UNIT-1

P. Madhuravani

Introduction to Object Oriented Programming, Structure of C++ program, Tokens, Keywords, Identifiers and Constants, Basic Data Types, Operators and Type Conversions.

Functions: Returning values from functions. Reference arguments, Overloaded function, Inline function, Default arguments, returning by reference.

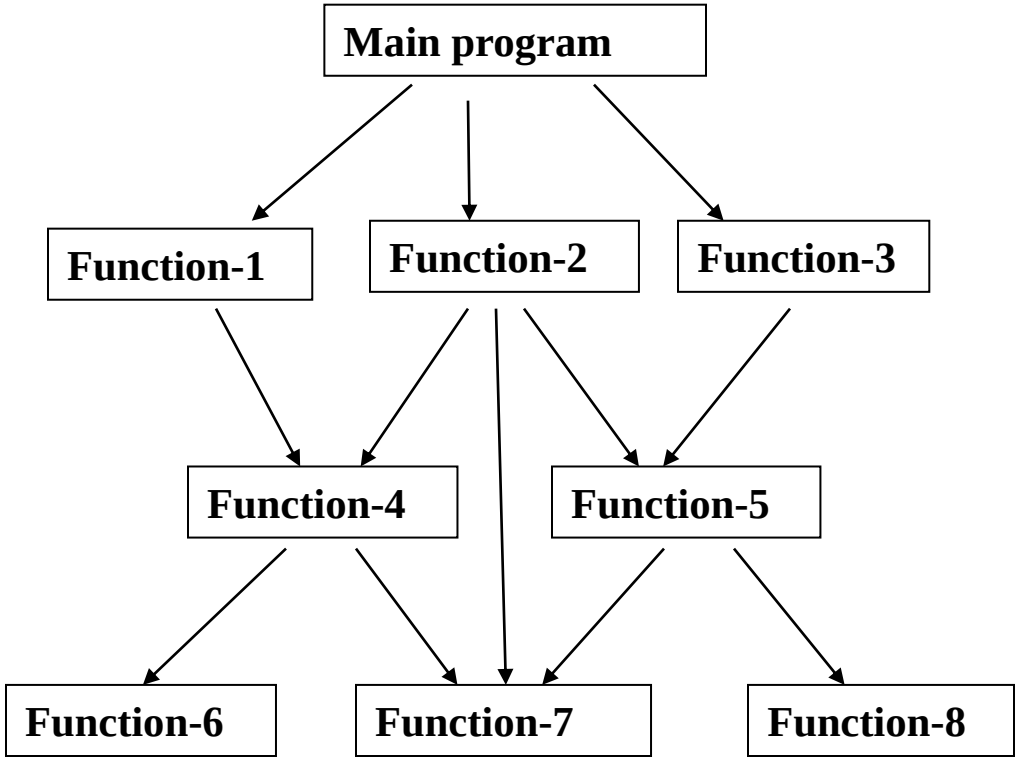
Introduction to Object Oriented Programming

History of C++

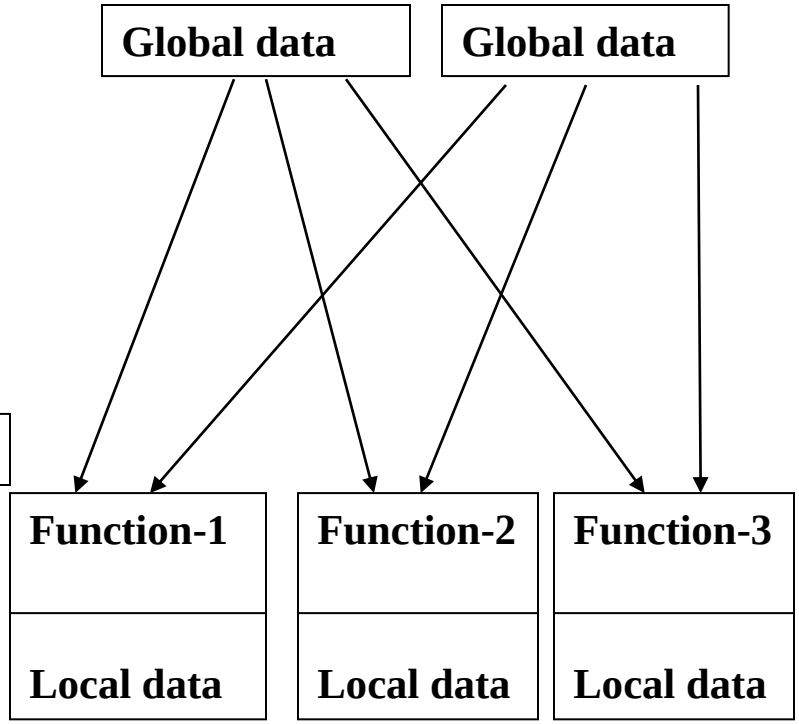
- Extension of C
- Early 1980s: Bjarne Stroustrup (Bell Laboratories)
- Originally named “C with Classes”
- Provides capabilities for object-oriented programming.
  - Objects: Reusable Software Components
    - Model items in real world
  - Object-Oriented Programs
    - Easy to understand, correct and modify
- Hybrid language
  - C-like style
  - Object-Oriented style
  - Both

Structured Oriented Programming:

- Program is divided into functions, every function has its **own data** and **global data**

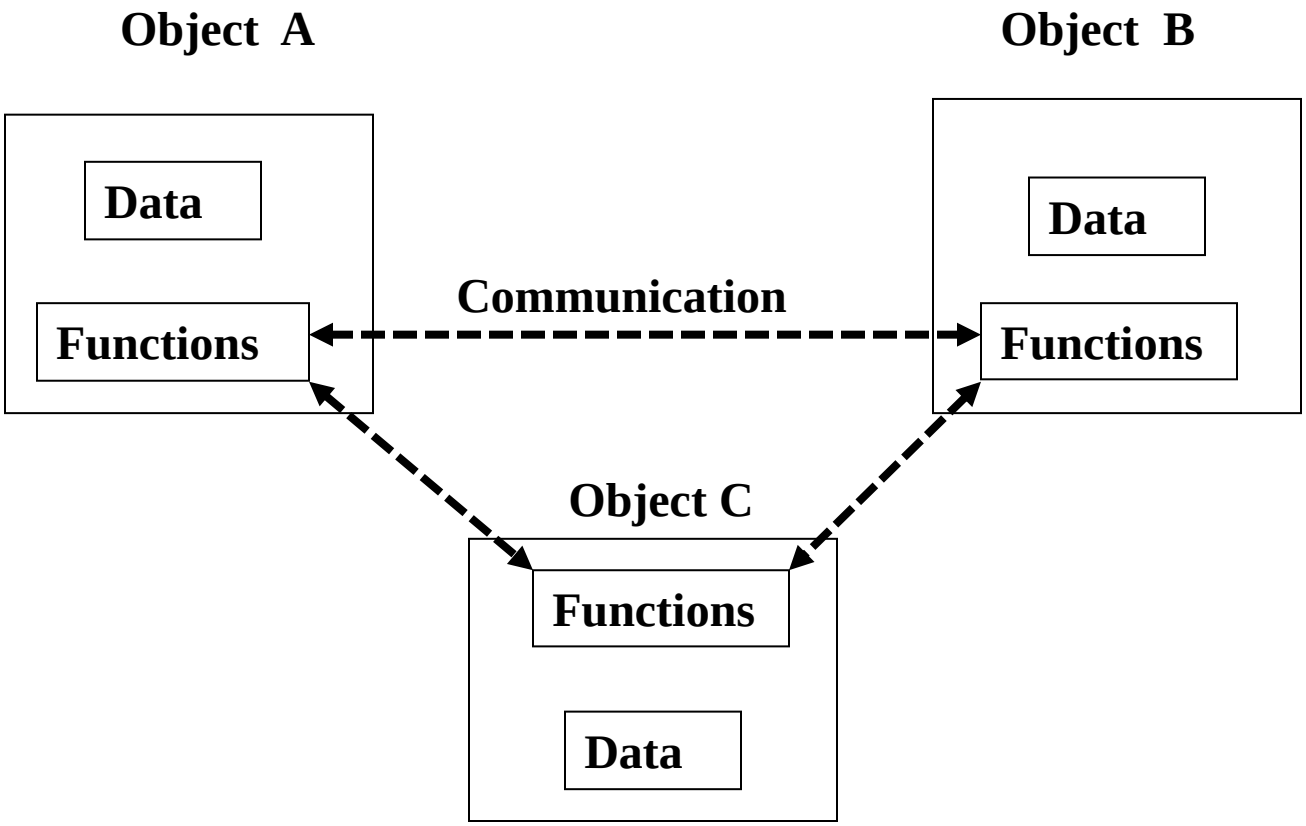


Relationship of Data & Functions in Structured Programming



Object Oriented Programming

OOP decompose the problem into number of entities called objects and it contains the data and functions.



Structured Vs OOP

Characteristics of Structure-Oriented Programming

- Emphasis is on doing **things.**
- Large programs are divided into smaller programs known as **functions.**
- Most of the functions **share global data**.
- Data **move openly around** the system from function to function
- Functions transform data from one form to another
- Employs **top-down approach** in program design

Characteristics of Object-Oriented Programming

- Emphasis is on **data** rather **than procedure.**
- Programs are divided into what are known as **objects.**
- Functions that operate on the data of an object are **tied together** in the data structure.
- Data **is hidden and cannot be accessed** by external functions.
- Objects may communicate with each other through functions.
- Employs **bottom-up approach** in program design
- **New data and functions** can be easily added whenever necessary.

Object Oriented Programming Concepts

- OOP is a programming style that is focused on objects.

- Object Oriented Programming Concepts
  - Classes
  - Objects
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Message Passing

Class and Object:

- A **class** is collection of objects of similar type or it is a template.

Ex: fruit mango;

class object

- **For example**, mango, apple and orange are members of the class fruit.
- Classes are user-defined data types and behave like the built-in types of a programming language.
- **Objects** are instances of the type class.
- Objects are the basic run-time entities in an object-oriented system.

Abstraction:

- **Abstraction** refers to the act of representing essential features without including the background details or explanations.
- The **access modifiers** in C++ or any OOP language, provides abstraction.
- If a variable is declared as **private**, then other classes cannot access it.

Encapsulation:

- The wrapping up of data and functions into a single unit (called class) is known as encapsulation.
- That is the data and method are given in the class definition.
- Data encapsulation is the most striking features of a class.

Inheritance:

- **Inheritance** is the process by which objects of one class acquire the properties of another class. The concept of inheritance provides the **reusability.**

- A class can inherit one or more classes. Inherited class is called as **parent** class or **super** class or **base** class. Class that inherits a parent class is called as **child** class or **sub** class or **derived** class.

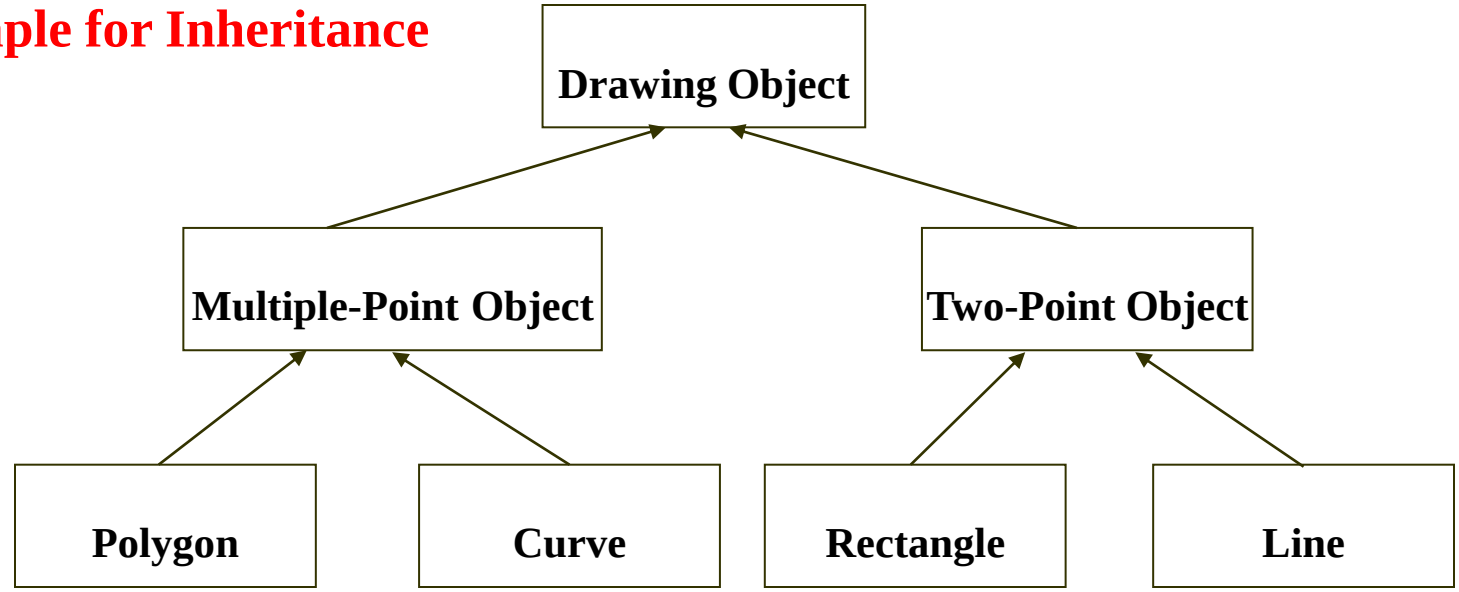
Polymorphism: **Poly – Many** **Morph – Form**

- Polymorphism is the characteristic that enables an **entity to co exist in more than one form.**

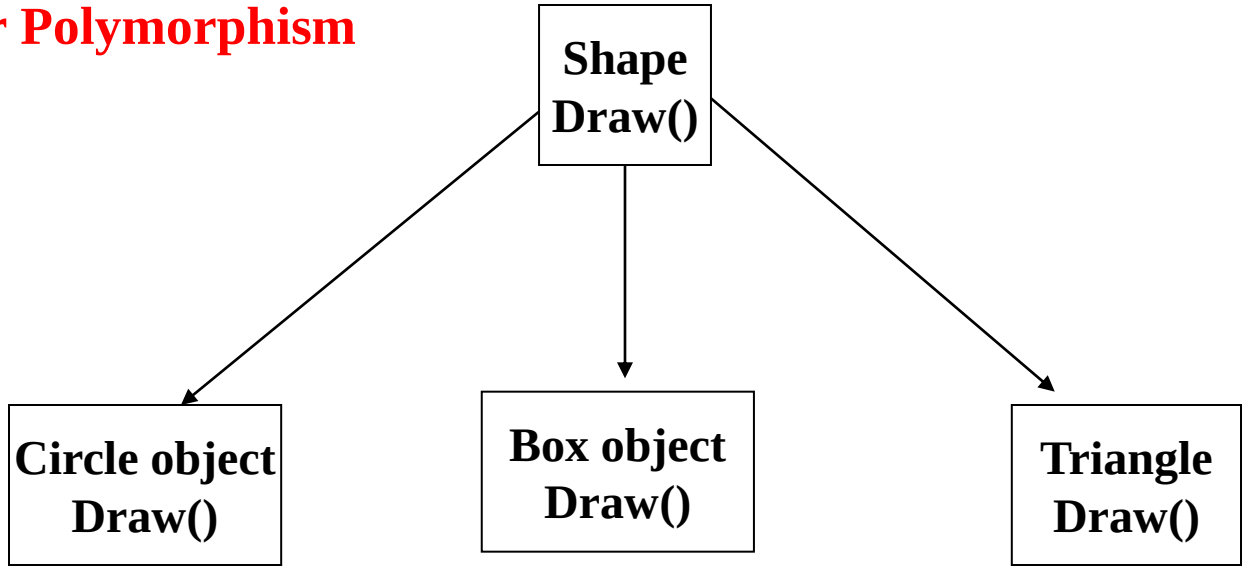
- **Ex:** It allows the single method to perform different actions based on the parameters.

- C++ supports **function overloading** and **operator overloading** to implement polymorphism

Example for Inheritance



Example for Polymorphism



Dynamic Binding:

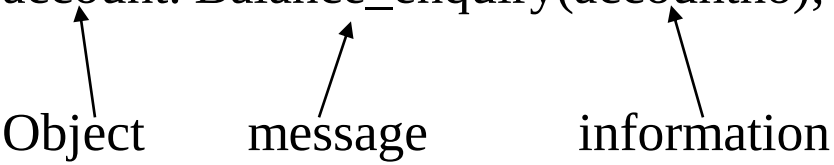
- When a method is called within a program, it associated with the program at run time rather than at compile time is called dynamic binding.

Message passing:

Set of objects that communicate with each other.

1. Create the classes that define objects and their behavior.
2. Creating the objects for that class.
3. Establish the communication among objects.

Example: account. Balance_enquiry(accountno);



Top Down approach:

- A Single module will be split into several smaller modules
- General to Specific

Bottom Up approach:

- Lot of small modules will be grouped to form a single large module
- Specific to General

- If the requirements are clear at the first instance we can go for **Top Down approach.**

- In circumstances where the requirements may keep on adding, we go for **Bottom Up approach**.

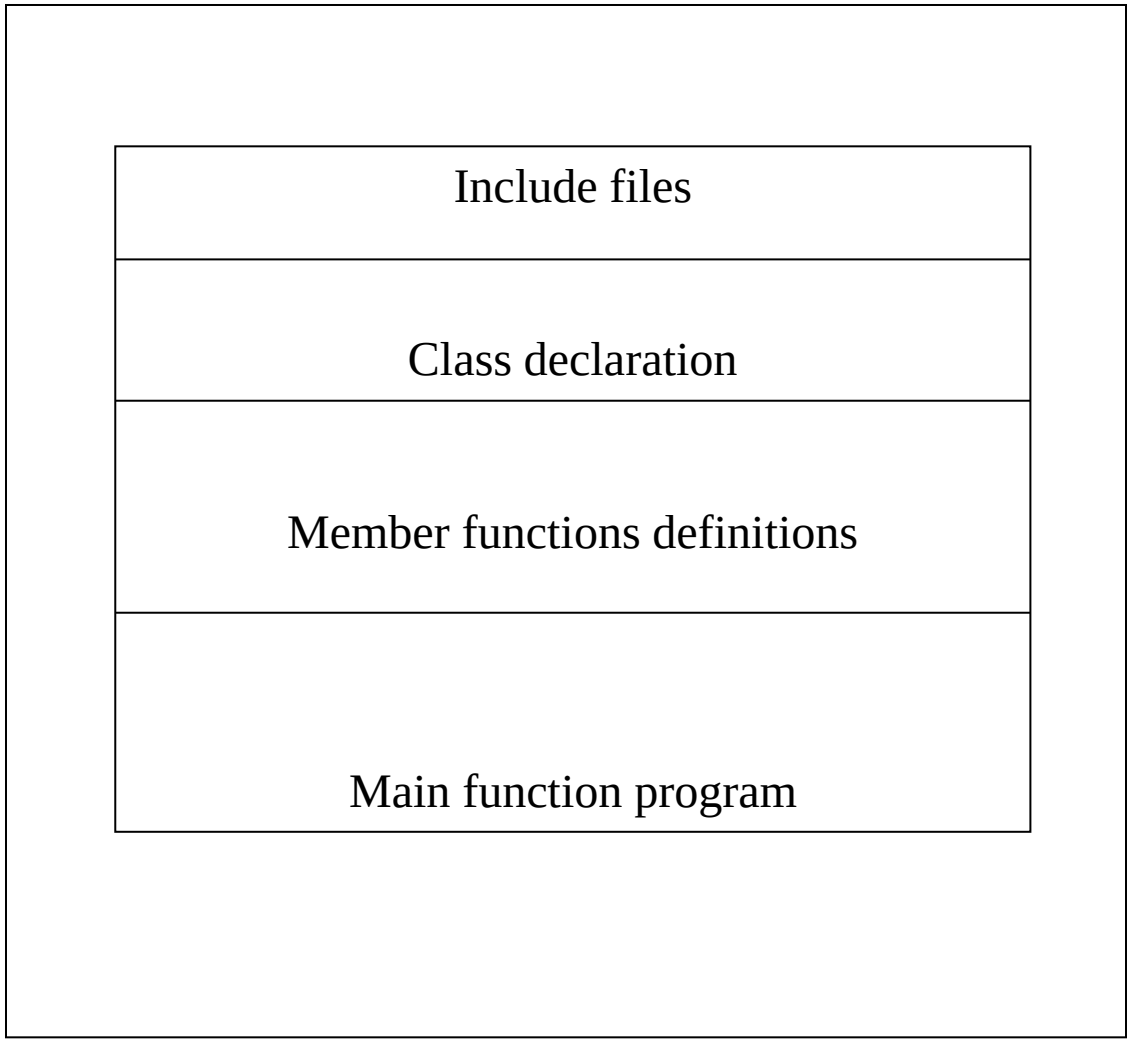
Benefits of OOP

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in the parts of the program.
- It is easy to partition the work in a project based on objects.
- Object oriented system easily upgraded from small to large systems.
- Software complexity can be easily managed.

Applications of OOP

- Real-time systems.
- Object-Oriented Databases.
- Neural Networks and Parallel Programming.
- Decision Support and Office Automation Systems.

Structure of C++ program:



- The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// some of its features
```

- The C comment symbols /*,*/ are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of
C++ program to illustrate
some of its features
*/
```

- We have used the following #include directive in the program:

```
#include <iostream>
```

- The #include directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file. The header file **iostream.h should be included at the beginning of all programs that use input/output statements.**

In C++, main () returns an integer value to the operating system. Therefore, every main () in C++ should end with a return (0) statement; otherwise a warning an error might occur. Since main () returns an integer type for main () is explicitly specified as int. Note that the default return type for all function in C++ is int.

I/O in C++

- Since C++ is a superset of C, all of the C I/O functions such as **printf** and **scanf** which are found in the **stdio.h** header file, are still valid in C++.
- C++ provides an alternative with the new stream input/output features by including "**iostream.h**".
- Several new I/O objects available when you include the **iostream** header file. Two important ones are:
  - cin // Used for keyboard input
  - cout // Used for screen output
- Both **cin** and **cout** can be combined with other member functions for a wide variety of special I/O capabilities in program applications.

- Since **cin** and **cout** are C++ objects, they are somewhat "intelligent":
  - They do not require the usual format strings and conversion specifications.
  - They do automatically know what data types are involved.
  - They do not need the **address operator, &.**
  - They do require the use of the stream **extraction (>>)** and **insertion (<<)** operators.
- Example with **cin** and **cout**:

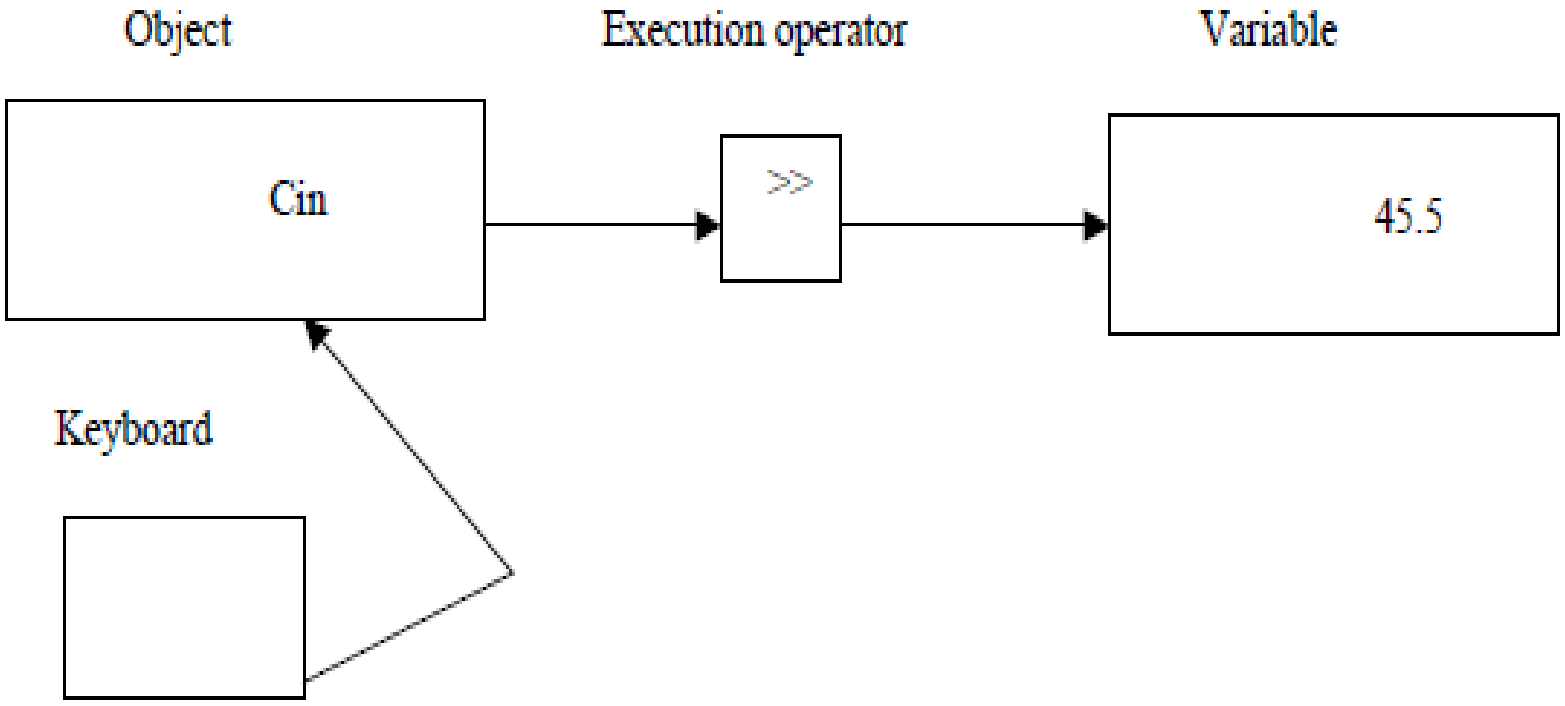
```
// program to find average of two numbers

#include<iostream.h>
void main()
{
        float n1,n2,avg;
        cout<<"Enter two values\n";
        cin>>n1>>n2;
        avg = (n1+n2)/2;
        cout<<"\nAverage is "<<avg;
 }
```

Input Operator

- The operator >> is known as extraction or get from operator.
- It extracts (or takes) the value from the keyboard and assigns it to the variable on its right fig 1.8.
- This corresponds to a familiar scanf() operation. Like <<, the operator >> can also be overloaded.

Input Operator



1.8 Input using extraction operator

Output operator

- The output statement is
- Cout<<"C++ is better than C.";

- Causes the string in quotation marks to be displayed on the screen.

- This statement introduces two new C++ features,
- cout and <<. The identifier cout(pronounced as C out) is a predefined object that represents the standard output stream in C++.

- Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices.

- The operator << is called the insertion or put to operator.

Output operator

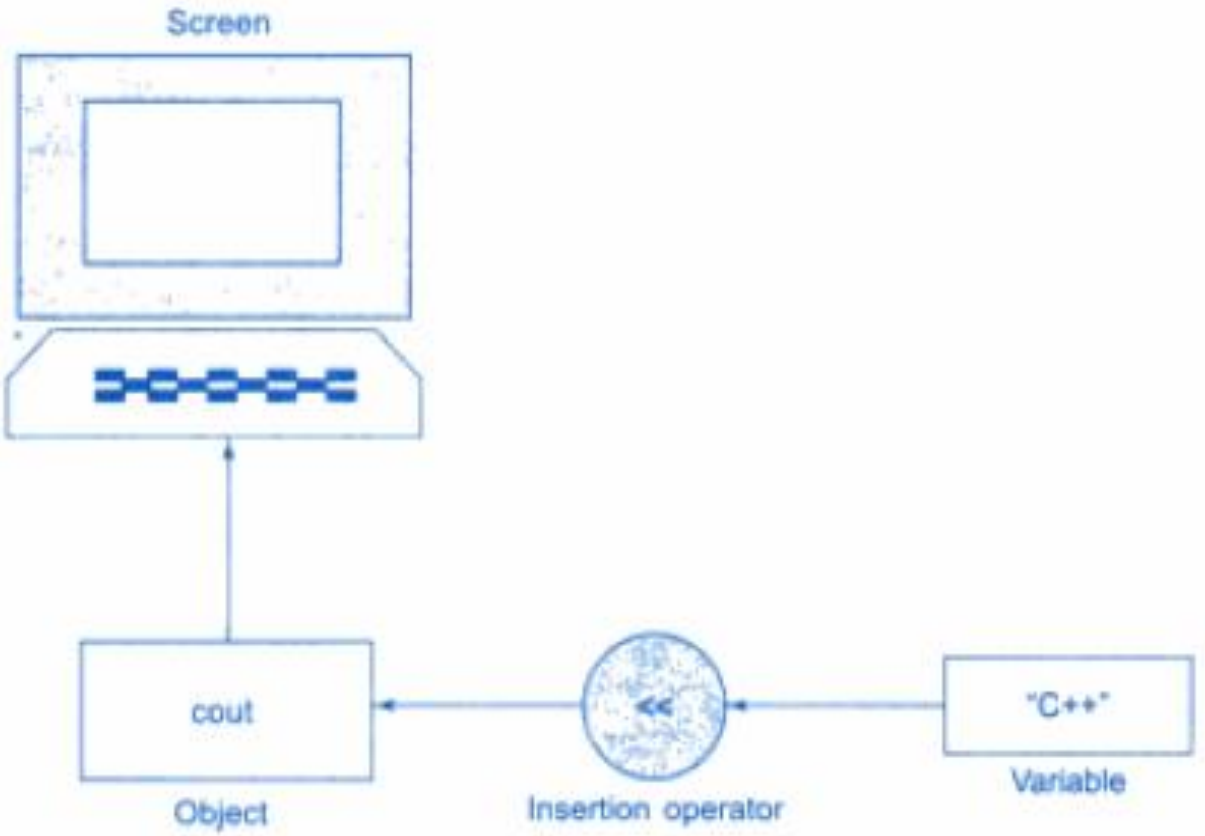


Fig. 2.1 ⇔ *Output using insertion operator*

Cascading of I/O Operators

- The statement

Cout << “Sum = “ << sum << “\n”;

- First sends the string “Sum = “ to cout and then sends the value of sum.
- Finally, it sends the newline character so that the next output will be in the new line.
- The multiple use of << in one statement is called cascading.
- When cascading an output operator, we should ensure necessary blank spaces between different items.
- Using the cascading technique, the last two statements can be combined as follows:

- Cout << “Sum = “ << sum << “\n”
- << “Average = “ << average << “\n”;

Tokens

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

The reserved words of C++ may be conveniently placed into several groups. In the first group we put those that were also present in the C programming language and have been carried over into C++. There are 32 of these, and here they are:

auto	const	double	float	int	short
struct	unsigned	break	continue	else	for
long	signed	switch	void	case	default
enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static
union	while				

Keywords

The reserved words of C++ may be conveniently placed into several groups. In the first group we put those that were also present in the C programming language and have been carried over into C++. There are 32 of these, and here they are:

auto	const	double	float	int	short
struct	unsigned	break	continue	else	for
long	signed	switch	void	case	default
enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static
union	while				

Keywords

Here are another 30 reserved words that were not in C, are therefore new to C++, and here they are:

asm	dynamic_cast	namespace	reinterpret_cast	try
bool	explicit	new	static_cast	typeid
catch	false	operator	template	typename
class	friend	private	this	using
const_cast	inline	public	throw	virtual
delete	mutable	protected	true	wchar_t:

Table 3.1 *C++ keywords*

asm	**double**	new	**switch**
auto	**else**	operator	template
break	**enum**	private	this
case	**extern**	protected	throw
catch	**float**	public	try
char	**for**	**register**	**typedef**
class	friend	**return**	**union**
const	**goto**	**short**	**unsigned**
continue	**if**	**signed**	virtual
default	inline	**sizeof**	**void**
delete	**int**	**static**	**volatile**
do	**long**	**struct**	**while**
Added by ANSI C++			
bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

Identifiers

Symbolic names can be used in C++ for various data items used by a programmer in his program. A symbolic name is generally known as an identifier. The identifier is a sequence of characters taken from C++ character set.

The rule for the formation of an identifier are:

- An identifier can consist of alphabets, digits and/or underscores.
- It must not start with a digit
- C++ is case sensitive that is upper case and lower case letters are considered different from each other.
- It should not be a reserved word.

Constants

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

Integer-Constants

Character-constants

Floating-constants

Strings-constants

Integer Constants

Integer constants are whole number without any fractional part. C++ allows three types of integer constants.

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

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

Hexadecimal integer constant: It consists of sequence of digits preceded by ox or OX

Character constants

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

Floating constants

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

String Literals

A sequence of character enclosed within double quotes is called a string literal. String literal is by default (automatically) added with a special character '\0' which denotes the end of the string. Therefore the size of the string is increased by one character. For example "COMPUTER" will re represented as "COMPUTER\0" in the memory and its size is 9 characters.

Punctuators

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

Brackets [ ] Opening and closing brackets indicate single and multidimensional array subscript.

Parentheses ( )Opening and closing brackets indicate functions calls,; function parameters for grouping expressions etc.

Braces { }Opening and closing braces indicate the start and end of a compound statement.

Comma , It is used as a separator in a function argument list.

Semicolon ; It is used as a statement terminator.

Colon : It indicates a labeled statement or conditional operator symbol.

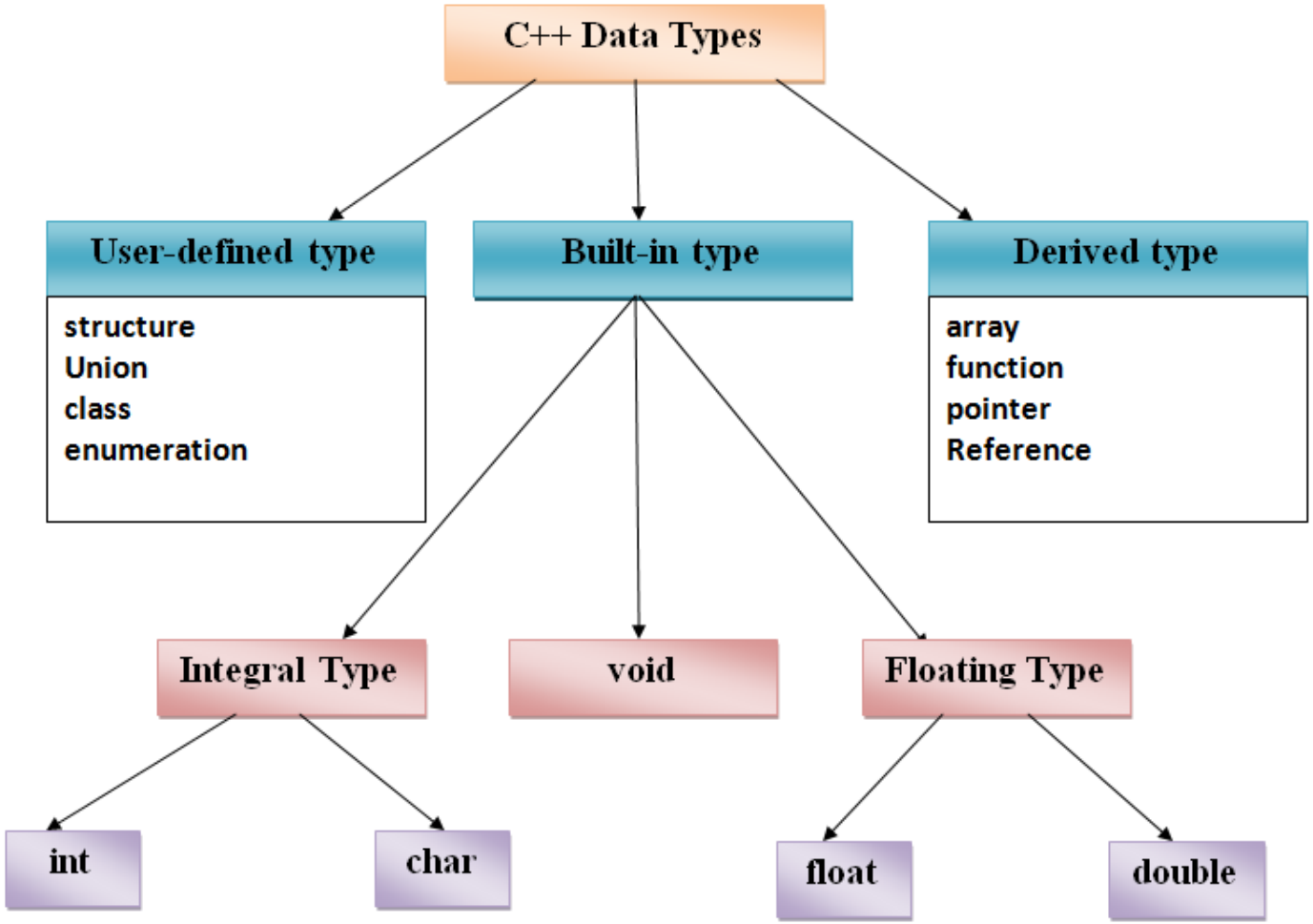
Asterisk * It is used in pointer declaration or as multiplication operator.

Equal sign = It is used as an assignment operator.

Pound sign #It is used as pre-processor directive.

Basic Data Types

C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character. C++ also provides the data type bool for variables that can hold only the values True and false.



Basic Data Types

Type	Typical Bit Width	Typical Range
char	1byte	-128 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-128 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

Operators

Operators are special symbols used for specific purposes. C++ provides many operators for manipulating data.

Generally, there are six type of operators : Arithmetical operators, Relational operators, Logical operators, Assignment operators, Conditional operators, Comma operator.

Arithmetical operators

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Operators

Relational operators

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

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

Operators

Logical operators

The logical operators are used to combine one or more relational expression. The logical operators are

Operators	Meaning
||	OR
&&	AND
!	NOT

Operators

Assignment operator

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side.

For example:

m = 5;
The operator takes the expression on the right, 5, and stores it in the variable on the left, m.

x = y = z = 32;
This code stores the value 32 in each of the three variables x, y, and z.

in addition to standard assignment operator shown above, C++ also support compound assignment operators.

Operators

Compound Assignment Operators

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

Operators

Increment and Decrement Operators

C++ provides two special operators viz '++' and '--' for incrementing and decrementing the value of a variable by 1. The increment/decrement operator can be used with any type of variable but it cannot be used with any constant. Increment and decrement operators each have two forms, pre and post.

The syntax of the increment operator is:

Pre-increment: ++variable
Post-increment: variable++

The syntax of the decrement operator is:

Pre-decrement: —variable
Post-decrement: variable—

Operators

Conditional operator

The conditional operator ?: is called ternary operator as it requires three operands. The format of the conditional operator is:
Conditional_ expression ? expression1 : expression2;
If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

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

The comma operator

The comma operator gives left to right evaluation of expressions. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

int a = 1, b = 2, c = 3, i; // *comma acts as separator, not as an operator* i = (a, b); // *stores b into i*

Operators

The sizeof operator

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

sizeof (char) returns 1 sizeof (float) returns 4

Operator Precedence

++, --(post increment/decrement)

++ (Pre increment) -- (Pre decrement),

sizeof ( ),

!(not),

-(unary),

+(unary)

*,/, %+, -

<, <=, >, >===,

!=&&? :=Comma operator

Type Conversion

When two operands of different data types are encountered in the same expression, the variable of lower data type is automatically converted to the data types of variable with higher data type, and then the expression is calculated.

For example: int a=98; float b=5;

cout<<a/3.0; //converts to float type, since 3.0 is of float type.

cout<<a/b; // converts a temporarily to float type, since b is of float type, and gives the result 19.6.

Type Casting

Type casting refers to the data type conversions specified by the programmer, as opposed to the automatic type conversions. This can be done when the compiler does not do the conversions automatically. Type casting can be done to higher or lower data type.

For example :

cout<<(float)12/5; //displays 2.4, since 12 is converted to float type.

Functions: Returning values from function

A function is a subprogram that acts on data and often returns a value. A program written with numerous functions is easier to maintain, update and debug than one very long program. By programming in a modular (functional) fashion, several programmers can work independently on separate functions which can be assembled at a later date to create the entire project.

Functions: Returning values from function

```
#include<iostream> using namespace std;
void area(float);
int main()
{
     float radius;
    cin>>radius;
     area(radius);
    return 0;
 }
void area(float r)
{    cout << “the area of the circle is” << 3.14*r*r << ”\n”;
}
```

Functions: Returning values from function

```
#include <iostream>
using namespace std;
int timesTwo(int num);   // function prototype int main() {      int number,
response;
       cout<<“enter a value\n”;
     cin>>number;
     response = timesTwo(number);
     cout<< "The answer is "<<response;      return 0;
 }
//timesTwo function
int timesTwo (int num)
 {
       int answer;
       answer = 2 * num;
       return (answer);
}
```

Reference Arguments

Arguments can be passed in two ways:

1. Call By Value
2. Call By Reference

Call By Value

In call by value method, the called function creates its own copies of original values sent to it. Any changes, that are made, occur on the function's copy of values and are not reflected back to the calling function.

Call By Reference

In call be reference method, the called function accesses and works with the original values using their references. Any changes, that occur, take place on the original values are reflected back to the calling code.

Consider the following program which will swap the value of two variables.

using call by reference

```
#include <iostream>
using namespace std;

void  swap(int &, int &);
int  main()
{
   int a=10,b=20;
   swap(a,b);
   cout<<a<<" "<<b;
   return 0;
}

void  swap(int &c, int &d)
{
   int t;
   t=c;
   c=d;
   d=t;
}
```

output:
20 10

using call by value

```
#include <iostream>
using namespace std;

void  swap(int , int );
int  main()
{
   int a=10,b=20;
   swap(a,b);
   cout<<a<<" "<< b;
   return 0; }

void  swap(int c, int d)
{
   int t;
   t=c;
   c=d;
   d=t;
}
```

output:
10 20

Inline Function

C++ **inline** function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Example:

```
inline int cube(int r)
{
   return r*r*r;
}
```

NOTE:

- Function is made inline by putting a word inline in the beginning.
- Inline function should be declared before main() function.
- It does not have function prototype.
- Only shorter code is used in inline function If longer code is made inline then compiler ignores the request and it will be executed as normal function.

```
#include <iostream>
using namespace std;
 inline int Max(int x, int y)
{
     return (x > y)? x : y;
 } // Main function for the program
int main( )
{
cout << "Max (20,10): " << Max(20,10) << endl;
cout << "Max (0,200): " << Max(0,200) << endl;
cout << "Max (100,1010): " << Max(100,1010) << endl;
return 0;
 }
```

Max (20,10): 20
Max (0,200): 200
Max (100,1010): 1010

Default Arguments

C++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to a parameter which does not have a mathching arguments in the function call. Default values are specified when the function is declared. The complier knows from the prototype how many arguments a function uses for calling.

Example :

```
float result(int marks1, int marks2, int marks3=75);
```

a subsequent function call average = result(60,70);

passes the value 60 to marks1, 70 to marks2 and lets the function use default value of 75 for marks3.

The function call

average = result(60,70,80); passes the value 80 to marks3.

Returning by Reference

In C++ Programming, we can pass values by reference and also you can return a value by reference.

```
#include <iostream>
using namespace std;
 int n;
int& test();
int main()
{
      test() = 5;
      cout<<n;
      return 0;
}
int& test()
{     return n;       }
```

- Ordinary function returns value but this function doesn't. Hence, you can't return constant from this function.
- You can't return a local variable from this function.