementation and initialisation ? Syntatically and logically the above statement is valid. Each segment in the for loop can comprise a set of instructions, each instruction should be separated by a comma operator. Can you analyse as to what will be the output of the following segment ?

```
void main()
{

  for (int i = 1, j = 0 ; i < 8,j<3;i++,j++)
       cout << i << '\t';
 for (int i = 1,,j = 0 ;j < 3,i < 8;i++,j++)
       cout << i << '\t';
}
```

```
Output produced will be :
1 2 3 // loop is executed till j < 3
1 2 3 4 5 6 7 // loop is executed
till i < 8 Recall the working of comma
operator.
```

Now look at the following for..loop constructs.

```
// Program – 3.18
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int sum =0, ctr = 1;
  for(;ctr <= 5;)
  {
    sum += ctr;
    ctr = ctr + 1;
  }
  cout << "\nSum :" << sum;
  getch();
}
```

Output displayed will be
Sum : 15

Have you noticed, initialization and incrementation segments are not included in the for(..) construct.

```
// Program – 3.19
# include <iostream.h>
# include <conio.h>
void main()
{
   clrscr();
   int sum =0, ctr = 1;
   char ch ='y';
   for(;ch == 'y';)
   {
     sum += ctr;
     ctr++;
      cout << "\nContinue <y/n>
? ..";
     cin >> ch;
   }
   cout << "\nSum :" << sum;
   cout << "\nChoice : " << ch;
   getch();
}
```

```
Continue <y/n> ? ..y
Continue <y/n> ? ..y
Continue <y/n> ? ..y
Continue <y/n> ? ..n
sum:10
Choice : n
```

Have you noticed that a for loop is used like a dynamic loop, where the iterations are determined during run time.

What is wrong with the following snippets ?

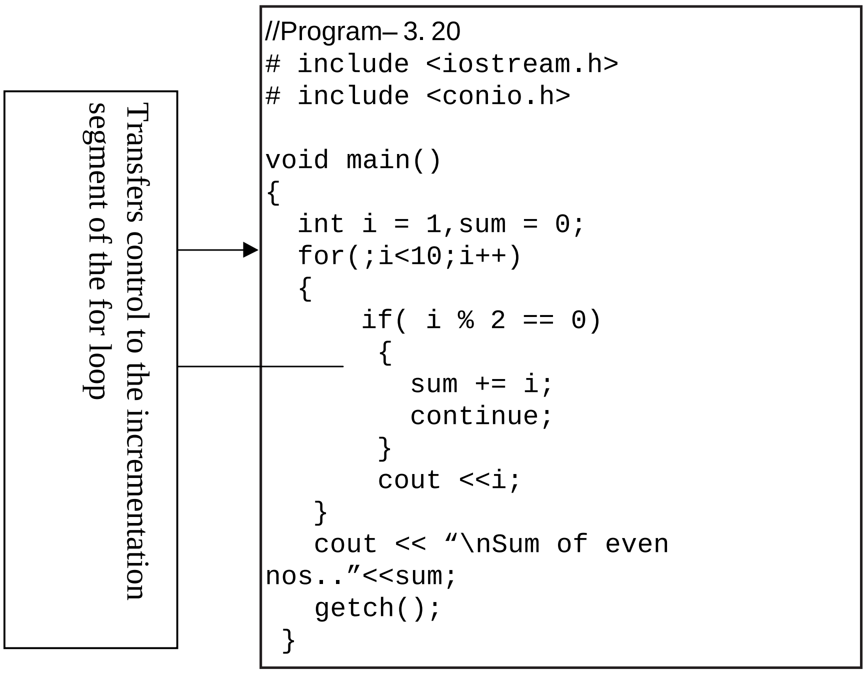
What is the impact of the following statements ?

```
int sum = 0;
for(ctr = 1; ctr < 5; ctr++);
  sum += ctr;
cout << sum;
```

The output will be 5. Can you reason it out ?
The reason is a semicolon placed after for loop, hence the statement sum+=ctr is not treated as part of the for loop body.

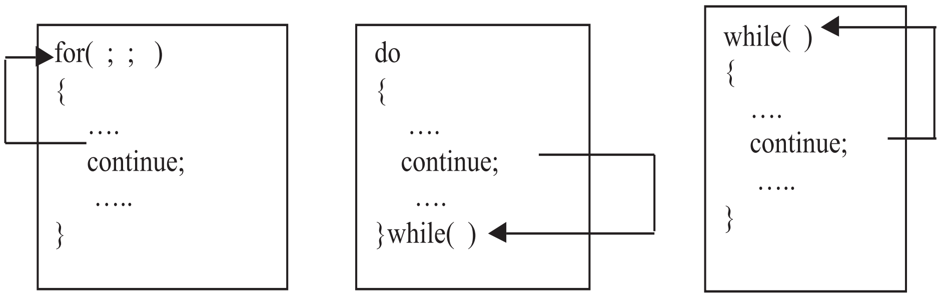
3.5.3 continue

The continue statement forces the next iteration of the loop to take place, skipping any code following the continue statement in the loop body.


Transfers control to the incrementation segment of the for loop

```
//Program– 3. 20
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 1,sum = 0;
  for(;i<10;i++)
  {
      if( i % 2 == 0)
       {
         sum += i;
         continue;
       }
       cout <<i;
   }
   cout << “\nSum of even
nos..”<<sum;
   getch();
 }
```

Working of continue statement in various loops is as follows :



```
for( ; ; )
{
  ….
  continue;
   …..
}
```

```
do
{
   ….
  continue;
   ….
}while( )
```

```
while( )
{
   ….
  continue;
   …..
}
```

What will be the output of the following segments ?

```
int ctr = 1;
for( ; ctr < 10; ctr++ )
 {
    cout << ctr;
    ctr = 1;
}
```

Since ctr is intialised in the body of the for loop, the loop will re sult in an infinite loop, and the output displayed will be 1

```
int ctr = 1;
for( ctr = 1; ; ctr++ )
     cout << ctr;
```

Since test expression is missing, the loop wil be executed until ctr reaches 32767, the integer data maximum value.

3.5.4 break

A loop's execution is terminated when the test condition evaluates to false. Under certain situations one desires to terminate the loop , irrespective of the test expression. For example consider the Program - 3. 21

```
//Program - 3.21
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   int a[] = {1,2,3,4,5,6,7,8,9};
   int search_item = 7;
   for(int x=0; x<9;x++)
  {
       if (a[x] == search_item)
       {
           cout << “\nItem found at position ..” << x;
           break;
        }
   }
    cout << ‘\n’ << “value of index position is ..” << x;
    getch();
}
```

Output displayed will be :
Item found at position .. 6
value of index position is .. 6

✓ The control is transferred to cout statement written outside the loop, because of break statement. The loop is terminated when x takes the value as 6, because of break statement.

✓ **break statement would exit the current loop only.**
✓ **break statement accomplishes jump from the current loop**

Nested loops : It is possible to nest loop construct inside the body of another. Look at the following Program - 3.22

```
// nesting of loops – Program- 3.22
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   for(int i = 1; i <= 3; i++)
   {
       int j = 1;
       while(j <= i)
       {
         cout << "* ";
         j++;
       }
       cout << '\n';
  }
  getch();

}
```

Output displayed :

```
*
* *
* * *
```

Working of the loops is as follows :

```
for(int i = 1; i <= 3; i++)
{
   int j = 1;
   while(j <= i)
   {
     cout << "* ";
     j++;
   }
   cout << '\n';
}
```

The iterations of the nested loops are as follows :

for ..loop	while loop
i = 1	is executed once (j<= I)
i = 2	Is executed twice (j = 1 .. 2)
i = 3	Is executed thrice (j = 1.. 3)

Table 3.4 Nested Loops Example

Can you write out as to what will be the output of the following program ?

```
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   int i = 1, j = 1;
   while(i <= 3)
   {
        cout << ‘\n’;
        for(i=1;i<=j;i++)
         cout << ‘*’;
        i++;
        j++;
   }
   getch();
}
```

Output ??

The rules for the formation of nested loops are :

1. An outer loop and inner loop cannot have the same control variable, as it will lead to logical errors
2. The inner loop must be completely nested inside the body of the outer loop.

3.6 Program Development

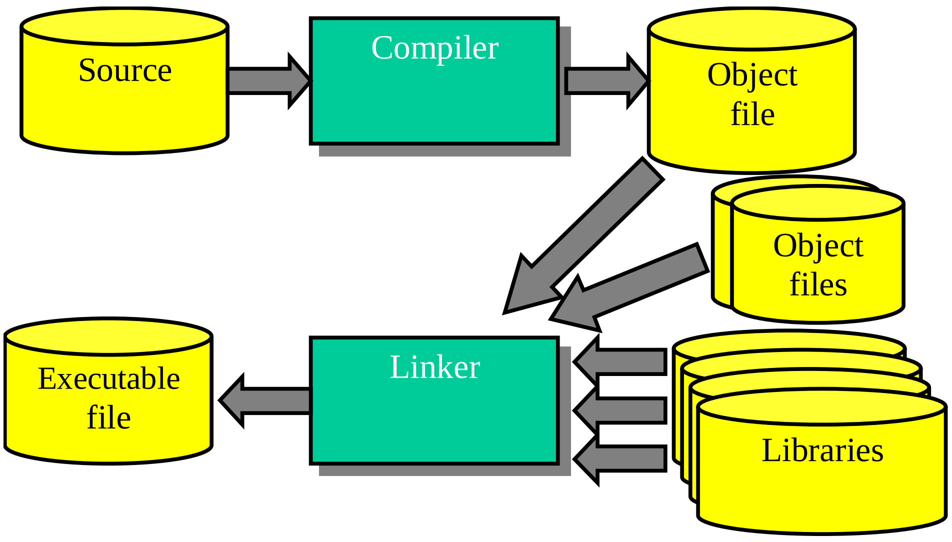


Fig. 3.1 Program Execution

Programs are written in high level language using the grammar of a computer language. A Program written in high level language is called as the Source Code. The source code has to be converted to machine-readable form. The machine-readable form of a program is called as Object file. Compilers create object files from source code. Compilers are translator programs that create a machine-readable program from the source code. Compiler checks for the grammar of language (syntax). An object file is created from an error free source code. The object file is linked with the essential libraries to generate an executable file. This sequence of actions is shown in Fig. 3.1.

Exercises

1. Categorise the following declarations as valid/invalid. If invalid, specify the reasons.

Declarations	Valid/Invalid	Reason
int A;a;		
char name(10);		
float f,int;		
double d, float f;		
int 1choice, _2choice		

2. Debug the following program. Rewrite the corrected program.

```
include <iostream.h>
include <conio.h>

void main()
{
  int N1,n1;
  cin << ‘ \nEnter two number s ..’;
  result := N1 * n1;
  cout << ‘ \n’ << Result;
}
```

3. Write appropriate declaration statements for the following :

   a. To store the result of the expression 8/3 .

   b. To initialise Emp_Name with the value “Kalam”

   c. To accept choice from user indicating Y-yes and N – no

4. Point out errors in the following snippets :

a.
```
int  a = 10, b = 5;
if a > b
cout << a;
```

b. if (a<b) && (a<0)

```
cout << "a is negative and …"
```

c. char option = 'Y';

```
do while option == 'y'
{
cout << '*';
……
}
```

d. for(int I = 1; I < 10; I++)

```
cout << I * 2;
```

e. do

```
{
 cout << '*';
}while(cout << "\nContinue <y/n>…";cin>>ch;ch == 'y');
```

5. What will be the output of the following snippets / programs?

```
// 5 a.
# include iostream.h>
# include <conio.h>

void main()
{
   int feet;
   const int inch_conversion = 12;
   clrscr();
   cout << "\nEnter feet …";
   cin >> feet;
   cout << "\nConverted to inches …"
          << feet * inch_conversion;
}
input-7 for feet
```

```
// 5 b.
# include <iostream.h>
# include <conio.h>

void main()
{
   int I = 1, sum = 0;
   clrscr();
   while(I++ <= 5)
  {
      cout << '\n' << I;
      s += I;
  }
 cout << "\nValue of the variable I after
executing the while loop ..." << I << "\nSum
:…" << s;
```

```
// 5 c
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 1, sum = 0;
  clrscr();
  while(++i <= 5)
  {
        cout << ‘\n’ << i;
        sum += i;
  }
  cout << ‘\n’ << i << ‘\t’ << sum;
  getch();
}
```

```
// 5 d
# include <iostream.h>
# include <conio.h>
void main()
{
  int i = 1, sum = 0;
  clrscr();
  for(i = 1; i <= 5; i++)
  {      cout << ‘\n’ << i;
         sum += i;
  }
  cout << ‘\n’ << i << ‘\t’ << sum;
  for(i = 1; i <= 5; ++i)
  {
         cout << ‘\n’ << i;
         sum += i;
  }
  cout << ‘\n’ << i << ‘\t’ << sum;
 }
```

```
//5e
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 1, j = 1;
  clrscr();
  do
  {
        while (j<=i)
        {
           cout << ‘#’;
           j++;
        }
        cout << ‘\n’;
        i++;
        j = 1;
  } while(i<= 5);
  getch();
}
```

```
// 5 f
# include <iostream.h>
# include <conio.h>

void main()
{
  int num = 1784, s= 0, d = 0, x;
  x = num;
  clrscr();
  for(;num > 0;)
  {     d = num % 10;
        s += d;
        num = num / 10;
  }
  cout << “\nThe sum of digits of “
       << x << “is : “
       << s;
  getch();
}
```

```
//5 g
# include <iostream.h>
# include <conio.h>
void main()
{
  clrscr();
  for(int i = 1,s = 0; ; i++)
  {
        if (i%2 == 0)
           continue;
        s += i;
        if ( i > 9)
           break;
  }
  cout << “\nThe sum is ....” <<
s;
  getch();
```

```
// 5 h
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  for(int i = 1,x = 0;i <= 5; i++)
    x = x + i%2==0 ? i*1 : i * -1;
  cout << x;
  getch();
}
```

```
//5 j
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  do
  {
    cout << “\ndo loop ...”;
  } while (0);
  getch();
}
```

```
//5 k
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int i = 0;
  for(i = -5; i >= 5; i–)
        cout << “Bjarne Stroustrup”;
  cout << “\nReturning to Edit Window..”;
  getch();
}
```

```
//5 l
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
 int month = 5;
 if (month++ == 6)
    cout << “\nMay ...”;
 else if (month == 6)
   cout << “\nJune ...”;
 else if (–month == 5)
   cout << “\nMay again ..”;

}
```

```
// 5 m
# include <iostream.h>
# include <conio.h>
void main()
{ int day = 3;
  switch (day)
  {
    case 0 : cout << “\nSunday ..”;
    case 1 : cout << “\nMonday ..”;
    case 2 : cout << “\nTuesday ..”;
    case 3 : cout << “\nWednesday .. “;
    case 4 : cout << “\nThursday ..”;
    case 5 : cout << “\nFriday ..”;break;
    case 6 : cout << “\nSaturday ..”;
  }
 }
```

```
// 5 n
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int bool = 2,b =4;
  while(bool)
  {
    cout << bool << ‘\t’ << ++b << ‘\n’;
    bool–;
    b–;
  }
  getch();

}
```

6. Program Writing

a. Write a program to compute a^b where a and b are of real and integer types(use while .. loop)
b. Write a program to compute the factorial of a given number. (use for( ) loop)
c. Write a program to generate fibonacci series upto n^th term. Fibonacci series is : 0,1,1,2,3,5,8,12,20,32 …
d. Write a program to print the following patterns :

```
1
1    2
1    2    3
1    2    3    4
1    2    3    4    5
```

```
4
3    4
2    3    4
1    2    3    4
```

```
A
B    C
D    E    F
G    H    I    J
```

Using a switch, write a program to accept the day in a month, and print the messages as :

If day is 1, message is 1st day in the month

If day is 2,22 , message is 2nd / 22nd day in the month

If day is 3,23, message is 3rd/23rd day in the month

If day is 4,14,15,16… message is 4th /14th .. day in the month

CHAPTER 4

FUNCTIONS

4.1 Introduction

Functions are the building blocks of C++ programs. Functions are also the executable segments in a program. The starting point for the execution of a program is main ( ). Functions are advantageous as they

- ✓ reduce the size of the program
- ✓ induce reusability of code

```
// To print X!/F! – Program - 4.1
# include < iostream.h >
# include <conio.h >
void main ( )
{  int x, f,xfact=1, ffact = 1;
clrscr ( );
cout << “\nEnter values ...”;
cin >> x >> f;   for (int a =1; a < = x;
a ++)
xfact * = a;
for (a = 1; a< = f; a++)
ffact * = a;
cout << xfact/ffact;
getch();
}
```

```
// To print X!/F! using functions
// Program - 4.2
# include <iostream.h >
# include <conio.h >
int fact (int num)
{
int factorial = 1;
for (int a=1; a<= num; a++)
 factorial *= a;
return factorial;
}
void main ( )
{   int x, f;
   clrscr ( );
   cout<<“\nEnter values…”;
    cin >> x >> f;
   cout << fact (x)/fact(f);
  }
```

In Program –4.1, the code for Factorial evaluation is repeated twice. In the Program –4. 2, the function **fact (int num)** is invoked whenever required. Functions thus encourage :

- ✓ Reusability of code (function fact is executed more than once)
- ✓ A function can be shared by other programs by compiling it separately and loading them together.

The general syntax showing the various blocks of a function :

```
// To print X!/F! using functions
// Program - 4.3
# include <iostream.h >
# include <conio.h >
int fact (int num)
{
  int factorial = 1;
  for (int a=1; a<= num; a++)
        factorial *= a;
  return factorial;
}

void main ( )
{   int x, f;
    clrscr ( );
    cout<<“\nEnter values…”;
    cin >> x >> f;
    cout << fact (x)/fact(f);
}
```

Prototype with arguments

return statement marks the end of the function and also transfers control to the statement after call statement

Statements to invoke function fact – call statement

4.2 Function Prototyping

Functions should be declared before they are used in a program. Declaration of a function is made through a function prototype.

For example look at the Program –4. 4.

```
//Program -4.4
# include <iostream.h>
   void fun (char name [ ]) ; ──────────→  function prototype
                                            (declaration of
                                            function)
  void main ( )
  {
    char n [ ] = { “C++ programming….”};
    fun (n);
  }
 void fun (char name[] )      ──────────→  Function
{ cout << name; }                           Definiton
```

The prototype provides the following information to the compiler

1. Number and type of arguments -(char name [ ] - is an argument)

2. The type of return values (in the above example fun does not have any return value, as the data type of the function is void. Recall Program –2 , in which the return type is int for the function Fact ( ))

The general syntax of a function prototype

<type > <function identifier > <arguments);

For example :

```
void fun (char);

int max (int, int);

int max (int a, int b);
```

The main purpose of function prototype is to help the compiler to check the data requirement of the function. With function prototyping, a template is always used when declaring and defining a function. When a function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly. Any violation in matching of the arguments or the return types will be treated as errors by compiler, and flagged at the time of compilation.

Why do you think the prototype int max (int, int) is valid??

In a function declaration, the names of the arguments are dummy variables and therefore they are optional. The variables in the prototype act as place holders.

The arguments' names are required in function definition, as the arguments are referenced inside the function.

```
void fun (char name [ ] )
{ cout << name; }
int fact (int num)
{ int f = 1;          |———————>  The  argument num is
  for (int a = 1;, a < = num; a ++)     referenced inside function.
fx = a;
return f;
}
```

4.3 Calling a Function

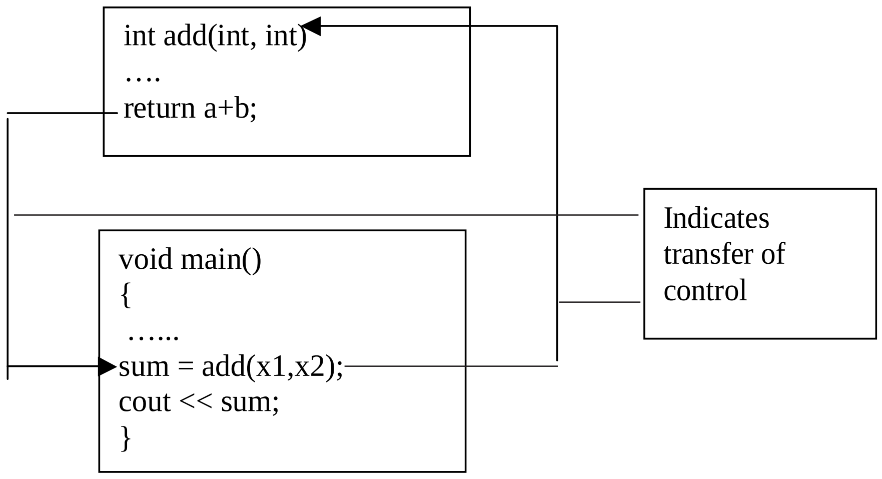
A function can be called or invoked from another function by using its name. The function name may include a set of actual parameters, enclosed in parentheses separated by commas. For example,

```
// Program - 4.5
# include <conio.h>
# include <iostream.h>


int add (int a, int b)

{ return a + b;}
void main ( )
{ int x1, x2, sum = 0
  cin >> x1 >> x2;

   sum = add (x, x2);
  cout << sum;
}
```

formal parameters

actual parameter

Working of a function :



```
int add(int, int)
….
return a+b;
```

```
void main()
{
 …...
sum = add(x1,x2);
cout << sum;
}
```

4.4 Parameter Passing in Functions

The call statement communicates with the function through arguments or parameters.

> Parameters are the channels through which data flows from the call statement to the function and vice versa.

In C++, functions that have arguments can be invoked by

- ✓ Call by value
- ✓ Call by reference

4.4.1 Call by Value

In this method, the called function creates new variables to store the value of the arguments passed to it. This method copies the values of actual parameters (parameters associated with call statement) into the formal parameters (the parameters associated with function header), thus the function creates its own copy of arguments and then uses them. Recall the example Program - 4.5

```
// Program - 4.5
# include <iostream.h>
# include <conio.h>
int add (int a, int b)
{ return a + b;}
void main ( )
{ int x1, x2, sum;
cin >> x1 >> x2;
sum = add (x, x2);
cout << sum;
}
Assume x1 = 5, x2 = 7
```

```
     Main()              add()

   x1 = 5               a = 5
   x2 = 7                b = 7
    sum =
Assume address of the variables :

   x1  = Oxf1          address   data
    x2 = Oxf3           Oxf1       5
    a   = Oxf7          Oxf2
    b   = Oxf9          Oxf3       7
  sum = Oxf6            Oxf4
                        Oxf5
                        Oxf6      12
                        Oxf7       5
                        Oxf8
                        Oxf9       7
```

Have you noticed that the actual parameters x1 and x2 and the formal parameters a&b have been allocated different memory locations? Hence, in call by value method, the flow of data is always from the call statement to the function definition.

```
// Program - 4.6
// To exchange values
#include <iostream.h>
#include <conio.h>
# include <iomanip.h>
void swap (int n1, int n2)
{   int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
    cout << ‘\n’<<n1<<‘\t’<<n2<<‘\n’;
}
```

```
void main ( )
{
   int m1 = 10, m2 = 20;
   clrscr ( );
   cout <<“\n Values before invoking swap”
        << m1 << ‘\t’ << m2;
  cout << “\n Calling swap..”;
  swap (m1, m2);
  cout << “\n Back to main.. Values are”
       << m1 << ‘\t’ << m2;
  getch ( );
}
```

Output

```
Values before invoking swap      10      20
Calling swap …..
20      10
Back to main…… Values are 10    20
```

Why do you think the exchange of values of the variables m1 and m2 are not reflected in the main program??

When arguments are passed by value, the called function creates new variables of the same data type as the arguments passed to it. The values of these arguments are copied into the newly created variables. Hence, changes or modifications that are made to formal parameters are not reflected in the actual parameters.

In call by value method, any change in the formal parameter is not reflected back to the actual parameter.

4.4.2 Call by reference

In this method, the called function arguments - formal parameters become alias to the actual parameters in the calling function. This means that when the function is working with its own arguments, it is actually working on the original data. Recall the example Program 4.6. Let us now rewrite the function using reference parameters.

```
//Program - 4.7
// To exchange values

# include <iostream.h>
#include <conio.h>
void swap (int &n1, int &n2)
{
  int temp;
  temp = n1;
  n1 = n2;
  n2 = temp;
  cout<<'\n'<< n1
        <<'\t'<<n2<<'\n';
}
```

```
void main ( )
{
   int m1 = 10, m2 = 20;
   clrscr();
   cout<<“\nValues before swap call”
         << ‘\t’ << m1 << ‘\t’ << m2;
   swap(m1,m2);
   cout<<“\n Calling swap..”;
   cout<<“\n Back to main.Values are”
         << ‘\t’ << m1 << ‘\t’<< m2;
   getch ( );
}
```

```
Output:

Values before invoking swap 10   20
Calling swap..
   20    10
Back to main.Values are 20        10
```

The modifications made to formal parameters are reflected in actual parameters, because formal and actual parameters in reference type point to the same storage area.

Look at the following depiction:

Main	**Swap**
m1 = 10	n1 = 10
m2 = 20	n2 = 20
	temp

Assume storage area of m1 is Oxf1, and m2 is Oxf4.

m1 = Oxf1 = 10

m2 = Oxf4 = 20

Reference to formal parameters may be read as

n1 = 10; n1 is a reference to m1, which may be depicted as:

```
int  &n1 = m1
```

This means that n1 is an alias to m1, hence m1 and n1 refer to same storage area, hence the statements may be rewritten as :

```
n1 = m1 = Oxf1 = 10
n2 = m2 = Oxf4 = 20
```

Address	Before Exchange	After exchange
Oxf1 (n1, m1)	10	20
Oxf4 (n2, m2)	20	10

Hence, changes made to formal parameters are reflected in actual parameters.

In call by reference method, any change made in the formal parameter is reflected back in the actual parameter.

Try out

```
// Reference variables
// Program -4.8
        # include <iostream.h>
        # include <conio.h>
        void main ( )
        {
          int num1 = 10, & num2 = num1;
           num2 ++;
            cout << num1;
        }
```

Output displayed will be **11**

By virtue of reference num1 and num2 point to the same storage location.
Hence, change of value in num1 is reflected in num2.

Rules for actual parameters:

1. The actual parameters can be passed in the form of constants or variables or expressions to the formal parameters which are of value type.

For example,

For a function prototype : int add (int n1, int n2); - the call statements may be as follows :

```
x = add (5, 10);
x = add (a1, a2); where a1 and a2 are variables
```

2. The actual parameters can be passed only as variables to formal parameters of reference type.

   For example,

```
int add (int & n1, int & n2);
x = add (a1, b1) ;    where a1 and b1 are variables
```

   The following call statements are invalid:

```
x = add ((a1 + b1), a1);
x = add (5,10);
```

```
// Program - 4.9
//To print 5 stars per row and 5 such rows
# include <iostream.h>
# include <conio.h>

void fun_starts (int &i)
{
 int j = 5;
 for (i= 1; i <= j; i++)
      cout << ' '<<'*';
}

void main ( )
{
  int mi = 1;
  clrscr( );
  for (; mi<= 5; mi++)
  {
      cout << '\n';
      fun_starts (mi);
  }
   getch ( );
      }

}
```

Why does the above program does not behave the way to produce the result as mentioned in comment line?

The output produced is :

```
      * * * * *
```

Reason – the variable i is a reference to the variable mi.Since the variable i gets a value 6 in the function, mi is also automatically up-dated to 6, hence, the 'for' loop in main ( ) is executed only once.

4.3.4 Default arguments

In C++, one can assign default values to the formal parameters of a function prototype.

For example :

```
// Program - 4.10
// formal parameters with default values
# include <iostream.h>
# include <conio.h>
float power (float n, int p = 1)
{
  float prd = 1;
  for (int i = 1;  i<= p; i++)
      prd *= n;
  return prd;
}

void main ( )
{
   clrscr ( );
   int x = 4, b = 2;
   cout << “\n Call statement is power(b, x)...”
         << power (b, x);
   cout << “\n Call statement is power(b).. “
         << power (b);
   getch ( );
}
```

Output:

```
Call statement is power (b, x)      ..      16
Call statement is power (b)         ..      2
```

In the call statement power (b,x), initialization is

n= b, p = x

In the second form power (b), the variable n is initialized, whereas p takes the value 1 (default argument), as no actual parameters is passed.

NOTE:

- ✓ The default value is given in the form of variable initialization.
- ✓ The default arguments facilitate the function call statement with partial or no arguments.
- ✓ The default values can be included in the function prototype form right to left, i.e., we cannot have a default value for an argument in between the argument list.

Try out the following Program.

```
//Program - 4.11
# include <iostream h>
# include <conio.h>
int area (int side1 = 10, int side2=20
{ return (side1 * side 2); }

void main ( )
{    int s1 = 4, s2 = 6;
     clrscr ( ) ;
     cout << area (s1, s2) << ‘\n’;
     cout << area (s1) << ‘\n’;
     cout << area (s2) << ‘\n’;
     getch ( );
}
```

```
Output:
 24
  80
 120
Variable initialization
I form - side1 = s1,
side2 = s2
II form - side1 = s1
III form - side1 = s2
```

What will be the output of the following program?

```
// Program - 4.12
// arguments with default values

# include <iostream.h>
# include <conio.h>

void print (int times, char ch = ‘ * ‘)
{
  cout << ‘\n’;
  for (int i = 1, i < = times; i ++)
     cout << ch;
}
void main ( )
{
   clrscr ( );
    print (50);
    print (‘A’, 97);
    print ( );
}
```

Solution:

print (50)	-	50 is assigned to the argument times. Hence, the output ‘ * ‘ will be printed 50 times.
print (‘A’, 97)	-	‘A’ is assigned to argument times (implicit conversion of character to integer takes place). Hence, times gets the value as 65.

The constant 97 is assigned to ch, hence ch gets the value as ‘a’.

The actual parameters are matched with formal parameters on the basis of one- to -one correspondence.
Hence, 65 times, ‘a’ will be printed.

print ( ) - In the absence of actual arguments, the formal parameters takes the default arguments. Hence, the output will be displayed as ‘ * ‘ - 50 times.

4.5 Returning Values

The functions that return no value is declared as void. The data type of a function is treated as int, if no data type is explicitly mentioned. For example,

```
int add (int, int);
add (int, int);
```

In both prototypes, the return value is int, because by default the return value of a function in C++ is of type int.

Look at the following examples:

Sl.No.	**Function Prototype**	**Return type**
1	float power (float, int)	float
2	char choice ( )	char
3	char * success ( )	pointer to character
4	double fact (int)	double

4.5.1 Returning by reference

Reference or alias variables:

```
// Program -  4.13
# include <iostream.h>
# include <conio.h>
void main ( )
{   int i = 5;
    int  &count = i ;
    cout << “\nCount: “ << count;
    count ++;
    cout << “\ni: “ << i;
    getch ( );
}
```

Output:

```
Count: 5
 i: 6
```

Why do you think count gets the value as 5?

Why is the variable i updated to 6, when count was incremented???

The reason being that the variables count and i refer to the same data in the memory. Reference variables also behave the same way.

Using this principle, try and find out as to what will be output of the following program:

```
// Program - 4.14
# include <iostream h>
# include <conio.h>
int &maxref (int &a, int &b)
{ if (a>b)
      return a;
   else
       return b;
}
void main ( )
{ int x = 20, y = 30, max = 0;
     max = maxref (x,y);
      cout << "\n Maximum is: " << max;
}
```

Output

Maximum is : 30

In the above program, the function maxref returns a reference to int type of variable. The function call maxref (x,y) will return a reference to either a or b depending upon which one is bigger of the two. Hence, the variable max gets the value of the variable y.

What will be the output of the following program?

```
// Program 4.15
# include <iostream h>
# include <conio.h>
int & maxref (int & a, int & b)
{ if (a > b),
return a;
   else
     return b;
}
void main ( )
{ int x = 20, y = 30, max = 0;
     maxref (x,y) = -1;
      cout << "\n Value of x is : " << x;
      cout << "\n Value of y is: " <<y;
       getch ( );
}
```

Output

```
Value of x is   :         20
Value of y is   :         -1
```

NOTE:

1. A function returning a reference can appear on the left-hand side of an assignment.
2. In the above example, the variable y gets the value -1, since the function maxref. establishes reference with the formal parameter b, whose corresponding variable in main block is 'y'.
3. The formal parameters for a reference function should always be of reference parameter type in the sense -

    int & maxref (int a, int b);
    will yield compilation error, as the scope of the variables a&b
    are within the function block maxref.

4.6 Inline Functions

We have listed out the advantages of functions as

- ✓ Reusability of code leading to saving of memory space and reduction in code size.

While this is true, we also know that call statement to a function makes a compiler to jump to the functions and also to jump back to the instruction following the call statement. This forces the compiler to maintain overheads like STACKS that would save certain special instructions pertaining to function call, return and its arguments. This reduces the speed of program execution. Hence under certain situations specially, when the functions are small (fewer number of instructions), the compiler replaces the function call statement by its definition ie., its code during program execution. This feature is called as inlining of a function technically called as **inline** function.

An **inline** looks like a normal function in the source file but inserts the function's code directly into the calling program.
Inline functions execute faster but require more memory space.

Now look at the following example.

```
// Program - 4.16
// inline functions

# include <iostream.h>
# include <conio.h>

inline float convert_feet(int x)
{
  return x * 12;
}

void main()
{
  clrscr();
  int inches = 45;
  cout << convert_feet(inches);
  getch();
}
```

```
// working of Program - 4.16
// inline functions

# include <iostream.h>
# include <conio.h>


void main()
{
  clrscr();
  int inches = 45;
  cout << inches * 12 ;
  getch();
}
```

As shown in the above example, the call statement to the function (convert_feet(inches) will be replaced by the expression in the return statement (inches * 12).

To make a function inline, one has to insert the keyword **inline** in the function header as shown in Program 4.16.

Note :

inline keyword is just a request to the compiler . Sometimes the compiler will ignore the request and treat it as a normal function and vice versa.

4.7 Scope Rules of Variables

Scope refers to the accessibility of a variable. There are four types of scopes in C++. They are:

1. Local scope
2. Function scope
3. File scope
4. Class scope

4.7.1 Local scope

```
// Program - 4.17
// to demonstrate local variable
# include < iostream.h
# include <conio.h>
void main ( )
{
      int a, b ;
       a = 10;
       b = 20;
   if (a > b)
   {  int temp; // local to this if block
      temp = a;
      a = b;
      b = temp;
    }
    cout << '\n Descending order…';
    cout << '\n' <<a << '\n' <<b;
    getch ( );
}
```

- A local variable is defined within a block.
- The scope of a local variable is the block in which it is defined.
- A local variable cannot be accessed from outside the block of its declaration.

Program-4.18 demonstrates the scope of a local variable.

```
//Program - 4.18
# include <iostream.h>
# include <conio.h>
void main ( )
{    int a, b;
   a = 10
   b = 20;
   if (a > b)
  {      int temp;
       temp = a;
       a= b;
       b = temp;
  }
    cout << a << b << temp;
   getch ( );
}
```

On compilation, the compiler prompts an error message:
Error in line no.13
The variable temp is not accessible.
The life time of a local variable is the life time of a block in its state of execution.
Local variables die when its block execution is completed.

- Local variables are not known outside their own code block. A block of code begins and ends with curly braces { }.
- Local variables exist only while the block of code in which they are declared is executing.

A local variable is created upon entry into its block and destroyed upon exit.

Identify local variables, in the Program-4.19 and also mention their scope.

4.7.2 Function scope

The scope of variables declared within a function is extended to the function block, and all sub-blocks therein.

```
// Program - 4.19
# include <iostream.h>
void main ( )
{ int flag = 1; a = 100;
  while (flag)
  {
    int x = 200;
   if (a > x)
   {  int j;
     -
    }
else
  { int h;
      -
  }}}
```

Local variable	Scope
1. x	Accessible in while block, and if blocks
	Accessible only in if (a>x) { } block
2. j	Accessible only in else block
3. k	

The variable flag of Program – 4.19 is accessible in the function main ( ) only. It is accessible in all the sub-blocks therein - viz, while block & if block.

The life time of a function scope variable, is the life time of the function block. The scope of formal parameters is function scope.

4.7.3 File scope

A variable declared above all blocks and functions (precisely above main ( )) has the scope of a file. The scope of a file scope variable is the entire program. The life time of a file scope variable is the life time of a program.

```
// Program - 4.20
// To demonstrate the scope of a variable

// declared at file level

# include <iostream.h>
# include <conio.h>
int i = 10;
void fun ( )
{ cout << i; }
void main ( )
{
   cout << i;
    while (i)
     {   -
         -
          -
      }
}
```

4.7.4 Scope Operator

The scope operator reveals the hidden scope of a variable. Now look at the following program.

```
// Program - 4.21
# include <iostream.h>
# include <conio.h>

int num = 15;

void main()
{
  clrscr();
  int num = 5;
  num = num + ::num;
  cout << num << '\t' <<
++::num;
  getch();
}
```

Have you noticed the variable **num** is declared both at file scope level and function main() level? Have you noticed the reference **::num ?** :: is called as scope resolution operator. It is used to refer variables declared at file level. This is helpful only under situations where the local and file scope variables have the same name.

4.7.5 Class scope

This will be discussed in Chapter 6.

Exercises

1. Construct function prototypes for descriptions given below:

a) procedural-function ( )
- is a function that takes no arguments and has no return value.

Solution -

```
     void procedural - function (void);
OR   void procedural - function ( ) ;
```

b) manipulative - function ( ) takes one argument of double type and returns int type.

Solution -

i. int manipulative - function (double); OR
ii. int manipulative - function (double d); OR
iii. manipulative - function (double)

c) fun-default ( ) takes two arguments, once with a default integer value, and the other float, has no return type

Solution -
void fun-default (float, int num = 10);

d) return - reference - fun ( ) takes two int arguments and return reference to int type

Solution -
int & return - reference - fun (int &, int &);

e) multi-arguments ( ) that takes two arguments of float, where the 1st argument is P1 should not be modified, and the 2nd argument is of reference type. The function has no return type.

Solution -
void multi - arguments (float const pi, int & a);

2. Identify errors in the following function prototypes:

a) float average (a, b);
b) float prd (int a,b);
c) int default-arg (int a = 2, int b);
d) int fun (int, int, double = 3.14);
e) void strings (char [ ]);

3. Given the function

```
void line (int times, char ch)
 {   cout << '\n';
    for (int i = 1; i < = times; i ++)
     cout << ch;
    cout << '\n';
}
```

Write a main ( ) function that includes everything necessary to call this function.

4. Write the scope of all the variables mentioned in this program.

```
# include <iostream.h>
float a, b ; void f1 (char);
int main ( )
{ char ch;
    -
     -

{ int i = 0;
-
-
-
}
  }
 void f1 (char g)
{ short x, y ;
 }  =
```

<u>Solution:</u>

a,b - file scope

ch - function scope - main ( )

i - scope within its block

x,y,g -function scope - f1 function

4. Identify errors in the following programs:

a)

```
# include <iostream.h>

xyz (int m, int n)
{ int m = 10;
     n = m * n;
     return n;
}
 void main( )
 { cout << xyz (9,27) ;}
```

<u>Solution</u>:

The variable ‘m’ is declared with function block, which is not permitted.

b)

```
# include <iostream.h>

void xyz ( );
void main ( )

{ int x = xyz ( ) ; }

void xyz ( )

{ return '10' ; }
```

Solution:

Function declared as void type, cannot have a return statement, hence the function call cannot be part of an expression

c)

```
# include <iostream.h>

void counter (int & a)
{ ++ a;}

void main ( )
{counter (50); }
```

Solution:

The actual parameter cannot be passed in the form of a value, as the formal parameter is of reference type

5. What will be the output of the following programs?

a)

```
# include <iostream.h>
int val = 10;
divide (int) ;
void main ( )
{int val = 5;
val = divide (::val/val);
cout << :: val<<val;
}
divide (int v)
{ return v/2;}
```

Solution: 101

b)
```
     # include <iostream.h>
    divide (int v)
    { return v / 10;}
             void main ( )
     {int val = -1;
    val = divide (400) = = 40;
cout << "\n Val." << val;
     }
```

Solution: 1 - Working

i) divide (400) yields
   a value 40
ii) divide (400) == 40 is
   interpreted as
   40 ==40 since the
   condition is true
   val gets 1

c)
```
     # include <iostream.h>
    int incre (int a)
    { return a++; }
    void main ( )
    {int x = 10; x = incre (x); cout << x;}
```

Solution: 10

d)
```
     # include <iostream.h>
    # include <iostream.h>
    void line( )
    {static int v = 5;
       int x = v - - ;
        while (x)
        {cout << ' * ' ; x — ;
         }
        cout << 'In';
    }
    void main ( )
    { clrscr ( );
         for (int i = 1; i < = 5; i ++
             line ( ) ;
         getch ( );
    }
```

Solution:

```
* * * * *
* * * *
* * *
* *
```

e)

```
 # include <iostream.h>
first (int i)
{ return i++; }
second (int x)
{ return x —; }
void main ( )
{ int val = 50;
    val = val * val/val
    val = second (val);
    val = first (val);
    cout << "\n Val: " << val;
}
```

<u>Solution</u>:

Val : 50

6. Program writing….

a) Write a program in C++ to define a function called float cube (int, int, int). Write main ( ) function, to test the working of cube ( ).

b) Define a function unsigned long it factorial (int);

The factorial of a number is calculated as follows: For example Factorial of 5 is calculated as 1 x 2 x 3 x 4 x 5
Write a main ( ) function to calculate the factorial (n).

c) Define a function called as
char odd - even - check (int);

The function should return 'E' if the given number is even, otherwise 'O'. Write a main ( ) to test and execute the function odd-even-check (int) and also print relevant message.

d) Define a function int prime (int);

The function should return a value 1, if the given number is prime, otherwise -1. Write a main ( ) to test and execute the function and also print relevant message.

CHAPTER 5

STRUCTURED DATA TYPE - ARRAYS

5.1 Introduction

An array in C++ is a aderived data type that can hold several values of the same type.

Processing a collection of data values by reading the data items individually and then processing each item may be very cumbersome and awkward if data is large. For example, consider the following situations:

1. To determine the largest number in the given set of numbers:

   a) if the set comprises of two numbers then the comparisons would be :

   ```
   if (a > b)
         max = a;
   else
         max = b;
   ```

   b) if the set comprises of three numbers then the comparisons would be :

   ```
   if (a > b) && (a > c)
         max = a;
   else if (b > c)
         max = b;
   else
         max = c;
   ```

c) if the given set comprises of 4 numbers then the comparisons would be :

```
  if (a > b && a > c && a > d)
          max = a;
else if (b > c && b > d)
          max = b;
else if (c > d)
          max = c;
else
          max = d;
```

Have you noticed the increase in comparisons, as the numbers increase??

In fact, handling large data becomes unwieldy, if one has to adopt the above methods for processing data.

Now look at this:

```
int a [4] = { 10, 40, 30, 20}; max = 0 ; i = 0;
                for (; i < 4; i ++)
                if a [i] > max
                        max = a [i] ;
                cout << “\n The largest number is” << max;
```

The above program code determines the largest value in the given list of numbers, i.e., 10, 40, 30, 20, thus storing 40 in max. Have you noticed the construct of if statement? In order to handle large data with ease, elements belonging to the same data type are decaled as ARRAYS.

An array is a collection of variables of the same type that are referenced by a common name.

Arrays are of two types:

One dimensional:	comprising of finite homogenous elements
Multi dimensional:	comprising of elements, each of which is itself a one- dimensional array

5.2 Single Dimension Array

These are suitable ways for processing of lists of items for identical types. An array is declared as follows:

```
int  num_array [5];
```

Syntax:

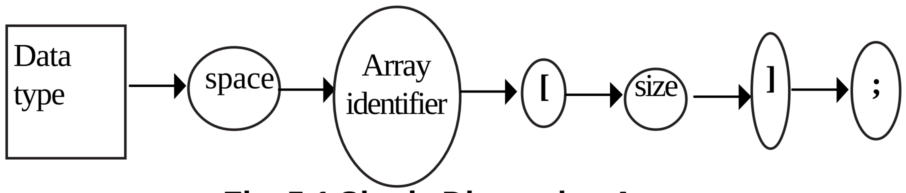


Fig. 5.1 Single Dimension Array

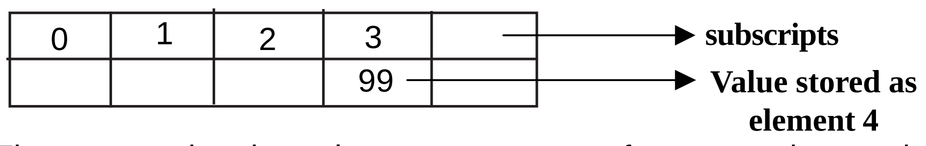
The size of the array should always be positive. The declaration int num_array [5]; is interpreted as **num_array is a one-dimensional array, that stores 5 integer values.** Each element of the array is accessed by the array name and the position of the element in the array. For example, num-array [3] = 99, stores the value 99 as the 4th element in the num_array.

num_array is the array identifier and **[3] is** subscript/position of the element

Memory allotted for num_array is 10 bytes, as it stores 5 integer element (memory required for one integer is 2 bytes hence 5 x 2 = 10 bytes).

Memory allocation is as follows:

num_array - identifier



The array subscripts always commences from zero, hence the subscripts for the variable num-array is between 0-4. The statement num-array [5] is invalid, because the valid subscripts are only between 0-4. The other valid examples of array declaration are:

i. int array [100];
ii. float exponents [10];
iii. char name [30];
iv. const i = 10;
    double val [i];
v. int days [ ] = {1,2,3,4,5,6,7};

In example [iv], the size of the array val is indicated through a constant variable i. In example [v], the size of the array days [ ] is indirectly indicated as 7. Can you guess how? Yes, the size is determined through the initialization statement -{1,2,3,4,5,6,7}. 7 elements determine the size of the array. Now look at the Program – 5.1 that demonstrates basic operations on arrays.

Examples of array processing:

cin >> number [4]	-	Reads fifth element
cout << number [subscript]	-	displays the element as indicated by subscript
number [3] = number [2]	-	assigns the contents of the 3rd element of the array to its 4th element
number [3] ++	-	increments the value stored as 4th element by 1
number [++ a] = 10	-	assigns the value 10 to the element as indicated by ++a

```
// Program - 5.1
// arrays and basic manipulations on
// arrays
# include <iostream.h>
# include <conio.h>

int a[5],ctr = 0, sum;

void main()
{
    for(;ctr<5;ctr++)
   {
      cout << “\nEnter value ..”;
      cin >> a[ctr];
   }
   // generating sum of all the elements
   // stored in array
   for(ctr = 0, sum = 0; ctr<5;ctr++)
    sum+= a[ctr];
   clrscr();
   // display values stored in array
   for(ctr = 0; ctr < 5; ctr++)
    cout <<‘\t’ << a[ctr];
   cout << “\nSum of the elements ...”
<< sum;
   getch();
}
```

```
// try out
// Program – 5.2

# include <iostream.h>

# include <conio.h>

char ch [ ] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’};
void main ( )
{
        for (int i = 0; i < 5; i++)
             cout << ch[ i];

   for (j=4; j>=0; j—)
        cout << ch [j];

 getch();
}
```

```
Output diaplayed :
 abcdeffedcba
```

```
// try out
// Program - 5.3
# include <iostream.h>
# include <conio.h>

void main ( )
{
int even [3] = {0, 2, 4}; int reverse [3];
for (int i=0, int j = 2; i<3; i++, j —)
       reverse [j] = even [i];
       clrscr ( );
for    (i=Ô;    i<3,    i++)
        cout ‘\t’ << even [i] << ‘\t’ << reverse [i] << ‘\n’;
       getch ( );
       }
```

Output of Program - 5.3

```
0                4
2                2
4                0
```

```
// try out
//Program - 5.4
# include <iostream.h>
# include <conio.h>
void main ( )
{
   int x[5] = {1,2,3,4,5}, y [5] = {5,4,3,2,1},
            result [5] = { 0,0,0,0,0 };
   int i= 0;
   while (i++ < 5)
            result [i] = x [i] - y [i];
   clrscr ( );
   cout << "\n The contents of the array are: \n";
   i= 0 ;
   do
   {
             cout << '\t' << x [i]
                      << '\t' << y [i]
                      << '\t' <<  result [i]<<'\n';
            i++;
   } while (i<5);
   getch ( );
}
```

Output of Program -5. 4

```
The contents of the array are:

1   -1      0
2   4       -2
3   3       0
4   2       2
5   1       4
```

```
//Try out
//Program - 5.5
# include <iostream.h>
# include <conio.h>
void main ( )
{
   int vector [ ] = {2, 4, 8, 10, 0};
   for(int i=4; i>2; i—)
       vector [i]= vector [i-1];
   clrscr( );
   cout << “\n Elements of array before insertion
\n”;
   for (i= 0; i<5; i++)
      cout << vector[i];
   vector [2] = 6;
   cout << “\n Elements after insertion \n”;
   for (i= 0; i<5; i++)
      cout << vector[i];
   getch ( );
}
```

Output of Program - 5.5

```
Elements of array before insertion
248810
Elements after insertion
246810
```

One can rearrange the data in a given array either in ascending or descending order. This process is called SORTING.

5.3 Strings

Strings are otherwise called as literals, which are treated as single dimensional array of characters. The declaration of strings is same as numeric array. For example,

i. char name [10];
ii. char vowels [ ] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’};
iii. char rainbow [ ] = VIBGYOR;

A character array (used as string) should be terminated with a '\0' (NULL) character. These arrays can be initialized as in the above examples, viz., (ii) and (iii).

```
// Program - 5.6
// Reading values into an array of characters

# include <iostream.h>
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main()
{
   clrscr();
   char name [30], address [30], pincode[7];
   cout << "\n Enter name ...";
   cin >> name;
   cout << "\n Enter address...";
   gets (address);
   cout << "\n Enter pincode ...";
   cin.getlin e (pincode, 7,'#');
   clrscr ( );
   cout << "\n Hello  " << name;
   cout << "\n Your address is ..." << address;
   cout << "\n Pincode  ....";
   cout.write (pincode, sizeof(pincode));
   getch ( );
}
```

In the above example Program - 5.6, the values for the variables name, address and pincode are read using

cin, gets ( ) and getline.

The instance cin, treats white space or carriage return (enter key) as terminator for string. For example,

cin >> name;

a) if the value for name is given as K V N Pradyot , then the value stored in name is only K, as white space is treated as string separator or terminator.
b) if the value for name is given as K.V.N.Pradyot , then the value stored in name is K.V.N.Pradyot.

To treat spaces as part of string literal, then one has to use gets ( ) defined in stdio.h or getline ( ) - a member function of standard input stream.

Syntax for gets ( ) is

    gets (char array identifier) or
    gets (char *)

Syntax for getline is

    cin.getline (char*, no.of characters, delimiter);

There are two methods to display the contents of string.

1. cout << name - this is similar to any other variable.
2. cout.write (pincode, 7); or cout.write (pincode, size of (pincode));

write ( ) is a member function of standard output stream, i.e., ostream. All member functions of a class, should be accessed through an object /instance of class. The two parameters required for write () function are identifier string characters, and no. of characters to be displayed.

For example,

```
//Program - 5.7
# include <iostream.h>
# include <conio.h>
void main()
{
   clrscr ( );
   char name[] = “Tendulkar”;
   int i=1;
   while (i<10)
   {
        cout.write (name, i);
        cout << ‘\n’;
        i++;
   }
   getch ( );
}
```

Output

```
T
Te
Ten
Tend
Tendu
Tendul
Tendulk
Tendulka
Tendulkar
```

String manipulators defined in string.h are described in Table 5.1.

Sl.No.	Function	Syntax	Purpose & value returned
1	strlen ( )	strlen (char *)	Returns the number of characters stored in the array.For example, name =
2	strcpy ( )	strcpy (char *,	Copies source string to target string. For example,strcpy
3	strcmp ( )	strcmp (char	Compares the two given strings, and returns 0 if strings are equal, value >0, if string 1 is greater than string 2. Otherwise value less than 0. For example, strcmp (“Abc”, “Abc”) returns 0strcmp (“Abc”, “abc”) returns a value less than 0strcmp

Table 5.1 String Functions

Strings czn be manipulated element by element like a char array. For example,

```
// Program - 5.8
# include <iostream.h>
# include <conio.h>
void main ( )
{
    clrscr();
    char name[] = “Pascal”, reverse[7];
    int i= 0;
    while (name[i] != ‘\0’)
        i++;
    reverse[i] = ‘\0’;
    –i;
    int j = 0;
    while (i>= 0)
        reverse [i–] = name [j++];
    cout << “\n The contents of the string are: “<< name;
    cout << “\n The contents of the reversed string ...”
         << reverse;
    getch ( );
}
```

What will be the output of the following program?

```
//Program - 5.9
# include <iostream.h>
# include <conio.h>
# include <string.h>
main()
{
   char word [ ] = “test”;
   int i=0, k = 4, j = 0;
   clrscr( );
   while (i < 4)
   {
       j = 0;
       while (j<=i)
          cout << word [j++];
       cout << ‘\n’;
       i++;
   }
  return 0;
}
```

5.4 Two-Dimensional Array

A two-dimensional array is an array in which each element is itself an array. For instance, an array marks [3] [4] is a table with 3 rows, and 4 columns.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

int makrs [3] [4] = {90, 70, 80, 0, 75, 95, 65, 0, 80, 90, 90, 0}; will create a table as;

	0	1	2	3
0	90	70	80	0
1	75	95	65	0
2	80	90	90	0

Note:

√ The number of elements in a 2-dimensional array is determined by multiplying the number of rows with number of columns. In this example - The array marks has 12 elements.

√ The subscripts always commence from zero. The subscript for rows is from 0 to 2, and for columns - 0 to 3.

√ An element in a 2-D array is referred as Marks [Row] [Column]. For example, marks [0] [3] = marks [0] [0] + marks [0] [1] + marks [0] [2] will sum up the marks of the 1st row, viz., 90, 70, 80.

A 2-D array is declared as:

Type array-id [Rows] [Columns];

Example:

1. int a[3] [2]- declares 3 rows and 2 columns for the array a
2. const i=5;
   float num [i] [3] - declares a 2-D table num with 5 rows and 3 columns
3. short fine ['A] ['E'] - declares a 2-D table of 65 rows and 69 columns

Note:

The dimensions (rows/columns) of an array can be indicated

1. using integer constants
2. using const identifier of integer or ordinal
3. using char constants
4. using enum identifiers

5.4.1 Memory representation of 2-D arrays

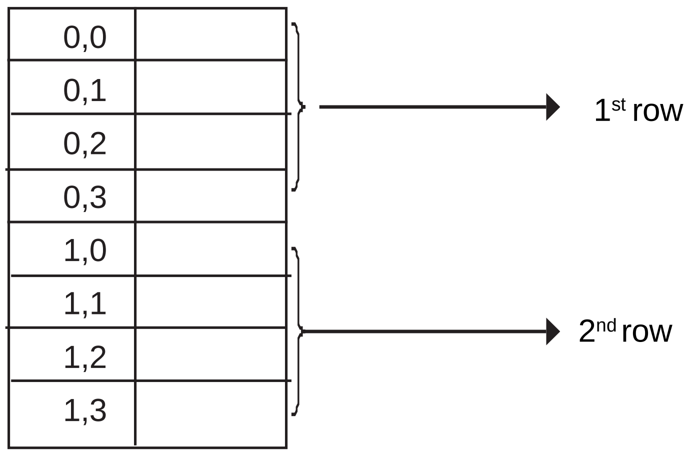
A 2-D array is stored in sequential memory blocks. The elements are stored either

1. row-wise manner (this method is called as row-major order)
2. column-wise manner (this method is called as column-major order)

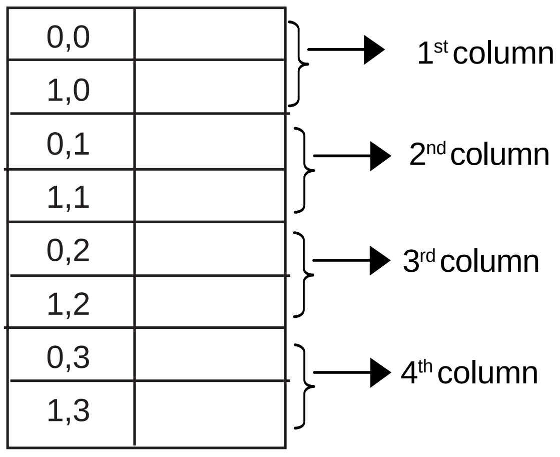
For example:

int sales [2] [4]; will be stored as follows:

In row-major order.



Column-major order



Have you noticed the position of sales [1] [0] in row-major order and column-major order?

The size of a 2-D array is calculated as follows:

Number of elements * memory req. for one element

For example - int sales [2] [4] the size will be calculated as follows:

Number of elements = Rows x columns - 2 x 4 = 8
$\therefore$ 8 x 2 (2 bytes is required for integer)
$\therefore$ size = 16 bytes

What will be the size of the array -

float num [4] [6];

Solution: 4 x 6 x 4 = 96 bytes

Consider the following array:

int num [4] [3] = {8, 7, 6, 4, 5, 8, 9, 7, 6, 1, 2, 3};

num

	0	1	2
0	8	**7**	6
1	4	**5**	8
2	**9**	7	6
3	1	**2**	3

Write the appropriate reference for the highlighted elements of the table num.

Solution:

num [0] [1]	-	(7)
num [1] [1]	-	(5)
num [2] [0]	-	(9)
num [3] [1]	-	(2)

Determine the number of elements in the following declaration:

a) int array [10] [12];
b) int x [ ] [2] = {0,1,1,2,2,3}

Solution:
a) 120 elements
b) 6 elements (rows = 3
columns = 2)

The size of first dimension (first index) is optional in array initialization.

```
            #include <iostream.h>
            # include <conio.h>
            #include <iomanip.h>
            void accept (int s [3] [4], int total)
            { int r = 0;  c = 0;
                      for (; r < 3; r ++)
                      {cout << "\n Month: " << r+1;
                          for (; c < 4; c++)
                              {cout << '\n' << c+1 <<
"Quarter..";
```

Arrays can be passed on as arguments to functions. The actual parameter is passed only by the identifier, ignoring dimensions.

Array parameters by default behave like a reference parameter, as the array identifier unlike other identifiers, represents the base address of the array. Hence, it results in sending an address to the formal parameter (like reference parameters).

```
// Program - 5.10
#include <iostream.h>
# include <conio.h>

void accept (int s[3][4], int &total)
{
   int r = 0, c = 0;
   for (; r < 3; r++)
   {
       cout << “\n Month: “ << r+1;
       for (c = 0; c < 4; c++)
       {
            cout << ‘\n’ << c+1 << “ Quarter..”;
            cin >> s[r][c];
            total += s[r][c];
       }
   }
}

void display (int d[3][4], int total)
{
   int r, c;
   clrscr ( );
   cout << “\nSales figures for 3 months & their
respective quarters..”;
   for (r = 0; r < 3; r++)
   {
        cout << “\n Month ...” << r+1;
        for (c = 0; c < 4; c ++)
          cout << ‘\t’ << d[r][c];
   }
   cout << “\n Total sales ..” << total;
}
void main ( )
{
    clrscr();
    int sales[3][4], t = 0;
    accept(sales,t);
    display(sales,t);
    getch();
}
```

Now look at the following program.

```
// Program - 5.11
# include <iostream.h>
# include <conio.h>
void accept (int a)
{
   cout << "\n Enter a number ..";
   cin >> a;
}

void display (int a)
{
   cout << '\n' << a;
}

void main ( )
{
   int num [2][2] ={{0,0},{0,0}}, r = 0, c = 0;
   clrscr ( );
   for (; r < 2; r++)
   for (; c < 2; c++)
      accept (num[r][c]);
   clrscr();
   for (r = 0; r < 2; r++ )
    for (c = 0; c < 2; c++)
            display (num[r][c]);
   getch();
}
```

Output: Assume data entered in accept () function is 1,2,3,4

```
0
0
0
0
```

Why do you think the array num is not updated with the values 1,2,3,4?

In this example, the parameter passed to void accept ( ) is element by element. Hence, it is treated as value parameter and not reference parameter.

Note: Only the array identifier represents the base address of an array.

Now, rewrite the above program with the change - void accept (int (a). On execution, if the same test data 1,2,3,4 is given, then the output displayed will be

```
1
2
3
4
```

5.4.2 Matrix

A matrix is a set of mn numbers arranged in the form of a rectangular array of m rows and n columns. Matrices can be represented through 2-D arrays.

Program 5.12 demonstrates to read values for 2 matrices and check their equality.

5.5 Array of Strings

An array of strings is a two-dimensional character array. The size of first index (rows) determines the number of strings and the size of second index (column) determines maximum length of each string. For example,

```
char day-names [7] [10]      =      {"Sunday",
                                     "Monday",
                                     "Tuesday",
                                     "Wednesday",
                                     "Thursday",
                                     "Friday",
                                     "Saturday"};
```

will appear in the memory as shown in Table 5.1.

```
//Program - 5.12
# include <iostream.h>
# include <conio.h>

void accept (int mat[3][3])
{
   clrscr();
   int r = 0, c = 0;
   for (; r < 3; r++)
   {
       cout << “\n Enter elements for row..” << r;
       for (c=0; c < 3; c++)
            cin >> mat[r][c];
   }
}

void main ( )
{
    int m1[3][3], m2[3][3];
    accept (m1);
    accept (m2);
    int i=0, j = 0, flag = 1;
    for (; i < 3; i++)
    {
        for (; j < 3; j ++)
          if (m1[i][j] != m2[i][j])
          {
             flag = 0;
             break;
          }

          if (flag == 0)
             break;
   }
   if  (flag)
      cout << “\n The matrices are equal ...”;
   else
      cout << “\n The matrices are not equal..”;
  getch ( );
}
```

	0	1	2	3	4	5	6	7	8	9	
0	S	u	n	d	a	y	\0				day-names [0]
1	M	o	n	d	a	y	\0				day-names [1]
2	T	u	e	s	d	a	y	\0			day-names [2]
3	W	e	d	n	e	s	d	a	y	\0	day-names [3]
4	T	h	u	r	s	d	a	y	\0		day-names [4]
5	F	r	i	d	a	y	\0				day-names [5]
6	S	a	t	u	r	d	a	y	\0		day-names [6]

Table 5.1 Array Elements in Memory

An individual string is accessed as

day-names [0], i.e., by specifying the 1st index only. A specific character or an element is accessed as day-names [0] [5], i.e., by specifying both 1st and 2nd indices.

Attaching teh null character (\0) to each string literal is optional. Even if we omit it, the C++ compiler will automatically attach it.

Exercises

1. Why do the following snippets show errors?

a) int a [5.5]

Dimension of an array should be only an integer

b) float f [3] = {1.0, 2.0};

This will not show any error. But the number of elements is one less than the size of the array.

c) float num [A];

Dimension of an array should be explicitly mentioned. Here, the identifier A does not have a value. The statement may be rewritten as

```
float num ['A']
        Or
const A = 10;
float num [A];
```

d) char a [3] [ ] = {"one", "two", "three"};

The option for omitting the size of an array is given only for 1st index and not the second index. The statement may be rewritten as
char a [ ] [6] = {"one", "two", "three"};

e) char ch [1] = 's';

Character Array should be initialized using double quotes. The correct statement is

```
char ch [1] = "s"
        Or
char ch [1] = {"s"}
```

f)
```
char test [4];
test = {"abc"};
```

An array cannot be assigned in this manner. The correct statements are:

```
char test [4] = "abc" - initializing at the time of declaration
        or
char test [4];
strcpy (test, "abc");
```

g) int num [ ] = {1,2,3}, num2 [3];
num2 = num;

Group assignment of array is not allowed. One can assign only component by component.

h) int num [3];
cout << num;
cin >> num;

Such I/O operations are not allowed on arrays. Manipulation of arrays is possible only by specific direction to its elements or components, i.e.

cout << num [1] / cin >> num [1]

i) int roster = {1,2,3,4};

The variable roster cannot take more than one value. Hence, the statements should be as:

int roster = 10; or int roster [ ] = {1,2,3,4};

2. What would be the contents of the array after initialization?

a) int rate [ ] = {30,40,50};

b) char ch [6] = {“ bbbb\0 “ }
ch [0] = ‘C’;
ch [4] = ‘T’;
ch [3] = ‘A’;

Note b indicates white/blank space.

c) char product-list [ ] [5] = {“nuts”, “Bolts”, “Screw”};

Solution:

a - rate [0] = 30, rate [1] = 40. rate [2] = 50
b - ch [0] = 'C', ch [1] = ' '; ch [2] = ' ', ch [3] = 'A',
ch [4] = 'T', ch [5] = '\ 0",
c - product-list [0] = "Nuts \ 0", product-list [1] = "Bolts \ 0",
product-list [2] = "Screw\0"

3. What would be the output of the following programs?

a)

```
 # include <iostream.h>
void main ( )
{char ch [ ] = {"END \ 0"S};
    cout << ch;
}
```

Solution: END

b)

```
 # include <iostream.h>
void main ( )
{int a [ ] = {1,2,3,4,5};
for (int i = 0, i < 4, i++)
a[i+1] = a[i];
for (i= 0; i<5; i++)
cout << '\n' << a[i];
```

Solution:

	0	1	2	3	4
a	1	2	3	4	5

i = 0 - a [1] = a [0]

will be as follows:

	0	1	2	3	4	
a	1	2	3	4	5	

i = 1 - a [2] = a [1]
i = 2 - a [3] = a [2]
i = 3 - a [4] = a [3]

Hence, the array contents.

The output will be displayed as:

```
1
1
1
1
1
```

c)
```
 # include <iostream.h>
# include <conio.h>
void main ( )
{ char name [ ] = {“Jerry \0”}; int k = 5;
        for (int i = 0 ;  i < 3; i ++, k —)
            name [k] = name [i];
         cout << name;
         getch ( );
}
```

Solution:

```
When i = 0  k = 5    name [5]   = name [0]
        i = 1  k = 4    name [4]   = name [1]
        i = 2  k = 3    name [3]   = name [2]
```

∴ the output will be displayed as

```
JerreJ
```

Program Writing

1. Write a program to declare and initialize an array called as int-array, that stores number 10,20,30,40 and 50. Display the sum of all the elements of int-array.

2. Write a program to declare an array of integers that can hold 10 values. Read the elements of the array from the user, and also display the contents in the reverse order.

3. Write a program to read a sentence into an identifier called as word from the user. Using while loop and switch statements, display the count of vowels present in the given sentence. For example:

   word [ ] = “The vowel count AEIOU aeiou”;
   Vowel count is 14

4. Write a program to create a MATRIX [3] [3]. Display the diagonal elements along with the sum of diagonal elements.

5. Write a program to read values for two matrices, viz., matrix A [4] [4], matrix B [4] [4]. Write program code to create sum-matrix [4] [4] that stores the sum of elements of matrix A and matrix B.

Example

matrix A	**matrix B**	**sum-matrix**
1 2 3 4	9 8 7 6	10 10 10 10
5 6 7 8	5 4 3 2	10 10 - -
9 1 2 3	1 9 8 7	- - - -
4 5 6 7	6 5 4 3	- - - -

CHAPTER 6

CLASSES AND OBJECTS

6.1 Introduction to Classes

The most important feature of C++ is the “Class”. Its significance is highlighted by the fact that Bjarne Stroustrup initially gave the name ‘C with Classes ‘. A class is a new way of creating and implementing a user defined data type. Classes provide a method for packing together data of different types. For Example:

```
class student
{
        char name[30];
        int rollno, marks1, marks2, total_marks;
};
```

The data variables, rollno, marks1, marks2, total_marks define the properties or features of a student ,thus packing together data of different types. The class data type can be further extended by defining its associated functions. These functions are also called as methods, as they define the various operations (in terms of accepting and manipulating data) that can be performed on the data.

In other words

✓ A class is a way to bind the data and its associated functions together

6.2 Specifying a class :

A class specification has two parts :

1) Class declaration
2) Class Function Definitions

```
// Program - 6.1
# include <iostream.h>
# include <conio.h>
class student
{
       private :
         char name[30];
         int rollno, marks1, marks2 ,total_marks;
       protected:
          void accept()
          {
              cout<<"\n Enter data name, roll no, marks 1 and
marks 2.. ";
              cin>>name>>rollno>>marks1>>marks2;
          }
          void compute()
          {
              total_marks = marks1+ marks2;
          }
          void display()
          {
                     cout<<"\n Name "<<name;
                     cout<<"\n Roll no "<<rollno;
                     cout<<"\n Marks 1.. "<<marks1;
                     cout<<"\n Marks 2.. "<<marks2;
                     cout<<"\n Total Marks.. "<< total_marks;
          }
        public:
        student()
        {
              name[0]='\0';
              rollno=marks1=marks2=total_marks= 0;
              cout<<"\n Constructor executed ...  ";
         }
         void execute()
         {
              accept();
              compute();
              display();
         }
};
void main()
{
       clrscr();
       student stud;
       stud.execute();
}
```

The form of class declaration is

General Form	With respect to the above example

```
class class-name
{
 private:
    variable declaration
    function declaration

 protected:
    variable decl.
    function decl.

 public:
    variable decl.
    function decl.
};
```
	```
	class student
	{ private;
	   char name [10];
	   int roll no, mark1, mark2, total marks;

	protected:
	   void accept( );
	   void compute( );
	   void display( );

	public:
	student( );
	void execute( );
	};
	```

✓ The keyword class specifies user defined data type class name

✓ The body of a class is enclosed within braces and is terminated by a semicolon

✓ The class body contains the declaration of variables and functions

✓ The class body has three access specifiers (visibility labels) viz., private , public and protected

✓ Specifying private visibility label is optional. By default the members will be treated as private if a visibility label is not mentioned

✓ The members that have been declared as private, can be accessed only from within the class

✓ The members that have been declared as protected can be accessed from within the class, and the members of the inherited classes.

✓ The members that have been declared as public can be accessed from outside the class also

6.3 Data Abstraction

The binding of data and functions together into a single entity is referred to as **encapsulation.**

The members and functions declared under private are not accessible by members outside the class, this is referred to as **data hiding**. Instruments allowing only selected access of components to objects and to members of other classes is called as **Data Abstraction.** Or rather Data abstraction is achieved through data hiding.

Data hiding is the key feature of object oriented programming (OOPS)

private	Accessible by only its own members and certain special functions called as **friend functions**
protected	Accessible by members of inherited classes
public	Access allowed by other members in addition to class member and objects

6.4 Data Members and Member Functions

Class comprises of members. Members are further classified as Data Members and Member functions. Data members are the data variables that represent the features or properties of a class. Member functions are the functions that perform specific tasks in a class. Member functions are called as methods, and data members are also called as attributes. Now look at the Table 6.1 where information is provided based on Program –6. 1 **class student. Classes** include special member functions called as constructors and destructors. These will be dealt in Chapter – 8 Constructors and Destructors.

student	The data type identifier **student** is also called as class tag
name, rollno, marks1, marks2, total	data members
public accept ( ) compute ( ) display ( ) execute ( ) student ( )	member functions or methods
stud	instance/object/variable of class student

Table 6.1 Class Student of Program - 6.1

6.5 Creating Objects

Look at the following declaration statement **student stud;** This statement may be read as **stud is an instance or object of the class student.**

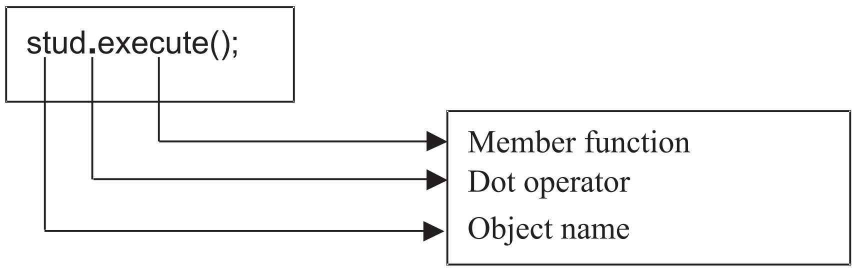
Once a class has been declared, variables of that type can be declared. 'stud' is a variable of type student ,student is a data type of class . In C++ the class variables are known as objects. The declaration of an object is similar to that of a variable of any basic type. Objects can also be created by placing their names immediately after the closing brace of the class declaration.

```
class student
{
        private:


          protected:


          public:

}stud;// stud is an object
```

```
class student
{
        private:
        ........
        protected:
        .......
         public:
};
void main()
{
   student s, s1[5];
}
the variables s and s1 are objects
or instances of the class stduent
```

6.6 Accessing Class Members

The members of a class are accessed using the dot operator. For example, the call statement to the function execute() of the class student may be given as:



The private data of a class can be accessed only through the member functions of its own class and certain special functions called as friend functions.

In the example class student, the data members name, marks1, marks2, rollno, total_marks are accessed only by the member functions accept(), display(), compute(). The objects declared outside the class cannot access members or functions defined under private or protected.

The member functions declared under public can be accessed by the objects of that class. The call statement stud.execute(); is a valid statement, as execute() is a member function defined under public visibility mode and hence can be accessed through the object stud defined outside the class. Where as the statements: stud.accept(), stud.compute(), stud.display(), stud.marks1 etc would force the compiler to throw error messages “not accessible”. Program –6.2 is another example that demonstrates the operation – addition of two numbers. This class wraps three integer variables, and its related member function to accept data, and perform addition. Since the variable sum is defined under public visibility mode, the object is accessing it.

```
//Program - 6.2
# include <iostream.h>
# include <conio.h>

class add
{
       private:
              int a,b;
       public:
              int sum;

              void getdata()
              {
                     a=5;
                     b=10;
                     sum = a+b;
              }
};

void main()
{
       add s;
       s.getdata();
       cout<<s.sum;
}
```

6.7 Defining methods of a class

```
class add
{
       int a,b;
       public:
              add()
              {
                     a='\0';                        Method 1
                     b='\0';
              }
       void display();
};
void add::display()
{
       int sum;
       sum = a+b;                                   Method 2
       cout<<sum;
}
```

In Method 1, the member function add() is declared and defined within class add.

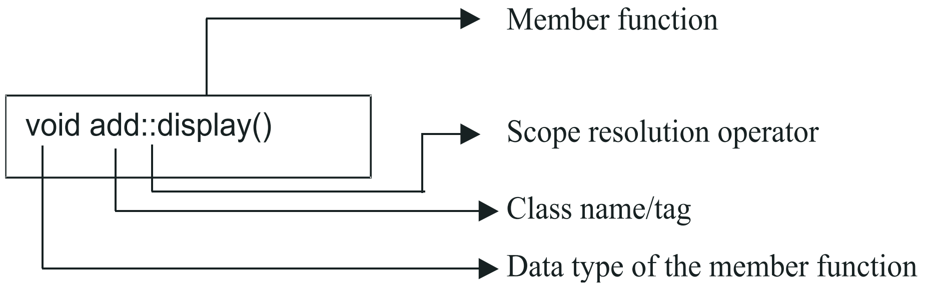
In Method 2, the member function display() is declared within the class, and defined outside the class.

Methods of a class can be defined in both ways. The members defined within the class behave like inline functions.

Member functions defined outside the class has the prototype as

type class_name :: function name();

For example:



The membership label class_name:: (add::) tells the compiler that the function function_name belongs to the class class_name. That is the scope of the function is restricted to the class specified in the function header.

The member function have some special characteristics that are often used in the program development .

- ✓ Several different classes can use the same function name. The ‘membership’ label will resolve their scope

- ✓ Member functions can access the private data of a class. A non-member function cannot do so.
- ✓ A member function can call another member function directly, without using the dot operator. (This is called as nesting of member functions)
- ✓ The member functions can receive arguments of a valid C++ data type. Objects can also be passed as arguments
- ✓ The return type of a member function can be of object data type
- ✓ Member functions can be of static type

6.8 Memory allocation of objects

The member functions are created and placed in the memory space only when they are defined as a part of the class specification. Since all the objects belonging to that class use the same member function, no separate space is allocated for member functions when the objects are created. Memory space required for the member variables are only allocated separately for each object. Separate memory allocations for the objects are essential because the member variables will hold different data values for different objects

Look at the following class declaration:

```
    class product
{
        int code, quantity;
        float price;
        public:
            void assign_data();
            void display();
};
void main()
{
    product p1, p2;
}
```

Member functions assign_data() and display() belong to the common pool in the sense both the objects p1 and p2 will have access to the code area of the common pool.

Memory for Objects for p1 and p2 is illustrated:

objects	Data members	Memory alloted
p1	Code, quantity and price	8 bytes
p2	Code,quantity and price	8 bytes

Table 6.2 Memory Allocation for Objects

Member functions of a class can handle arguments like any other non member functions as illustrated in Program - 6.3.

6.9 Static Data Members

A data member of a class can be qualified as static

The static member variable

- ✓ Is initialized to zero, only when the first object of its class is created . No other initialization is permitted
- ✓ Only one copy of the member variable is created (as part of the common pool) and is shared by all the other objects of its class type
- ✓ Its scope or visibility is within the class but its lifetime is the lifetime of the program.

```
      // Program - 6.3
      #include<iostream.h>
      #include<conio.h>
      class product
      {
            int code, quantity;
            float price;
        public:
           void assign_data( int c, int q, float p)
            {
               code = c;
              quantity = q;
               price = p;
            }

            void display()
            {
               cout<<"\n Code : "<<code;
               cout<<"\n Quantity :"<<quantity;
               cout<<"\n Price :  "<< price;
            }
};
void main()
{
            product p;
            p.assign_data( 101, 200, 12.5);
            p.display();
}
```

```
    // Program - 6.4
    // To demonstrate the use of static member variables

    #include<iostream.h>
    #include<conio.h>
    class simple_static
    {
                int a,b,sum;
                static int count;
        public:
                void accept()
                {
                        cout<<"\n Enter values.. ";
                        cin>>a>>b;
                        sum = a+b;
                        count++;
                }
                void display()
                {
                        cout<<"\n The sum of two numbers … "<<sum;
                        cout<<"\n This is addition… "<<count;
                }
        };
int static_simple count=0;
void main()
{
        simple_static p1,p2,p3;
        p1.accept();
        p1.display();
        p2.accept();
        p2.display();
        p3.accept();
        p3.display();
}
```

OUTPUT :

```
Enter  values …… 10 20
The sum of two numbers ……….. 30
```
This is addition   1

```
Enter values……… 5 7
The sum of two numbers……………12
This is addition 2

Enter values……….. 9 8
The sum of two numbers ……………17
This is addition 3
```

The static variable count is initialized to zero only once. The count is incremented whenever the sum of the two numbers was calculated. Since the function accept() was invoked three times, count was incremented thrice and hence the value is 3. As only one copy of count is shared by all the three objects, the value of count is set to 3. This is shown in Fig. 6.2.

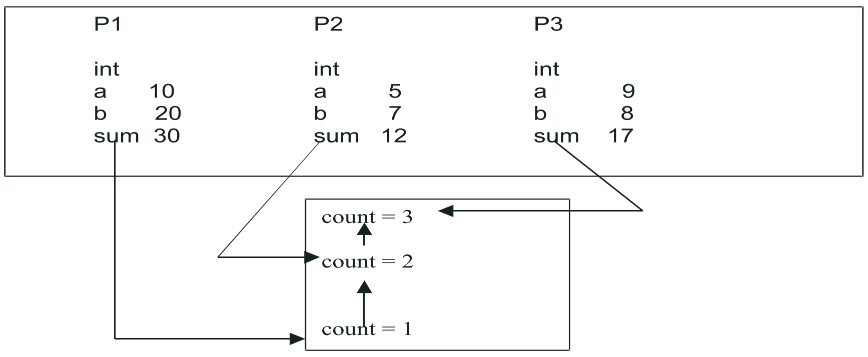


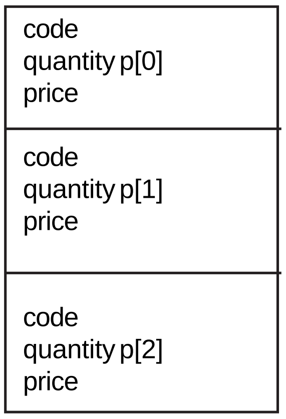
Fig. 6.2 Static Member Variable - count

The initial value to a static member variable is done outside the class.

6.10 Arrays of objects

Consider the following class definition and its corresponding memory allocation:

```
class product
{
    int code,quantity;
    float price;
public :
   void assign_Data();
    void display();
} p[3];

void main()
{
  p[0].assign_Data();
   p[0].display();
}
```



Exercises

I. Identify and correct the errors in the following

```
class x
{
        public:
                int a,b;
                void init()
                {
                        a =b = 0';
                }
                int sum();
                int square();
};
int sum()
{
        return a+b;
}
int square()
{
        return sum() * sum()
}

Solution :
int x::sum() and int x::square()
```

II

```
#include<iostream.h>
class simple
{
                int num1, num2 , sum = 0;
        protected:
                accept()
                {
                        cin>>num1>>num2;
                }
        public:
                display()
                {
                        sum = num1 + num2;
                }
};
void main()
{       simple s;
        s.num1=s.num2= 0;
        s.accept();
        display();
}
```

Solution:

1) The member sum cannot be initialized at the time of declaration
2) The member variable num1 and num2 cannot be accessed from main() as they are private
3) s.accept() is invalid. The method accept() is defined under protected
4) display() should be invoked through an object

III

```
#include<iostream.h>
#include<conio.h>
class item
{
        private:
                int code,quantity;
                float price;
                void getdata()
                {
                        cout<<"\n Enter code, quantity, price ";
                        cin>>code>>quantity>>price;
                }
                public:
                        float tax='\0';
                        void putdata()
                        {
                                cout<<"\n Code : "<<code;
                                cout<<"\n Quantity : "<<quantity;
                                cout<<"\n Price :  "<<price;
                                if( quantity >100 )
                                        tax = 2500;
                                else
                                        tax =1000;
                                cout<<“ \n Tax :”<<tax;
                        }
};

void main()
{ item i;   }
```

Complete the following table based on the above program

Memory allocation for instance i	Private data members	Public data members	Methods or data members that can be accessed by i

IV

1) Define a class employee with the following specification private members of class employee

   empno- integer
   ename – 20 characters
   basic – float
   netpay, hra, da, float

calculate () – A function to find the basic+hra+da with float return type
public member functions of class employee
havedata() – A function to accept values for empno, ename, basic, hra, da and call calculate() to compute netpay

dispdata() – A function to display all the data members on the screen

2) Define a class MATH with the following specifications

private members
num1, num2, result – float
init() function to initialize num1, num2 and result to zero

protected members
add() function to add num1 and num2 and store the sum in result
prod() function to multiply num1 and num2 and store the product in the result

public members
getdata() function to accept values for num1 and num2
menu() function to display menu

1. Add…
2. Prod…

invoke add() when choice is 1 and invoke prod when choice is 2 and also display the result.

CHAPTER 7

POLYMORPHISM

7.1 Introduction

The word polymorphism means many forms (poly – many, morph – shapes). In C++, polymorphism is achieved through function overloading and operator overloading. The term overloading means a name having two or more distinct meanings. Thus an '**overloaded function'** refers to a function having more than one distinct meaning. Function overloading is one of the facets of C++ that supports object oriented programming.

7.2 Function overloading

The ability of the function to process the message or data in more than one form is called as function overloading.

Consider the situation wherein a programmer desires to have the following functions

```
area_circle()       // to calculate the area of a circle
area_triangle()     // to calculate the area of a triangle
area_rectangle()  // to calculate the area of a rectangle
```

The above three different prototype to compute area, for different shapes can be rewritten using a single function header in the following manner

```
float area ( float radius);
float area ( float half, float base, float height );
float area ( float length , float breadth);
```

Now look at the ease in invoking the function area(…) for any of the three shapes as shown in Program - 7.1

```
// Program - 7.1
// to demonstrate the polymorphism - function overloading

#include<iostream.h>
#include<conio.h>

float area ( float radius )
{      cout << “\nCircle …”;
      return ( 22/7 * radius * radius );
}
float area (float half, float base, float height)
{      cout << “\nTriangle ..”;
      return (half* base*height);
}


float area ( float length, float breadth )
{       cout << “\nRectangle …”;
      return ( length *breadth ) ;
}

void main()
{
      clrscr();
      float r,b,h;
      int choice = 0 ;
      do
      {
             clrscr();
             cout << “\n Area Menu “;
             cout << “\n 1. Circle ... “;
             cout << “\n 2. Traingle ...“;
             cout << “\n 3. Rectangle ... “;
             cout <<“\n 4. Exit ... “;
             cin>> choice;
             switch(choice)
             {
                    case 1 :
                           cout << “\n Enter radius ... “;
                           cin>>r;
                           cout<<“\n The area of circle is ... “
                              << area(r);
                           getch();
                           break;
                    case 2:
                           cout<< “\n Enter base, height ... “;
                           cin>>b>>h;
                           cout<<“\n The area of a triangle is .. “
                                << area (0.5, b, h);
                           getch();
                           break;
                    case 3:
                         cout<< “\n Enter length, breadth.. “;
                          cin>>h>>b;
                          cout<<“\n The area of a rectangle is ... “
                               << area(h,b);
                        getch();
                        break;
              }
             }while (choice <=3);
}
```

Have you noticed the function prototypes for all the 3 functions? The prototypes are:

```
float area ( float radius );
float area (float half, float base, float height)
float area ( float length, float breadth );
```

How do you think each function 'area' definition is differing from one and another ? Yes, each function prototype differs by their number of arguments. The first prototype had one argument, second one 3 arguments and the third one had 2 arguments. In the example we have dealt , all the three functions has float type arguments. It need not necessarily be this way. Arguments for each prototype can be of different data type . Secondly the number of arguments for each function prototype may also differ. The following prototypes for function overloading is invalid. Can you tell why is it so ?

Function Prototype

```
void fun(int x);
void fun(char ch);
void fun(int y);
void fun(double d);
```

Invalid prototype

void fun(**int x**);
void fun(**int y**);

Both the prototypes have same number and type of arguments. Hence it is invalid.

How are functions invoked in function overloading?

The compiler adopts BEST MATCH strategy. As per this strategy, the compiler will

- ✓ Look for the exact match of a function prototype with that of a function call statement

- ✓ In case an exact match is not available, it looks for the next nearest match. That is, the compiler will promote integral data promotions and then match the call statement with function prototype.

For example, in the above example (program –1) we have **float area(float radius)** with area(r) where the parameter **'r'** should be of float type. In case, the variable 'r' is of integer type, then as per integral promotions integer constant/variable can be mapped to char, or float or double. So, by this strategy the area(r) will be mapped to area(float radius).

Integral promotions are purely compiler oriented. By and large integral promotions are as follows:

- ✓ char data type can be converted to integer/float/double
- ✓ int data type can be converted to char/double/float
- ✓ float data type to integer/double/char
- ✓ double data type to float or integer

Now based on the following call statements to area() of Program – 7.1 can you tell as to what will be the output?

Function call statement	Output displayed
area(5.0)	
area(0.5,4.0,6.0)	
area(3.0,4.5)	

Rules for function overloading

1) Each overloaded function must differ either by the number of its formal parameters or their data types
2) The return type of overloaded functions may or may not be the same data type
3) The default arguments of overloaded functions are not considered by the C++ compiler as part of the parameter list
4) Do not use the same function name for two unrelated functions

Improper declarations leading to conflict in a function call statement is shown below.

```
void fun ( char a, int times)
{
      for (int i=1; i<=times;i++)
                  cout<<a;
}
void fun( char a= '*', int times )
{
      for(int i=1;i<=times;i++)
                  cout<<a;
}
void fun( int times)
{
      for(int i=1; i<=times ;i++)
                  cout<<'@';
}
void main()
{
      fun ( '+', 60);
      fun(60);
}
```

When the above program is compiled, two errors will be flagged:

- ✓ Conflict between fun(char a, int times) and fun(char a='*', int times)
- ✓ Conflict between fun(char a='*', int times) and fun (int times)

The call statement fun('+', 60) can be matched with fun (char a, int times) and fun (char a='*', int times)

The call statement fun(60) can be matched with fun (char a='*', int times) and fun (int times)

Overload a function with the help of different function definitions having a unique parameter list. That is, the parameter list differ either by number or types.

7.3 Operator Overloading

The term operator overloading, refers to giving additional functionality to the normal C++ operators like +,++,-,—,+=,-=,*.<,>. The statement sum = num1 + num2 would be interpreted as a statement meant to perform addition of numbers(integer/float/double) and store the result in the variable sum. Now look at the following statement:

```
name =  first_name + last_name;
```

where the variables name, first_name and last_name are all character arrays.Can one achieve concatenation of character arrays using '+' operator in C++? The compiler would throw an error stating that '+' operator cannot handle concatenation of strings. The user is forced to use **strcat()** function to concatenate strings. Won't it be a lot easier if one is permitted to use '+' operator on strings as used for number data type? The functionality of '+' operator can be extended to strings through **operator overloading**.

Look at the following example:

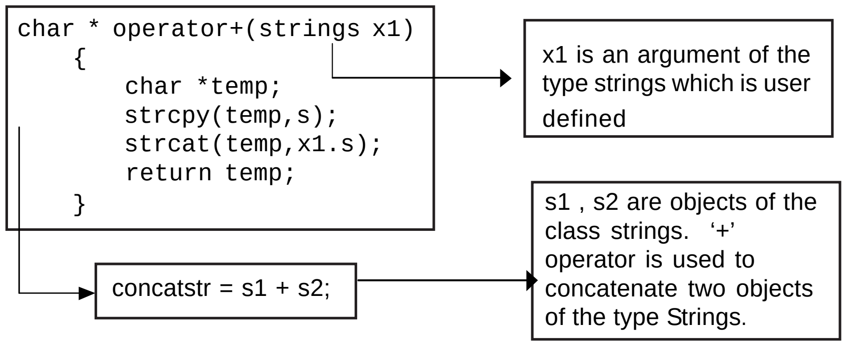
```
// Program -7.2 – OPERATOR OVERLOADING
# include <iostream.h>
# include <conio.h>
# include <string.h>

class strings
{
   char s[10];
   public :
   strings()
    {
          s[0] = '\0';
    }

   strings(char *c)
    {
         strcpy(s,c);
    }

   char * operator+(strings x1)
    {
         char *temp;
         strcpy(temp,s);
         strcat(temp,x1.s);
         return temp;
    }
};
```

```
void main()
{
clrscr();
strings s1("test"),s2(" run\0");
char *concatstr ;
concatstr = s1 + s2;
cout << "\nConcatenated string ..."
     << concatstr;
getch();
}
```

The statement **concatstr = s1 + s2** merges two strings, as the operator '+' is given additional function through the member function:

char * operator + (strings x1)

The member function **char * operator + (strings x1)** takes x1 as the argument. It may be viewed as:



```
char * operator+(strings x1)
    {
        char *temp;
        strcpy(temp,s);
        strcat(temp,x1.s);
        return temp;
    }
```

x1 is an argument of the type strings which is user defined

```
concatstr = s1 + s2;
```

s1 , s2 are objects of the class strings. '+' operator is used to concatenate two objects of the type Strings.

Fig. 7.1 demonstrates the association of variables and their values.

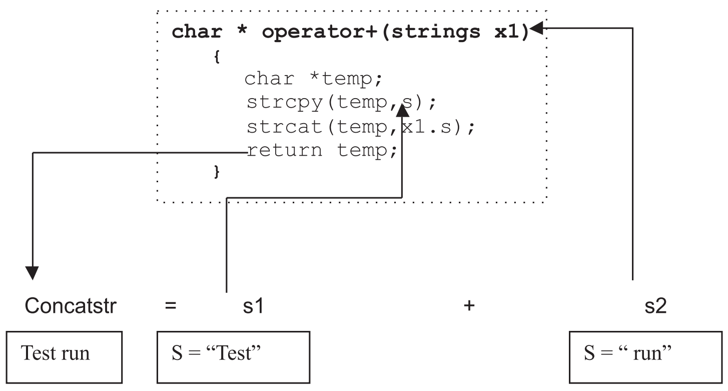


Fig. 7.1 Association of Variables and Values

Operator overloading provides:

- ✓ New function definitions for basic C++ operators like +, *, -, ++, - -, >, <, += and the like. One cannot overload C++ specific operators like membership operator (.), scope resolution operator (::), sizeof operator and conditional operator.

- ✓ The overloaded function definitions are permitted for user defined data type.

- ✓ Operator functions must be either member functions or friend functions. (Friend functions is beyond the scope of this book)

- ✓ The new definition that is provided to an operator does not overrule the original definition of the operator. For example, in the above program – OPERATOR OVERLOADING the '+' operator has been used to merge two strings. In the same program one can also perform addition of numbers in the usual way. The compiler applies user defined definition based on the style of call statement. That is the statements cout << 5 + 10 will display the result as 15 (original definition of ‘+’ is applied), where as concatstr = s1 + s2 will invoke the member function **char * operator + (strings s1)** as the operands provided for the ‘+’ operator are s1 and s2 which are the objects of the class strings.

The process of overloading involves:

- ✓ Create a class that defines the data type that is to be used in the overloading operations
- ✓ Declare the operator function **operator ()** in the public part of the class.
- ✓ Define the operator function to implement the required operations.

The following examples demonstrate the ease of using operators with user defined data types – objects.

Program – 7.3 demonstrates as to how one can negate the data members of a class using the operator – (minus)

```
// Program - 7.3
# include <iostream.h>
# include <conio.h>

class negative
{
   int i;
   public :
   void accept()
   {
       cout << “\nEnter a number ...”;
         cin >> i;
   }
   void display()
   {
         cout << “\nNumber ...”<<i;
   }
```

```
 void operator-()
  {
       i = -i;
  }
};

void main()
{
      clrscr();
      negative n1,n2;
      n2.accept();
      -n2;
      n2.display();
      getch();
}
```

The function void **operator –()** simply negates the data members of the class as one would do with a normal variable as follows: sum = **-** num1;

Look at the following program and answer the questions:

```
// Program – 7.4
# include <iostream.h>
# include <conio.h>

class distance
{
   int feet,inches;
   public :
   void distance_assign(int f, int i)
   {
        feet = f;
        inches = i;
   }

   void display()
   {
       cout << “\nFeet   : “ << feet
              << “\tInches : “ << inches;
   }

   distance operator+(distance d2)
   {
        distance d3;
        d3.feet = feet + d2.feet;
        d3.inches = (inches + d2.inches) % 12;
        d3.feet += (inches + d2.inches)/12;
        return d3;
   }
};

void main()
{
    clrscr();
    distance dist_1,dist_2;
    dist_1.distance_assign(12,11)
    dist_2.distance_assign(24,1);
    distance dist_3 = dist_1 + dist_2;
    dist_1.display();
    dist_2.display();
    dist_3.display();
    getch();
}
```

1. Identify the operator that is overloaded.

2. Write out the prototype of the overloaded member function.

3. What types of operands are used for the overloaded operator?

4. Write out the statement that invokes the overloaded member function.

Program-7.5 demonstrates the overloaded functions of += and -=

```
//Program-7.5
// operator overloading

# include <iostream.h>
# include <conio.h>
# include <string.h>

class library_book
{
   char name[25];
   int code,stock;

   public :

   void book_assign(char n[15],int c,int s)
  {
        strcpy(name,n);
        code = c;
        stock = s;
  }

   void display()
  {
         cout << "\n   Book details ....";
         cout << "\n   1. Name        ....." << name;
         cout << "\n   2. Code        ....." << code;
         cout << "\n   3. Stock       ....." << stock;
  }

   void operator +=(int x)
  {
       stock += x;
  }

   void operator -=(int x)
  {
        stock -= x;
  }
};
```

```
class library_cdrom
{
 char name[25];
 int code,stock;

 public :

 void cdrom_assign(char n[15],int c,int s)
 {
            strcpy(name,n);
            code = c;
            stock = s;
 }

 void display()
 {
            cout << "\n  CD ROM details ....";
            cout << "\n  1. Name     ....." << name;
            cout << "\n  2. Code     ....." << code;
            cout << "\n  3. Stock    ....." << stock;
 }

 void operator +=(int x)
 {
           stock += x;
 }

 void operator -=(int x)
 {
            stock -= x;
 }
};


void main()
{
  library_book book;
  library_cdrom cdrom;

  book.book_assign("Half Blood Prince",101,55);
  cdrom.cdrom_assign("Know your Basics",201,50);

   char choice,borrow;

do
  {
            cout << "\nBook,cdrom,exit<b/c/e> ...";
              cin >> choice;
              if (choice != 'e')
                {
                     cout << "\nBorrow/Return <b/r> ...";
                     cin >> borrow;
                }
   switch (choice)
           {
     case 'b' :
             switch (borrow)
             {
                case 'b' : book += 1;break;
                case 'r' : book -= 1;break;
             }
             book.display();
             break;
     case 'c' :
             switch (borrow)
             {
                case 'b' : cdrom += 1;break;
                case 'r' : cdrom -= 1;break;
             }
             cdrom.display();
             break;

    case 'e' : cout << "\nTerminating ..";
                                  break;
   }
   } while (choice != 'e');
   getch();
  }
```

Have you noticed the ease with which the objects stock is incremented or decremented in a standard style by using the operators **+= / -=**

```
book  += 1;
book  -= 1;
cdrom += 1;
cdrom  -=  1;
```

The mechanism of giving **special meaning to an operator** is called as **operator overloading.**

Rules for overloading operators:

There are certain restrictions and limitations in overloading operators**.** They are:

- ✓ Only existing operators can be overloaded. New operators cannot be created.
- ✓ The overloaded operator must have at least one operand of user defined type.
- ✓ The basic definition of an operator cannot be replaced or in other words one cannot redefine the function of an operator. One can give additional functions to an operator
- ✓ Overloaded operators behave in the same way as the basic operators in terms of their operands.
- ✓ When binary operators are overloaded, the left hand object must be an object of the relevant class
- ✓ Binary operators overloaded through a member function take one explicit argument.

Exercises

I. Write a program that uses function overloading to do the following tasks

   a. find the maximum of two numbers (integers)

   b. find the maximum of three numbers (integers)

   SOLUTION: function protype – max (int , int) and max(int , int, int)

II. Write function definitions using function overloading to

   a. increment the value of a variable of type float

   b. increment the value of a variable of type char

   SOLUTION: function prototype – float incr (float) , char incr (char)

III. Write a program in C++ to do the following tasks using function overloading

   a. compute x^y where x and y are both integers

   b. compute x^y where x and y are both float

   SOLUTION: function prototype – int power(int,int),float power(float,float);

IV. What is the advantage of operator overloading?

V. List out the steps involved to define an overloaded operator.

VI. List out the operators that cannot be overloaded.

VII. Write a program to add two objects of the class **complex_numbers .** A complex number has two data members – real part and imaginary part. Complete the following definition and also write a main() function to perform addition of the complex_numbers objects c1 and c2 .

```
Class complex_numbers
{
   float x;
   float y;
   public :
   void assign_data(float real, float imaginary);
   void display_data();
   complex_numbers operator +(complex_numbers n1);
}
```

CHAPTER 8

CONSTRUCTORS AND DESTRUCTORS

8.1 Introduction

When an instance of a class comes into scope, a special function called the constructor gets executed. The constructor function initializes the class object. When a class object goes out of scope, a special function called the destructor gets executed. The constructor function name and the destructor have the same name as the class tag. Both the functions return nothing. They are not associated with any data type.

8.2 Constructors

```
// Program - 8.1
// to determine constructors and destructors

#include<iostream.h>
#include<conio.h>
class simple
{
 private:
        int a,b;
 public:
 simple()
 {
        a= 0 ;
        b= 0;
        cout<< “\n Constructor of class-simple “;
 }

        ~simple()
  {
        cout<<“\n Destructor of class – simple .. “;
  }

        void getdata()
 {
        cout<<“\n Enter values for a and b... “;
        cin>>a>>b;
 }

     void putdata()
    {
     cout<<“\nThe two integers .. “<<a<<‘\t’<< b;
     cout<<“\n The sum of the variables .. “<< a+b;
    }
};

 void main()
{
 simple s;
 s.getdata();
 s.putdata();
}
```

When the above program is executed, constructor simple() is automatically executed when the object is created. Destructor ~ simple() is executed, when the scope of the object 's' is lost, i.e., at the time of program termination.

The output of the program will be as follows:

```
Constructor of class - simple ..
Enter values for a & b… 5 6
The two integers….. 5   6
The sum of the variables….. 11
Destructor of class - simple …
```

8.3 Functions of constructor

1) The constructor function initializes the class object

2) The memory space is allocated to an object

8.4 Constructor overloading

Function overloading can be applied for constructors, as constructors are special functions of classes. Program - 8.2 demonstrates constructor overloading.

The constructor add() is a constructor without parameters(non parameterized). It is called as default constructor. More traditionally default constructors are referred to compiler generated constructors i.e., constructors defined by the computers in the absence of user defined constructor. A non- parameterized constructor is executed when an object without parameters is declared.

```
// Program - 8.2
// To demonstrate constructor overloading

# include<iostream.h>
#include<conio.h>
class add
{
       int num1, num2, sum;
       public:
       add()
       {
             cout<<"\n Constructor without parameters.. ";
             num1= 0;
             num2= 0;
             sum = 0;
       }

       add ( int s1, int s2 )
       {
             cout<<"\n Parameterized constructor... ";
             num1= s1;
             num2=s2;
             sum=NULL;
       }

       add (add &a)
       {
             cout<<"\n Copy Constructor ... ";
             num1= a.num1;
             num2=a.num2;
             sum = NULL;
       }

       void getdata()
       {
             cout<<"Enter data ... ";
             cin>>num1>>num2;
       }
       void addition()
       {
             sum=num1+num2;
       }

       void putdata()
       {
             cout<<"\n The numbers are..";
             cout<<num1<<'\t'<<num2;
             cout<<"\n The sum of the numbers are.. "<< sum;
       }
};

void main()
{
       add a, b (10, 20) , c(b);
       a.getdata();
       a.addition();
       b.addition();
       c.addition();
       cout<<"\n Object a :  ";
       a.putdata();
       cout<<"\n Object b :  ";
       b.putdata();
       cout<<"\n Object c.. ";
       c.putdata();
}
```

OUTPUT:

```
Constructor without parameters….

Parameterized Constructor...

Copy Constructors…

Enter data .. 5    6

Object a:

The numbers are   5    6

The sum of the numbers are ….. 11


Object b:

The numbers are   10    20

The sum of the numbers are … 30

Object c:

The numbers are   10   20

The sum of the numbers are ….. 30
```

The constructor add (int s1, int s2) is called as parameterized constructor .To invoke this constructor , the object should be declared with two integer constants or variables .

Note: char, float double parameters can be matched with int data type due to implicit type conversions

For example: add a (10, 60) / add a (ivar, ivar)

The constructor add (add &a) is called as copy constructor. A copy constructor is executed:

1) When an object is passed as a parameter to any of the member functions
   Example void add::putdata(add x);

2) When a member function returns an object
   For example, add getdata();

3) When an object is passed by reference to constructor
   For example, add a; b(a);

The following program – 2 demonstrates as to when a copy constructor is executed?

```
// Program – 8.3
// To demonstrate constructor overloading

# include<iostream.h>
#include<conio.h>
class add
{
       int num1, num2, sum;
       public:
       add()
       {
              cout<<“\n Constructor without parameters.. “;
              num1= ‘\0’;
              num2= ‘\0’;
              sum = ‘\0’;
       }

       add ( int s1, int s2 )
       {
              cout<<“\n Parameterized constructor... “;
              num1= s1;
              num2=s2;
              sum=NULL;
       }

       add (add &a)
       {
              cout<<“\n Copy Constructor ... “;
              num1= a.num1;
              num2=a.num2;
              sum = NULL;
       }

       void getdata()
       {
              cout<<“Enter data ... “;
              cin>>num1>>num2;
       }
       void addition(add b)
       {
              sum=num1+ num2 +b.num1 + b.num2;
       }

       add addition()
       {
           add a(5,6);
           sum = num1 + num2 +a.num1 +a.num2;
       }

       void putdata()
       {
              cout<<“\n The numbers are..”;
              cout<<num1<<‘\t’<<num2;
              cout<<“\n The sum of the numbers are.. “<< sum;
       }
};
```

```
void main()
{
      clrscr();
      add a, b (10, 20) , c(b);
      a.getdata();
      a.addition(b);
      b = c.addition();
      c.addition();
      cout<<“\n Object a :  “;
      a.putdata();
      cout<<“\n Object b :  “;
      b.putdata();
      cout<<“\n Object c.. “;
      c.putdata();
}
```

```
ouput of the above program
Constructor without parameters..
 Parameterized constructor...
 Copy Constructor ... Enter data ... 2  3

 Copy Constructor ...
 Parameterized constructor...
 Parameterized constructor...
 Object a :
 The numbers are..2     3
 The sum of the numbers are.. 35
 Object b :
 The numbers are..0     1494
 The sum of the numbers are.. 0
 Object c..
 The numbers are..10    20
 The sum of the numbers are.. 41
```

Have you noticed as to how many times copy constructor is executed?

Copy constructor is for the following statements of Program – 8.3.

```
add c(b);// object b is passed as argument

a.addition(b); // object b is passed as argument with the member
function addition

b = c.addition(); // the member function addition is returning an
object.
```

In the above example the function addition is also overloaded. So, primarily functions declared anywhere within the program can be overloaded.

8.5 Rules for constructor definition and usage

1) The name of the constructor must be same as that of the class

2) A constructor can have parameter list

3) The constructor function can be overloaded

4) The compiler generates a constructor, in the absence of a user defined constructor

5) The constructor is executed automatically

8.6 Destructors

A destructor is a function that removes the memory of an object which was allocated by the constructor at the time of creating a object. It carries the same name as the class tag, but with a tilde (~) as prefix.

Example :

```
class simple
        {
                  ____
                 ____
                 public :
                 ~simple()
                 {
               .......................
                 }
        }
```

8.7 Rules for destructor definition and usage

1) The destructor has the same name as that of the class prefixed by the tilde character '~'.

2) The destructor cannot have arguments

3) It has no return type

4) Destructors cannot be overloaded i.e., there can be only one destructor in a class

5) In the absence of user defined destructor, it is generated by the compiler

6) The destructor is executed automatically when the control reaches the end of class scope

Exercises

I Complete the following table

	Constructor	Destructor
1. Should be declared under	- scope	- scope
2. overloading is	-	-
3. Is executed when an object is		
The function of a		

II. Why do the following snippets throw error ?

```
Class simple
{
  private :
   int x;
  simple()
 { x = 5; }
};
```

```
Class simple
{
  private :
   int x;
 public :
  simple(int y)
 { x = y; }
};
void main()
{
  simple s;
}
```

```
Class simple
{
  private :
   int x;
 public :
  simple(int y)
 { x = y; }

 simple(int z = 5)
{
    x = z;
}
};
void main()
{
  simple s(6);
}
```

CHAPTER 9

INHERITANCE

9.1 Introduction

Inheritance is the most powerful feature of an object oriented programming language. It is a process of creating new classes called **derived classes**, from the existing or **base classes**. The derived class inherits all the properties of the base class. It is a power packed class, as it can add additional attributes and methods and thus enhance its functionality. We are familiar with the term inheritance in real life (children acquire the features of their parents in addition to their own unique features). Similarly a class inherits properties from its base (parent) class .

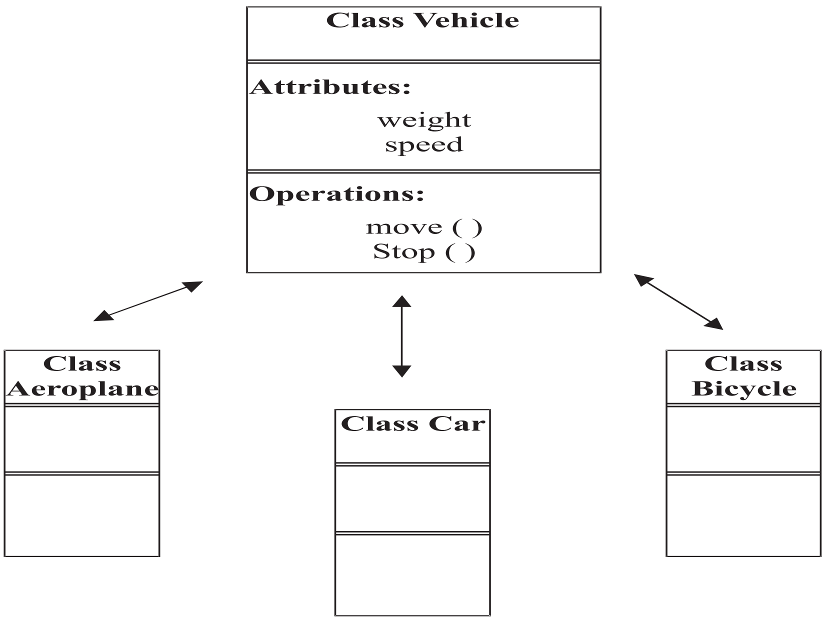


Fig.9.1 Inheritance

9.2 Advantages of inheritance

Inheritance has the following basic advantages.

1) Reusability of code : Many applications are developed in an organization. Code developed for one application can be reused in another application if such functionality is required. This saves a lot of development time.
2) Code sharing : The methods of the base class can be shared by the derived class.
3) Consistency of interface: The inherited attributes and methods provide a similar interface to the calling methods. In the Fig. 9.1 the attributes and methods of the class vehicle are common to the three derived classes – Aeroplane, Car and Bicycle. These three derived classes are said to be having a consistence interface.

9.3 Derived Class and Base class

A base class is a class from which other classes are derived. A derived class can inherit members of a base class.

While defining a derived class, the following points should be observed

a. The keyword **class** has to be used
b. The name of the derived class is to be given after the keyword class
c. A single colon
d. The type of derivation, namely **private, public or protected**
e. The name of the base class or parent class
f. The remainder of the derived class definition

```
// Program - 9.1

#include< iostream.h>
#include<conio.h>
class add
{
      int sum;
      protected:
            int num1, num2;
      public:
      add()
      {
         num1= num2= sum=0';
         cout<<"\n Add constructor .. ";
      }

      accept()
      {
         cout<<"\n Enter two numbers .. ";
         cin>>num1>>num2;
      }

      plus()
      {
            sum = num1 + num2;
            cout<<"\n The sum of two numbers is .. "<< sum;
      }
};

class subtract :public add
{
      int sub;
      public:
       subtract()
       {
         sub = 0;
         cout<<"\n Subtract constructor .. ";
       }
       minus()
       {
         add::accept();
         sub= num1-num2;
         cout<<"\n The difference of two numbers are ... "
              << sub;
       }
};
```

```
void main()
{
     subtract s;
     int choice = 0;
     cout<<"\n Enter your choice ";
     cout<<" \n1. Add..\n2. Subtract ..";
     cin>>choice;
     switch( choice )
     {
           case 1:
                 s.accept();
                 s.plus();
                 break;
           case 2:
                 s.minus();
                 break;
     }
}
```

In the Program – 9.1 the base class is add and the derived class is subtract. The derived class should be indicated as

class der_name : visibility mode base class-id

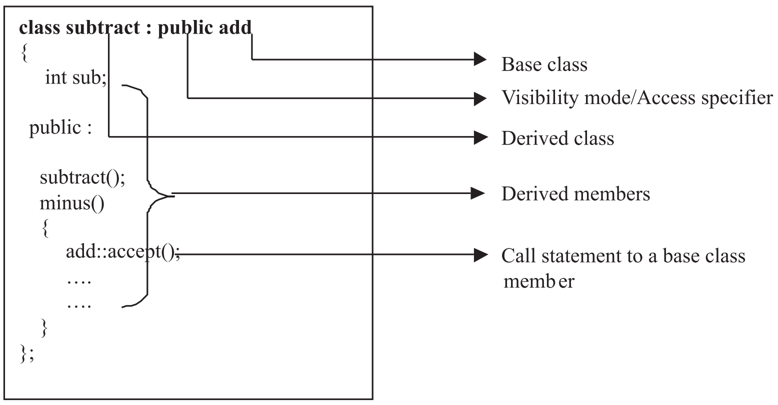
{

data members of the derived_class

functions members of derived_class

}

In the Program - 9.1 the derived class, subtract is defined as



```
class subtract : public add
{
   int sub;

 public :

  subtract();
  minus()
  {
     add::accept();
     ….
     ….
  }
};
```

Base class
Visibility mode/Access specifier
Derived class
Derived members
Call statement to a base class member

9.4 Visibility Mode/Accessibility specifier

An important feature in Inheritance is to know as to when a member of a base class can be used by the objects or the members of the derived class. This is called as **accessibility**. The three access specifiers are private, protected and public. Access specifier is also referred to as visibility mode.The default visibility mode is private. The following Table 9.1 explains the scope and accessibility of the base members in the derived.

Base Class members	Derived Class		
	Private	Protected	Public
Private members	Are not inherited but they continue to exist		
Protected members	Inherits protected members as private members	Inherits protected and public as protected of derived	Protected members are retained as protected of the derived
Public members	Are inherited as private members of the derived	Inherits as protected members of the derived	Inherits public members as public of derived

Table 9.1 Scope and Access of Base Members in the Derived Class

When a base class is inherited with private visibility mode the public and protected members of the base class become 'private' members of the derived class

When a base class is inherited with protected visibility mode the protected and public members of the base class become ' protected members ' of the derived class

When a base class is inherited with public visibility mode , the protected members of the base class will be inherited as protected members of the derived class and the public members of the base class will be inherited as public members of the derived class.

When classes are inherited publicly, protectedly or privately the private members of the base class are not inherited they are only visible i.e continue to exist in derived classes, and cannot be accessed

The declaration of classes add and subtract of Program-9.1 is as follows

```
class add
{
      private:
              int sum;
     protected :
              int num1, num2;
    public:
              add();
              accept();
              plus();
};
class subtract : private add
{
      int sub;
   public:
      subtract();
      minus();
};
```

The data members and member functions inherited by subtract are:

int num1 & num2	with status as private of subtract
accept(); plus();	with status as private

I

```
class subtract : private add
{
        int sub;
  public:
       subtract( );
       minus();
};
```

II

```
class subtract : protected add
{
        int sub;
  public:
       subtract();
       minus();
};
```

III

```
class subtract : private add
{
        int sub;
  public:
       subtract();
       minus();
};
```

Accessibility of base members is shown in Table 9.2.

The constructors of the base class are not inherited, but are executed first when an instance of the derived class is created.

class subtract	Ⅱ	III
private:	protected	public

sub
num1
num2
accept()
plus()
public:
subtract()
minus();
private mode
of inheritance
	sub
	protected:
	num1. num2
	accept();
	plus();
	public:
	subtract()
	minus()
	protected mode
	of inheritance
		sub
		protected:
		num1. num2
		public:
		accept();
		plus();
		subtract()
		minus();
		public mode
		of inheritance

Table 9.2 accessibility of Base Members

Consider the objects declared in the Program – 9.1. Complete the following table based on this program.

Constructors executed are _____________ , __________________

The objects of classes	DATA MEMBERS	METHODS/ FUNCTIONS
add		
subtract		

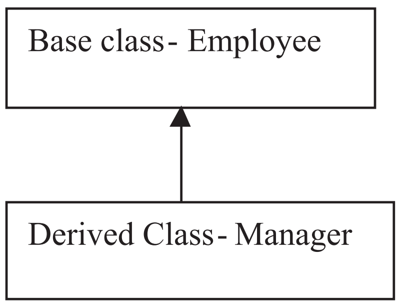
Table 9.3 Complete this Table

9.5 Types of inheritance

Classes can be derived from classes that are themselves derived. There are different types of inheritance viz., Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance.

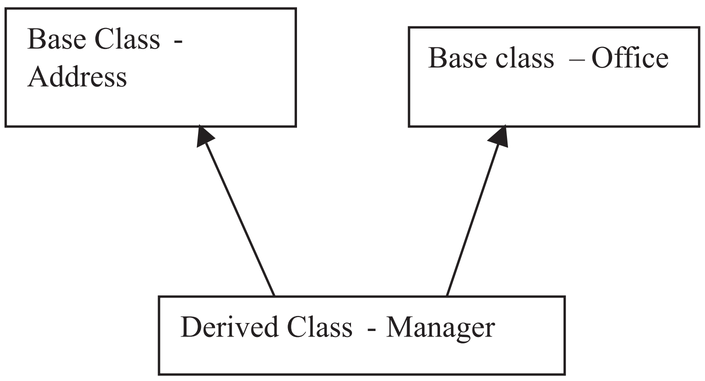
1) Single Inheritance

When a derived class inherits only from one base class, it is known as single inheritance



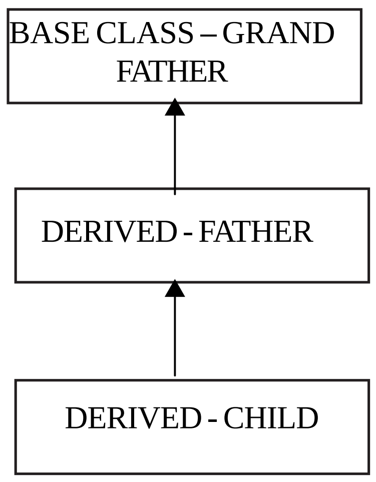
2) Multiple Inheritance

When a derived class inherits from multiple base classes it is known as multiple inheritance



3) Multilevel Inheritance

The transitive nature of inheritance is reflected by this form of inheritance. When a class is derived from a class which is a derived class itself – then this is referred to as multilevel inheritance.



What will be the output of Program 9.2?

```
// Program - 9.2
#include<iostream.h>
#include<conio.h>
class base
{
      public:
      base()
      {
            cout<<“\nConstructor of base class...
“;
      }
      ~base()
      {
      cout<<“\nDestructor of base class.... “;
      }
};
class derived:public base
{
  public :
  derived()
  {
    cout << “\nConstructor of derived ...”;
  }
  ~derived()
  {
      cout << “\nDestructor of derived ...”;
  }
};

class derived2:public base
{
  public :
  derived()
  {
    cout << “\nConstructor of derived2 ...”;
  }
  ~derived()
  {
      cout << “\nDestructor of derived2 ...”;
  }
};

void main()
{
  derived2 x;
}
```

OUTPUT

```
Constructor  of base class…
Constructor of derived ….
Constructor of derived2 …
Destructor of derived2…
Destructor of derived
Destructor of base class ..
```

✓ The constructors are executed in the order of inherited class i.e., from base constructor to derived. The destructors are executed in the reverse order

✓ Classes used only for deriving other classes are called as Abstract Classes ie., to say that objects for these classes are not declared.

Exercises

1) Given the following set of definitions

```
 class node
 {
             int x;
             void init();
       public:
             void read();
       protected:
             void get();
 };
 class type : public node
 {
             int a;
public:
```

```
                void func1();
        protected:
                int b;
                void getb();
}
class statement :private type
{
                int p;
        public:
                void func2();
        protected:
                void func3();
};
```

Complete the following table

Members of the class type	Accessibility of members /their classes		
	private	protected	public
Members inherited by class type			
Defined in class type			

Table- 4 class node ….

Members of the class statement	Accessibility of members/ their classes		
	private	protected	public
Members inherited by class statement			
Defined in class statement			

Table- 5 class statement

Objects	Can access members	
	Data members	Member functions
class type		
class statement		

Table- 6 Objects…

2) Find errors in the following program. State reasons

```
  #include<iostream.h>
  class A
  {
        private :
              int a1;
        public:
              int a2;
        protected:
              int a3;
  };
  class B : public A
{
           public:
           void func()
           {
                 int b1, b2 , b3;
                 b1 = a1;
                 b2 = a2;
                 b3 = a3;
           }
};
void main()
{
           B der;
           der.a3 = 0';
           der.func();
 }
```

3) Consider the following declarations and answer the questions given below

```
class vehicle
{
        int wheels;
        public:
                void inputdata( int, int);
                void outputdata();
        protected :
                int passenger;
};
class heavy_vehicle : protected vehicle
{
        int diesel_petrol;
        protected:
                int load;
        public:
                void readdata( int, int);
                void writedata();
};
class bus: private _heavy_vehicle
{
        char marks[20];
        public:
                void fetchdata( char );
                void displaydata();
};
```

a. Name the base class and derived class of the class heavy_vehicle

b. Name the data members that can be accessed from the function displaydata()

c. Name the data members that can be accessed by an object of bus class

d. Is the member function output data accessible to the objects of heavy_vehicle class

4) What will be the output of the following program

```
#include<iostream.h>
#include<conio.h>
class student
{
            int m1, m2, total;
    public:
            student ( int a, int b)
            {
                        m1 = a;
                        m2 = b;
                  cout<<"\n Non parameterized constructors..";
            };
```

CHAPTER 10

IMPACT OF COMPUTERS ON SOCIETY

10.1 Introduction

“India lives in her seven hundred thousand villages”
- Mohandas Karamchand Gandhi

To reach out the benefits of IT to the common man we need at least three technical elements :

- Connectivity [Computer networks and Internet facility]
- Affordable computers or other similar devices
- Software

It is interesting to observe that 85% of computer usage is “Word Processing”. Computers can do many more things for the common man than this. Quality IT education will enable the common man to put computers to a better use. This chapter presents the possible ways in which computers can be used to develop the society.

10.2 Computers for Personal Use

Personal computers have totally changed the way we work, live and think. Word Processing, Databases, Spreadsheets and Multimedia presentation packages have increased the efficiency at work. There are many packages that are being used. Desktop Publishing and other impressive packages for graphics are adding value to the work done. Paint, games and a large set of similar packages are providing facilities for people of all age groups to use the computer. Browsing, e-mail and chat have changed our life style.

Today computers come in different sizes and shapes. Some adaptation of the basic computer model is making it more useful in the homes of the user.

10.3 Computerized Homes

Home	Products and a brief Description
Living Room	• LCD Screen, Camera and Speakers: Mounted on the Wall to provide better effect and save floor space. • Archive Unit: Enables data storage and management· • Emotion Containers: They are small compartments with a screen, speaker and a scent to derive emotional comfort. This can prevent people from acquiring bad habits • Personal Archives: They store personal details like family photographs and personal treasures. In addition it enables connectivity to other people. • Picture Phone and Pad: Picture based personal telephone directory.

Kids Room	• Devices that provide listening access to audio sources in home such as radio, television • Devices with kara-oke features allowing one to sing along with the audio coming from the original source • Robots that function as Electronic Pets· • Devices that enable game plying over the net. In addition real world characters are translated into the computer world and a kid can play with them
Home Office	• Packages to make animated stories· • Memo Frame: Easy interaction with other people through touch screen, scanner and microphone facilities. Bookshelf: To manage and study electronic books· Personal Creativity Tool: To draw, capture and work with multimedia elements·Advanced data accessing systems
Bed Room	• Touch and Voice Control for various appliances· • Display Monitors, Special Headphones and Moving Telephone· • Projection TV: Projects the TV pictures on the ceiling or walls • Alarm Clock
Bath Room	Mirrors, Medical Box and Special Speakers
Kitchen	Speakers, Rack Telephone, Intelligent Aprons, Kettle, Toaster, Food Analyzer, Health Monitor, Devices to preserve food
Dining Room	• Interactive Tablecloth to keep the food sufficiently warm • Ceramic Audio player and speakers· • Communication facilities around the dinner table • Interactive screens to consult with the cook and other kitchen staff

Table 10.1 Computerized Products for Home

10.4 Home Banking and Shopping

Traditional banking needed the user to go to the bank to perform related activities. These activities include depositing or withdrawing money from the account or securing loans. Banks gradually began providing many other services including term deposits, agricultural loans, paying bills related to other services such as telephones, electricity and locker facilities. As the confidence of the common man in banking improved, banks became a key component in the national economy.

This also means long queues at the banks during working hours. Introduction of IT in banks reduced the time required to provide service to a user. Long queues were being handled quickly.

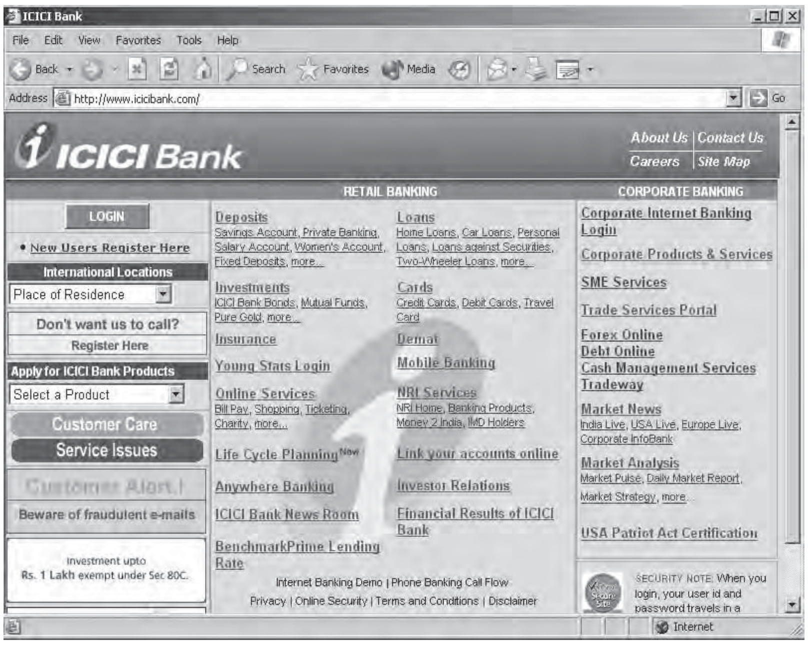
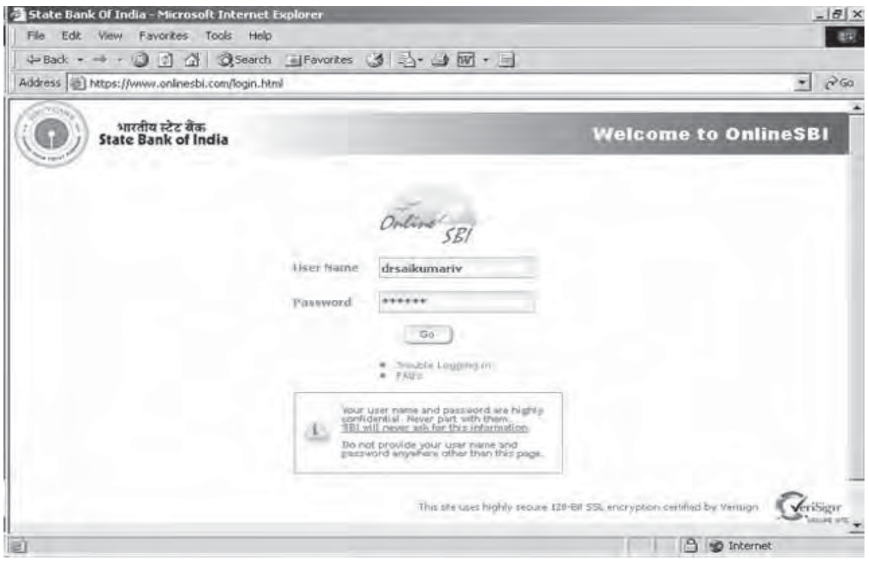

ICICI Bank
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.icicibank.com/
Go
ICICI Bank
About Us | Contact Us
Careers | Site Map
RETAIL BANKING
CORPORATE BANKING
LOGIN
• New Users Register Here
International Locations
Place of Residence
Don't want us to call?
Register Here
Apply for ICICI Bank Products
Select a Product
Customer Care
Service Issues
Customer Alert !
Beware of fraudulent e-mails
Investment upto Rs. 1 Lakh exempt under Sec 80C.
Deposits
Savings Account, Private Banking, Salary Account, Women's Account, Fixed Deposits, more...
Loans
Home Loans, Car Loans, Personal Loans, Loans against Securities, Two-Wheeler Loans, more...
Investments
ICICI Bank Bonds, Mutual Funds, Pure Gold, more...
Cards
Credit Cards, Debit Cards, Travel Card
Insurance
Demat
Young Stars Login
Mobile Banking
Online Services
Bill Pay, Shopping, Ticketing, Charity, more...
NRI Services
NRI Home, Banking Products, Money 2 India, IMD Holders
Life Cycle Planning New
Link your accounts online
Anywhere Banking
Investor Relations
ICICI Bank News Room
Financial Results of ICICI Bank
BenchmarkPrime Lending Rate
Internet Banking Demo | Phone Banking Call Flow
Privacy | Online Security | Terms and Conditions | Disclaimer
Corporate Internet Banking Login
Corporate Products & Services
SME Services
Trade Services Portal
Forex Online
Debt Online
Cash Management Services
Tradeway
Market News
India Live, USA Live, Europe Live, Corporate InfoBank
Market Analysis
Market Pulse, Daily Market Report, Market Strategy, more...
USA Patriot Act Certification
SECURITY NOTE: When you login, your user id and password travels in a
Internet

Fig.10.1 ICICI Bank


State Bank Of India - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://www.onlinesbi.com/login.html
Go
भारतीय स्टेट बैंक
State Bank of India
Welcome to OnlineSBI
Online SBI
Go
Trouble Logging in
FAQs
Internet

User Name drsaikumariv

Password ******

Your user name and password are highly confidential. Never part with them. SBI will never ask for this information.

Do not provide your user name and password anywhere other than this page.

This site uses highly secure 128-Bit SSL encryption certified by Verisign

Fig.10.2 State Bank of India

Advanced machines like ATM enable withdrawal of money from the accounts in a particular bank anytime and anywhere. This helps the user in emergency situations where money is needed during the nights and holidays. However, the user has to go to the nearest ATM facility. It is possible that every branch of any well recognized bank will have a ATM facility soon.

e-Banking permits banking from the comfort of the home by using internet facilities. It has truly improved the reach and services of banks.

Computers are used in many areas even for Shopping. You can purchase any product, any brand, any quantity from anywhere through e-shopping. You need not go to the shop. The pictures and other details of what can be purchased are available on the website of the shop. You have to select and order. Advance payment for these items is assured by various methods. Credit cards and prior registration with the shop are the popular methods. The items purchased will be delivered at your home.

10.5 Computers in Education

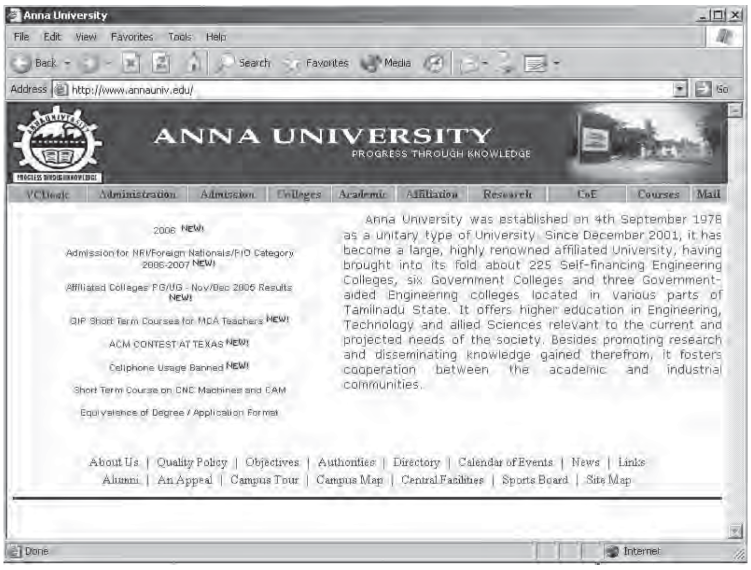
Computers are used in many areas of Education including:

- Buying and browsing the latest edition of books by both local & foreign authors Educational CDROMs

- Computer Based Tutorials (CBT).

- Spreading information about schools, colleges, universities and courses offered, admission procedures, accommodation facilities, scholarships, educational loans, career guidance.

- e-Learning that enables online educational programs leading to degrees and certifications


Anna University
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.annauniv.edu/ Go
ANNA UNIVERSITY
PROGRESS THROUGH KNOWLEDGE
VCDesk Administration Admission Colleges Academic Affiliation Research CoE Courses Mail
About Us | Quality Policy | Objectives | Authorities | Directory | Calendar of Events | News | Links
Alumni | An Appeal | Campus Tour | Campus Map | Central Facilities | Sports Board | Site Map
Done
Internet

2006 NEW!

Admission for NRI/Foreign Nationals/PIO Category 2006-2007 NEW!

Affiliated Colleges PG/UG - Nov/Dec 2005 Results NEW!

QIP Short Term Courses for MCA Teachers NEW!

ACM CONTEST AT TEXAS NEW!

Cellphone Usage Banned NEW!

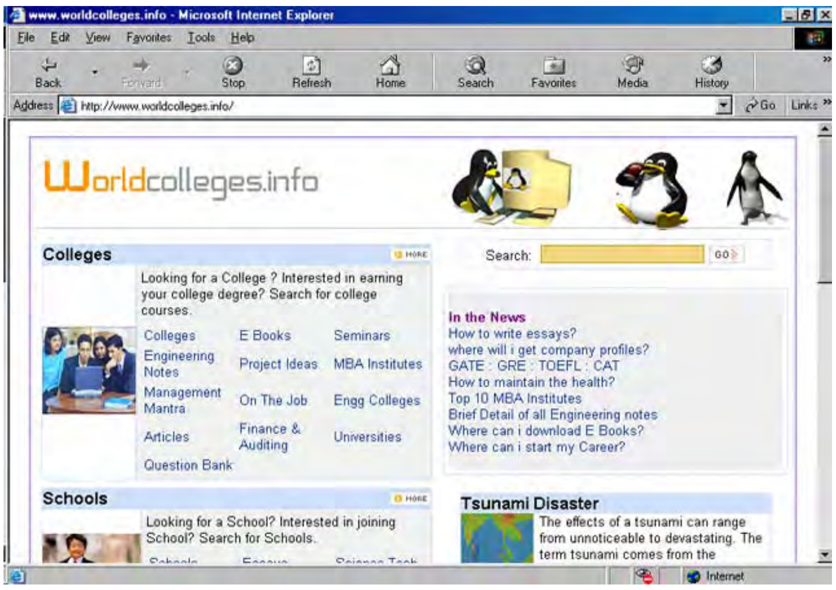
Short Term Course on CNC Machines and CAM

Equivalence of Degree / Application Format

Anna University was established on 4th September 1978 as a unitary type of University. Since December 2001, it has become a large, highly renowned affiliated University, having brought into its fold about 225 Self-financing Engineering Colleges, six Government Colleges and three Government-aided Engineering colleges located in various parts of Tamilnadu State. It offers higher education in Engineering, Technology and allied Sciences relevant to the current and projected needs of the society. Besides promoting research and disseminating knowledge gained therefrom, it fosters cooperation between the academic and industrial communities.

Fig. 10.3 Anna University

Fig.10.4 Computer Based Tutorials


www.worldcolleges.info - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History
Address http://www.worldcolleges.info/
Go Links
Worldcolleges.info
Search: GO
MORE
Internet

Colleges

Looking for a College ? Interested in earning your college degree? Search for college courses.

Colleges	E Books	Seminars
Engineering Notes	Project Ideas	MBA Institutes
Management Mantra	On The Job	Engg Colleges
Articles	Finance & Auditing	Universities
Question Bank		

In the News
How to write essays?
where will i get company profiles?
GATE : GRE : TOEFL : CAT
How to maintain the health?
Top 10 MBA Institutes
Brief Detail of all Engineering notes
Where can i download E Books?
Where can i start my Career?

Schools

Looking for a School? Interested in joining School? Search for Schools.

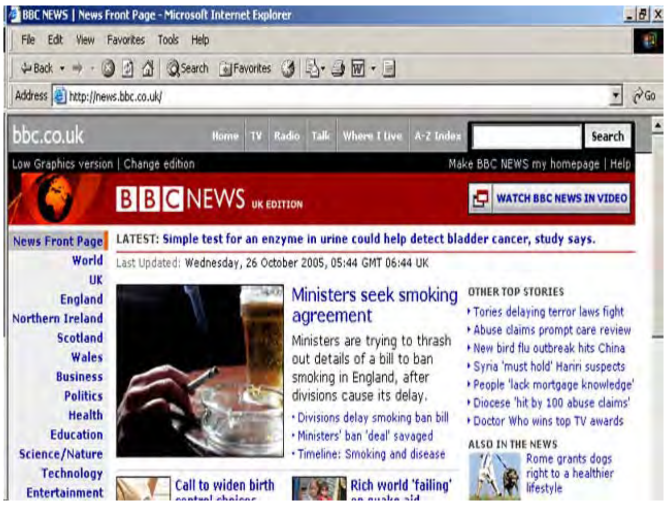
Tsunami Disaster

The effects of a tsunami can range from unnoticeable to devastating. The term tsunami comes from the

Fig.10.5 Information on Education around the Globe

10.6 Computers in Entertainment

Computers contribute to entertainment also. You can update your knowledge in fine arts like painting, music, dance, yoga, games, science, nature, latest news and events. You can know more about various places of worship and places of interest for tourists.



BBC NEWS | News Front Page - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://news.bbc.co.uk/ Go

bbc.co.uk Home TV Radio Talk Where I Live A-Z Index Search

Low Graphics version | Change edition Make BBC NEWS my homepage | Help

BBC NEWS UK EDITION WATCH BBC NEWS IN VIDEO

News Front Page
World
UK
England
Northern Ireland
Scotland
Wales
Business
Politics
Health
Education
Science/Nature
Technology
Entertainment

LATEST: Simple test for an enzyme in urine could help detect bladder cancer, study says.

Last Updated: Wednesday, 26 October 2005, 05:44 GMT 06:44 UK

Ministers seek smoking agreement

Ministers are trying to thrash out details of a bill to ban smoking in England, after divisions cause its delay.

- Divisions delay smoking ban bill
- Ministers' ban 'deal' savaged
- Timeline: Smoking and disease

OTHER TOP STORIES
- Tories delaying terror laws fight
- Abuse claims prompt care review
- New bird flu outbreak hits China
- Syria 'must hold' Hariri suspects
- People 'lack mortgage knowledge'
- Diocese 'hit by 100 abuse claims'
- Doctor Who wins top TV awards

ALSO IN THE NEWS
Rome grants dogs right to a healthier lifestyle

Call to widen birth

Rich world 'failing'

Fig. 10.6 Computers in Entertainment

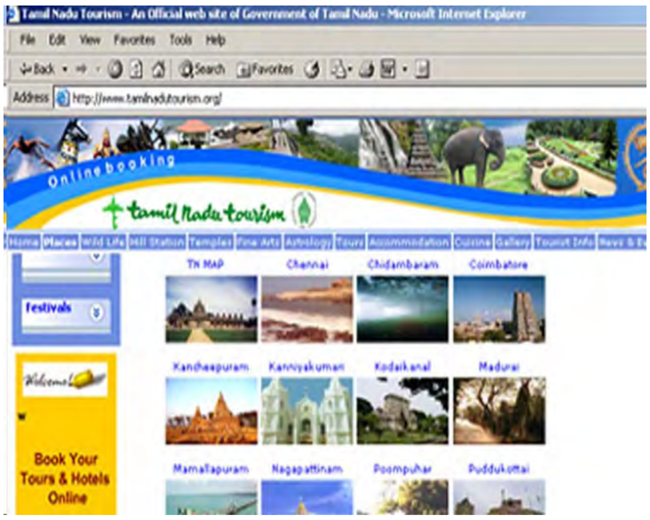
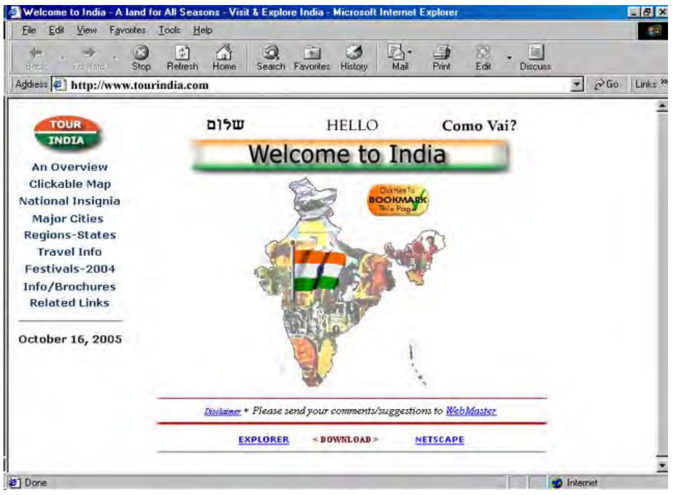

Tamil Nadu Tourism - An Official web site of Government of Tamil Nadu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.tamilnadutourism.org/
Online booking
tamil nadu tourism
Home Places Wild Life Hill Station Temples Fine Arts Astrology Tours Accommodation Cuisine Gallery Tourist Info
Festivals
Welcome
Book Your Tours & Hotels Online
TN MAP
Chennai
Chidambaram
Coimbatore
Kancheepuram
Kanniyakumari
Kodaikanal
Madurai
Mamallapuram
Nagapattinam
Poompuhar
Puddukottai

Fig.10.7 Computers in Tourism (Tamil Nadu)


Welcome to India - A land for All Seasons - Visit & Explore India - Microsoft Internet Explorer

http://www.tourindia.com

שלום HELLO Como Vai?

Welcome to India

An Overview
Clickable Map
National Insignia
Major Cities
Regions-States
Travel Info
Festivals-2004
Info/Brochures
Related Links

October 16, 2005

Disclaimer * Please send your comments/suggestions to WebMaster

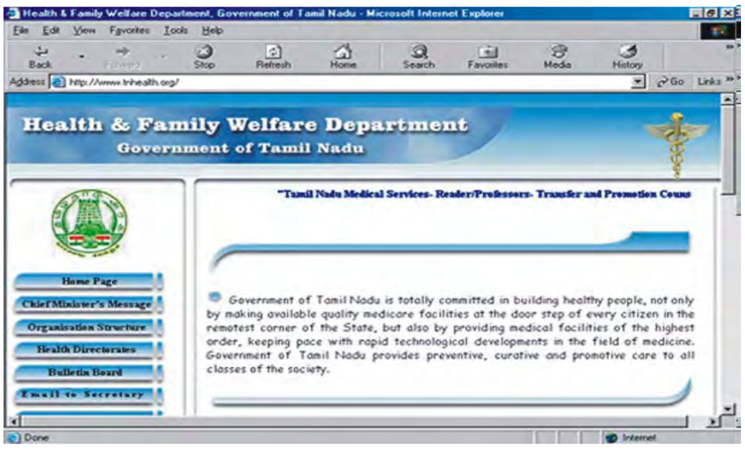
EXPLORER < DOWNLOAD > NETSCAPE

Fig.10.8 Computers in Tourism (India)

10.7 Computers in Healthcare

Healthcare is dominated by large amounts of data and limited financial and human resources and need for accountability of those resources

Healthcare has improved significantly ever since computers found their way into the hands of doctors and health administrators.


Health & Family Welfare Department, Government of Tamil Nadu - Microsoft Internet Explorer

http://www.tnhealth.org/

Health & Family Welfare Department
Government of Tamil Nadu

"Tamil Nadu Medical Services- Reader/Professors- Transfer and Promotion Couns

Government of Tamil Nadu is totally committed in building healthy people, not only by making available quality medicare facilities at the door step of every citizen in the remotest corner of the State, but also by providing medical facilities of the highest order, keeping pace with rapid technological developments in the field of medicine. Government of Tamil Nadu provides preventive, curative and promotive care to all classes of the society.

Fig.10.9 Computers in Health and Family Welfare

Computers are used in many areas of healthcare including

- Hospital Management System
- Patient Tracking System
- Exchange of diagnostic records between healthcare units
- Tracking and Monitoring Communicable Diseases
- Decision support systems with highly advanced computing techniques

Today many doctors are innovating to suit their needs. It is indeed a good sign for the patients. Tele-medicine is built largely on the foundational systems mentioned above. Internet facilitates remote diagnostics. This ensures expert advice at places where it is not there.

10.8 Computers in Agriculture

Farming and agriculture might seem like low technology enterprises, but these industries have benefited from computerization more than the casual observer might think. Farmers, both professional and hobbyists benefit from online resources such as seed estimators and pest information sites. Professional farmers can use revenue estimators to help them plan which crops will produce the highest profits based on weather patterns, soil types, and current market values.

M S S R F - Centre for Research on Sustainable Agricultural & Rural Development - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites Media History
Address http://www.mssrf.org/
Go Links
MS Swaminathan Research Foundation
Centre for Research on Sustainable Agricultural & Rural Development
Home
About Us
Awards & Achievements
Programmes
Special Programmes
Bio Informatics
Events & Conferences
Notice Board
Friends of MSSRF
Publications
Archives
Other Links
Contact Us
Mission 2007 - Hunger Free India
Mission 2007 - Every Village a Knowledge Centre
Information Village Research Project
Remote Sensing & GIS Lab
Budget 2005 "A Knowledge Centre in Every Village"
Rural Infrastructure Development Fund (RIDF)
Support to Village Knowledge Centres
The Third MSSRF South - South Exchange Travelling Workshop Report

MS Swaminathan Research Foundation - MSSRF

M. S. Swaminathan Research Foundation (MSSRF) was registered in 1988 as a non-profit Trust. The basic mandate of MSSRF is to impart a pro-nature, pro-poor and pro-women orientation to a job-led economic growth strategy in rural areas through harnessing science and technology for environmentally sustainable and socially equitable development.

MSSRF is doing research in the following five areas: Coastal Systems Research, Biodiversity and Biotechnology, Ecotechnology and Sustainable Agriculture, Reaching the Unreached, and Education, Communication, Training and Capacity Building.

Fig.10.10 Computers in Agriculture

Some of the areas where software has been developed are:

- Agricultural Finances and Accounting
- Alternative farming techniques
- Animal Husbandry
- Buildings and Irrigation
- Chat with other agriculturists and scientists
- Farmland Assessment
- Fertilizer Analysis
- Finding links to farm resources, chat boards, classified advertisements, and other farm-related information
- Gardening
- Improving Cow Herds and Increasing revenues
- Land Management
- Livestock
- Milk production
- Use of satellite imagery to decide on the crops

10.9 Internet in real time Applications

All applications mentioned above happen in real time and over the net. You can reserve or book air and train tickets from your own place and at your own pace through computers.

Welcome to
Web Based Rail Reservations
www.irctc.co.in
ENTER
Indian Railway Catering and Tourism Corporation Ltd.,

Fig.10.11 Computers in Realtime Applications

Exercises

This Chapter has the support of multimedia content to understand more about the applications presented. You must see this content and where possible visit the websites indicated.

This multimedia content is provided to your school on a separate CD. Please contact your teacher to get this CD.

CHAPTER 11

IT ENABLED SERVICES

11.1 Introduction

Information Technology that helps in improving the quality of service to the users is called IT Enabled Services [ITES]. IT Enabled Services are human intensive services that are delivered over telecommunication networks or the Internet to a range of business segments. ITES greatly increases the employment opportunities.

Is typing a letter using the computer an ITES? The answer is No. However, a facility that allows the user to speak into a special device called 'Dictaphone' and then convert the speech into a letter is an ITES.

Word processors, Spreadsheets and Databases have ensured that many traditional services are IT Enabled. However, the user is expected to learn several aspects of these IT tools before gaining from their use. ITES adds value to these services by reducing the learning that needs to be done by the users. ITES thus has the potential to take the power of IT to users who do not know IT.

ITES can improve the quality of the service either directly or indirectly. Improved customer satisfaction, better look and feel and an improved database are some direct benefits. Indirect benefits are seen after sometime. Data collected for one purpose may be useful for some other purpose also after some time.

Some of the IT enabled services presented in this chapter are:

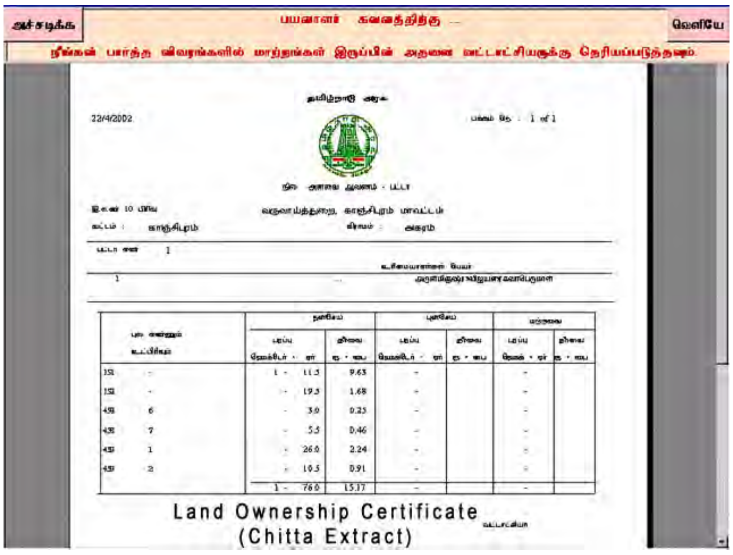
- e-Governance
- Call Centers
- Data Management
- Medical [Telemedicine and Transcription].
- Data Digitization
- Website Services

ITES such as Business Process Outsourcing (BPO), Digital Content Development / Animation, Human Resources Services and Remote Maintenance are other important areas.

ITES requires practical IT skills especially in the area of Databases, Internet and good communication skills in English. A formal training in Soft Skills to understand the basic aspects of Industry Culture, professionalism and etiquette is needed for the effective implementation of ITES.

11.2 e-Governance

Computers help you to look at the government websites and the services provided by them. The various websites provided by the government give the details about the departments, specific functions, special schemes, documents, contacts, links, IAS intranet, site map, search, what's new, press releases, feedback. These websites are both in English and Tamil.



பயனாளர் கவனத்திற்கு --

நீங்கள் பார்த்த விவரங்களில் மாற்றங்கள் இருப்பின் அதனை வட்டாட்சியருக்கு தெரியப்படுத்தவும்.

தமிழ்நாடு அரசு

22/4/2002 பக்கம் எண் : 1 of 1

நில அளவை ஆவணம் - பட்டா

[illegible] 10 [illegible] வருவாய்த்துறை, காஞ்சிபுரம் மாவட்டம்

வட்டம் : காஞ்சிபுரம் கிராமம் : அகரம்

பட்டா எண் 1

உரிமையாளர்கள் பெயர்

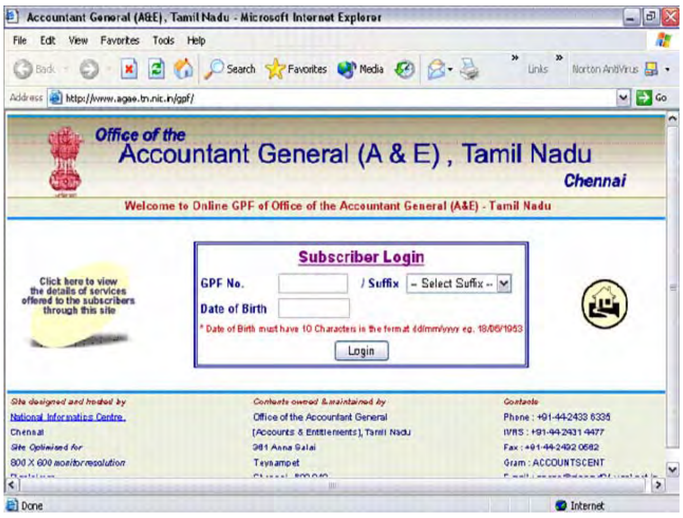
1 ... [illegible]

புல எண்ணும் உட்பிரிவும்		நன்செய்		புன்செய்		மற்றவை	
		பரப்பு ஹெக்டேர் - ஏர்	தீர்வை ரூ - பை	பரப்பு ஹெக்டேர் - ஏர்	தீர்வை ரூ - பை	பரப்பு ஹெக்டேர் - ஏர்	தீர்வை ரூ - பை
151	-	1 - 11.5	9.63	-		-	
151	-	- 19.5	1.68	-		-	
458	6	- 3.0	0.25	-		-	
458	7	- 5.5	0.46	-		-	
458	1	- 26.0	2.24	-		-	
458	2	- 10.5	0.91	-		-	
		1 - 76.0	15.17	-		-	

Land Ownership Certificate
(Chitta Extract)

வட்டாட்சியர்

Fig.11.1 Getting Land Certificate using Internet


Accountant General (A&E), Tamil Nadu - Microsoft Internet Explorer

http://www.agae.tn.nic.in/gpf/

Office of the
Accountant General (A & E) , Tamil Nadu
Chennai

Welcome to Online GPF of Office of the Accountant General (A&E) - Tamil Nadu

Click here to view the details of services offered to the subscribers through this site

Subscriber Login

GPF No. / Suffix -- Select Suffix --

Date of Birth

* Date of Birth must have 10 Characters in the format dd/mm/yyyy eg. 18/06/1953

Login

Site designed and hosted by
National Informatics Centre,
Chennai
Site Optimised for
800 X 600 monitor resolution

Contents owned & maintained by
Office of the Accountant General
(Accounts & Entitlements), Tamil Nadu
361 Anna Salai
Teynampet

Contacts
Phone : +91-44-2433 6335
IVRS : +91-44-2431 4477
Fax : +91-44-2492 0682
Gram : ACCOUNTSCENT

Fig.11.2 Office of the Accountant General (A& E), Tamil Nadu

11.3 Call Centers

Information Technology is happening all over the globe. Users or Customers of IT products are all over the world. The customers are in need of a facility that ensures communication services on all days of the year all round the clock – 24 X 365.

A call center is sometimes defined as a telephone based shared service center for specific customer activities and are used for number of customer-related functions like marketing, selling, information transfer, advice, technical support and so on. A call center has adequate telecom facilities, trained consultants, access to wide database, Internet and other on-line information support to provide information and support services to customers. It operates to provide round the clock and year round service i.e.24 x 365 service.

11.4 Data Management

Data Management is a category of IT Enabled Services pertaining to collection, digitization and processing of data coming from various sources. Traditional data processing services comprise punching data from manually filled forms, images or publications; preparing databases and putting them together. However, with the advent of multimedia and internet, sources have increased to include manually printed documents, images, sounds and video. Equally diverse are the new output media which include databases on servers, hard copy publications, CD-ROM records emanating from internet based queries.

Data management is the key for effective and profitable use of IT in organizations. The range of ITES in this category are:

- ASCII format for upload to your database
- Character Recognition and Processing
- Custom reports
- Data Entry
- Data entry front end edits
- Document Preparation
- Forms are imaged and transferred to CD ROM
- Handwritten, Machine Print, Mark Sense, Bar Coding (Reader Response can be captured and processed from any hard copy or faxed document).
- Image Capturing
- Image Keying
- Image Storage & Retrieval
- Outcome studies
- Statistical analysis

Some of the organizations that can potentially benefit from ITES in this category are:

- Back office Operations such as Accounts, Financial services
- Banking
- Government agencies
- Hospital
- Insurance
- Legal
- Manufacturing
- Municipalities
- Police departments

- Public utilities
- Publishing
- Telecom
- Transportation

Each of the organizations mentioned above presents a huge opportunity in ITES in the critical area of Data Management. Banking, Financial Services and Insurance sectors are popularly termed BFSI. BFSI and Pension Services are high growth areas for ITES.

Data Security and Customer Privacy are two important aspects that must be ensured by the ITES provider in this area. An ITES provider may be serving multiple organizations. The service provider must ensure the privacy aspects of every organization. Computer Ethics is critical for the success of ITES.

11.5 Medical Transcription and Tele-Medicine

Medical Transcription is a permanent, legal document that formally states the result of a medical investigation. It facilitates communication and supports the insurance claims. There are three main steps involved in Medical Transcription. These include:

Step 1: Hospitals that want to use this form of ITES sign up with a service provider. Doctors are trained in the process. The doctor dictates into a special device or a free telephone. The sound is then stored on a server at the other end.

Step 2: The sound is digitized and sent to the ITES provider. This service provider is usually in a different country. Providing transcription services in countries like USA is becoming very expensive both to the patient and the hospital. So, ITES in this category reduces the cost by having it done in a country where the cost is affordable. The digitized data is converted back to sound. The trained transcriptionists listen to the dictation and transcribe. This is a formal record of the diagnosis made by the doctor.

Step 3: The transcribed files are sent out to quality control persons, who listen to the dictation and check the transcription. Corrections are made if required. Then the transcribed reports are transmitted back to hospital as a word document. This is valid for legal purposes and making insurance claims.

11.6 Data Digitization

Digitization refers to the conversion of non-digital material to digital form. A wide variety of materials as diverse as maps, manuscripts, moving images and sound may be digitized.

Digitization offers great advantages for access, allowing users to find, retrieve, study and modify the material. However, reliance on digitization as a preservation strategy could place much material at risk. Digital technologies are changing rapidly. Preservation is a long term strategy and many technologies will become obsolete soon. This instability in technology can lead to the loss of the digitized objects. This defeats the purpose of preservation. Some application areas of the digital technology are as follows:

- Annual reports and price list
- Books
- Database archiving
- Electronic Catalogues & Brochures
- Engineering and Design
- Geographical Information System.
- Movies, Sounds and High quality image preservation
- Product/Service Training Manuals
- Research Journals and Conference Papers

The steps in data digitization are:

- Understanding the customer needs
- Customer needs are used as the basis for defining the objectives of digitization

- A pilot application is built
- After the pilot application is approved, the full digitization of data identified by the customer is undertaken.
- Different types of data are digitized using different techniques. Many advance software packages are available to improve the quality of the digitized form of the original document.
- The digitized data is indexed and a table of contents is produced to improve accessibility. Highly advanced and reliable storage facilities are used to stock the digitized data.

There are many benefits of digitization. Some of the key benefits are:

- Long term preservation of the documents.
- Storage of important documents at one place.
- Easy to use and access to the information.
- Quick and focused search of relevant information in terms of images and text.
- Easy transfer of information in terms of images and text.
- Easy transfer of information through CD-ROM, internet and other electronic media

11.7 Website Services

Computers also help us in accessing website services such as:

- Agriculture Marketing Network
- Career guidance
- Employment Online
- General Provident Fund
- Results of various Examinations

In the very near future there will be many more ITES that can be utilized even from the remote corners of the world.

Exercises

This Chapter has the support of multimedia content to understand more about the applications presented. You must see this content and where possible visit the websites indicated.

This multimedia content is provided to your school on a separate CD. Please contact your teacher to get this CD.

CHAPTER 12

COMPUTER ETHICS

Computer ethics has its roots in the work of Norbert Wiener during World War II. Wiener's book included

(1) An account of the purpose of a human life
(2) Four principles of justice
(3) A powerful method for doing applied ethics
(4) Discussions of the fundamental questions of computer ethics, and
(5) Examples of key computer ethics topics.

In the mid 1960s, Donn Parker of SRI International in Menlo Park, California began to examine unethical and illegal uses of computers by computer professionals. By the 1980s a number of social and ethical consequences of information technology were becoming public issues in America and Europe: issues like computer-enabled crime, disasters caused by computer failures, invasions of privacy via computer databases, and major law suits regarding software ownership.

During 1990s many universities introduced formal course in computer ethics. Many textbooks and other reading materials were developed. It triggered new research areas and introduction of journals.

Generally speaking, ethics is the set of rules for determining moral standards or what is considered as socially acceptable behaviors. Today, many computer users are raising questions on what is and is not ethical with regard to activities involving information technology. Obviously, some general guidelines on ethics are useful responsibly in their application of information technology.

General guidelines on computer ethics are needed for:

- Protection of personal data
- Computer Crime
- Cracking

12.1 Data Security

Personal data is protected by using an appropriate combination of the following methods.

Physical Security:

Physical security refers to the protection of hardware, facilities, magnetic disks, and other items that could be illegally accessed, stolen, damaged or destroyed. This is usually provided by restricting the people who can access the resources.

Personal Security:

Personal security refers to software setups that permit only authorized access to the system. User Ids and passwords are common tools for such purpose. Only those with a need to know have Ids and password for access.

Personnel Security:

Personnel security refers to protecting data and computer system against dishonesty or negligence of employees.

12.2 Computer Crime

A computer crime is any illegal activity using computer software, data or access as the object, subject or instrument of the crime.

Common crimes include:

- Crimes related to money transfer on the internet
- Making long distance calls illegally using computers
- Illegal access to confidential files
- Stealing hardware
- Selling or misusing personal
- Hardware and software piracy
- Virus
- Cracking
- Theft of computer time

It must be observed that 80% of all computer crimes happen from within the company. Over 60% of all crimes go unreported.

Making and using duplicate hardware and software is called piracy. We tend to pirate because:

- We like free things
- Why pay for something when we can get it for free?
- Our thinking and actions are self-serving
- If we have the opportunity to get away with something, benefit financially, and minimal risk is involved, the way in which we've been conditioned by our capitalist society to do it.

A virus is a self-replicating program that can cause damage to data and files stored on your computer. These are programs written by programmers with great programming skills who are motivated by the need for a challenge or to cause destruction. 57000 known virus programs are in existence. 6 new viruses are found each day.

Most of the computers in an organization have lot of free computer time to spare. In other words a lot of computer time is not used. Many solutions for using this spare time are being researched. However, this idle time of computers in an organization is being stolen illegally. Some other software runs on an idle computer without the knowledge of the organization. This is called theft of 'computer time'.

A commonly cited reference is the **Ten Commandments of Computer Ethics** written by the Computer Ethics Institute. This is given below.

- Thou shalt not use a computer to harm other people.
- Thou shalt not interfere with other people's computer work.
- Thou shalt not snoop around in other people's computer files.
- Thou shalt not use a computer to steal.
- Thou shalt not use a computer to bear false witness.
- Thou shalt not copy or use proprietary software for which you have not paid.

- Thou shalt not use other people's computer resources without authorization or proper compensation.
- Thou shalt not appropriate other people's intellectual output.
- Thou shalt think about the social consequences of the program you are writing or the system you are designing.
- Thou shalt always use a computer in ways that insure consideration and respect for your fellow humans.

Computer crimes require special laws to be formed by the government. Different countries have different ways of making the laws and awarding punishment to those who commit the crimes. India has Cyber laws to prevent computer crimes.

12.3 Cracking

Cracking is the illegal access to the network or computer system. Illegal use of special resources in the system is the key reason for cracking. The resources may be hardware, software, files or system information. Revenge, business reasons and thrill are other common reasons for committing this crime.

12.4 Work, Family and Leisure

Portable computers and telecommuting have created the condition where people can take their work anywhere with them and do it any time. As a result, workers find their work is cutting into family time, vacations, leisure, weakening the traditional institutions of family and friends and blurring the line between public and private life. This is becoming an important issue in computer ethics.

Exercises

1. What is the need for a password to log into a computer system?
2. How does the Operating System enhance the Security ?