Government Arts College(Autonomous)
Coimbatore – 641018

Re-Accredited with 'A' grade by NAAC

Object Oriented Programmimg with C++

Dr. S. Chitra
Associate Professor
Post Graduate & Research Department of Computer Science
Government Arts College(Autonomous)
Coimbatore – 641 018.

Year	Subject Title	Sem.	Sub Code
2018 -19 Onwards	OBJECT ORIENTED PROGRAMMING WITH C++	III	18BCS33C

Objective:

•Learn the fundamentals of input and output using the C++ library
•Design a class that serves as a program module or package.
•Understand and demonstrate the concepts of Functions, Constructor and inheritance.

UNIT – I

Principles of Object Oriented Programming: Software Crisis - Software Evolution - Procedure Oriented Programming - Object Oriented Programming Paradigm - Basic concepts and benefits of OOP - Object Oriented Languages - Structure of C++ Program - Tokens, Keywords, Identifiers, Constants, Basic data type, User-defined Data type, Derived Data type – Symbolic Constants – Declaration of Variables – Dynamic Initialization - Reference Variable – Operators in C++ - Scope resolution operator – Memory management Operators – Manipulators – Type Cast operators – Expressions and their types – Conversions – Operator Precedence - Control Structures

UNIT – II

Functions in C++: Function Prototyping - Call by reference - Return by reference - Inline functions - Default, const arguments - Function Overloading – Classes and Objects - Member functions - Nesting of member functions - Private member functions - Memory Allocation for Objects - Static Data Members - Static Member functions - Array of Objects - Objects as function arguments - Returning objects - friend functions – Const Member functions .

UNIT – III

Constructors: Parameterized Constructors - Multiple Constructors in a class - Constructors with default arguments - Dynamic initialization of objects - Copy and Dynamic Constructors - Destructors - Operator Overloading - Overloading unary and binary operators – Overloading Using Friend functions – manipulation of Strings using Operators.

UNIT – IV

Inheritance: Defining derived classes - Single Inheritance - Making a private member inheritable – Multilevel, Multiple inheritance - Hierarchical inheritance - Hybrid inheritance - Virtual base classes - Abstract classes - Constructors in derived classes - Member classes - Nesting of classes.

UNIT – V

Pointers, Virtual Functions and Polymorphism: Pointer to objects – this pointer- Pointer to derived Class - Virtual functions – Pure Virtual Functions – C++ Streams –Unformatted I/O- Formated Console I/O – Opening and Closing File – File modes - File pointers and their manipulations – Sequential I/O – updating a file :Random access –Error Handling during File operations – Command line Arguments.

TEXT BOOKS

1.E. Balagurusamy, “Object Oriented Programming with C++”, Fourth edition, TMH, 2008.

- **UNIT – I : Principles of Object Oriented Programming**

Software Crisis :

- Developments in software technology continue to be dynamic.
- New tools and techniques are announced in quick succession.
- This has forced the software engineers and industry to continuously look for new approaches to software design and development.
- These rapid advances appear to have created a situation of crisis within the industry.

The following issues need to be addressed to face this crisis:

- How to represent real-life entities of problems in system design?
- How to design systems with open, interfaces?
- How to ensure reusability and extensibility of modules?
- How to develop modules that are tolerant to any changes in future?
- How to improve software productivity and decrease software cost?
- How to improve the quality of software?
- How to manage time schedules?
- How to industrialize the software development process?

Some of the quality issues that must be considered for critical software evaluation are:

- **1. Correctness**
- **2. Maintainability**
- **3. Reusability**
- **4. Openness and interoperability**
- **5. Portability**
- **6. Security**
- **7. Integrity**
- **8. User friendliness**

Software Evolution:

- Since the invention of the computer, many programming approaches have been used.
- The Primary motivation in all programming styles is the concern to handle the increasing complexity of programs that are reliable and maintainable.
- These Programming techniques include
- 1. Machine Level Programming
- 2. Assembly Language Programming
- 3. High Level Programming

Machine level Language :

- Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU).
- Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory.
- Every program directly executed by a CPU is made up of a series of such instructions.

Assembly level Language :

- An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions.

- Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

High level Language :

- High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture.

- High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

- The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog.

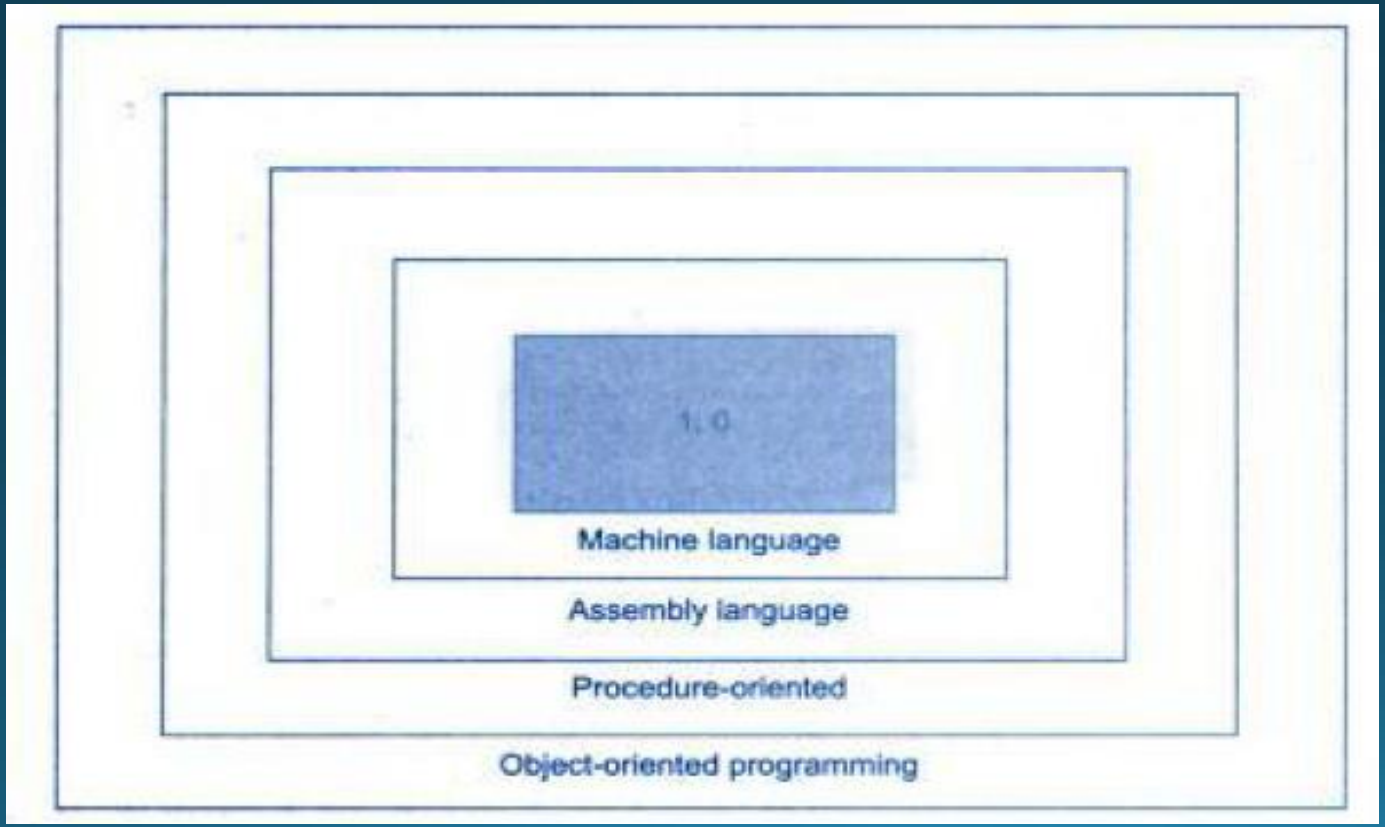
- Such languages are considered high-level because they are closer to human languages and farther from machine languages.

- In contrast, assembly languages are considered low- level because they are very close to machine languages.

High-Level programming approaches are broadly classified as:

1. Procedure-Oriented Programming (POP) &

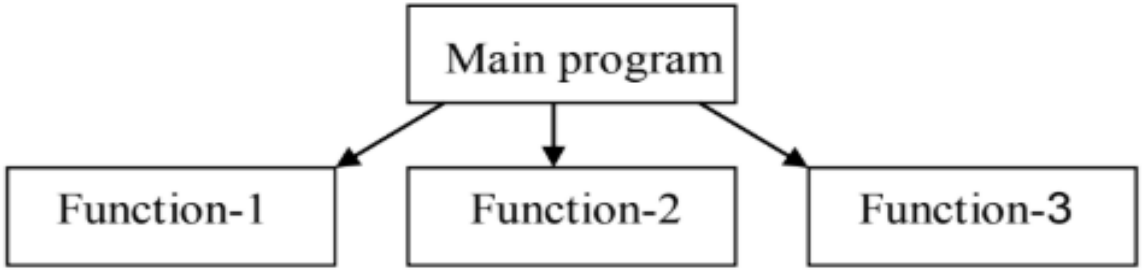
2. Object-Oriented Programming (OOP).



Procedure-Oriented Programming (POP)

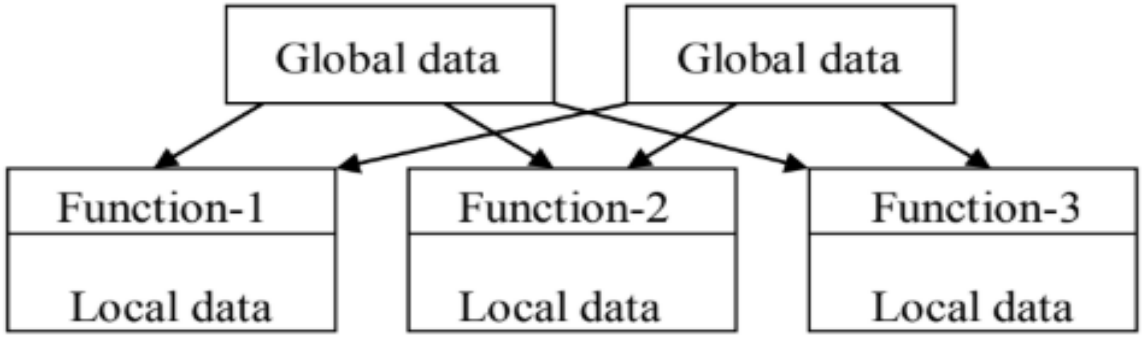
- Conventional programming, using high level languages such as COBOL, FORTRAN and C, are commonly known as pro*cedure-oriented pro*g*ram*mi*ng* (POP). In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing.
- A number of functions are written to accomplish these tasks. The primary focus is on functions.

Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.



The disadvantage of the procedure oriented programming languages is:

1. Global data access
2. It does not model real word problem very well
3. No data hiding



Analysis of Procedure-oriented-Programming

- Concentration is on the development of functions, very little attention is given to the data that are being used by various functions.
- Data move freely between functions where there is chance for the intruders to hake the data.
- Critical applications should be designed in such a way that the data should be protected when they are transferred between functions

Object-Oriented Programming Paradigm

- The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach mainly providing data hiding feature.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.

Object-Oriented Programming (OOP)

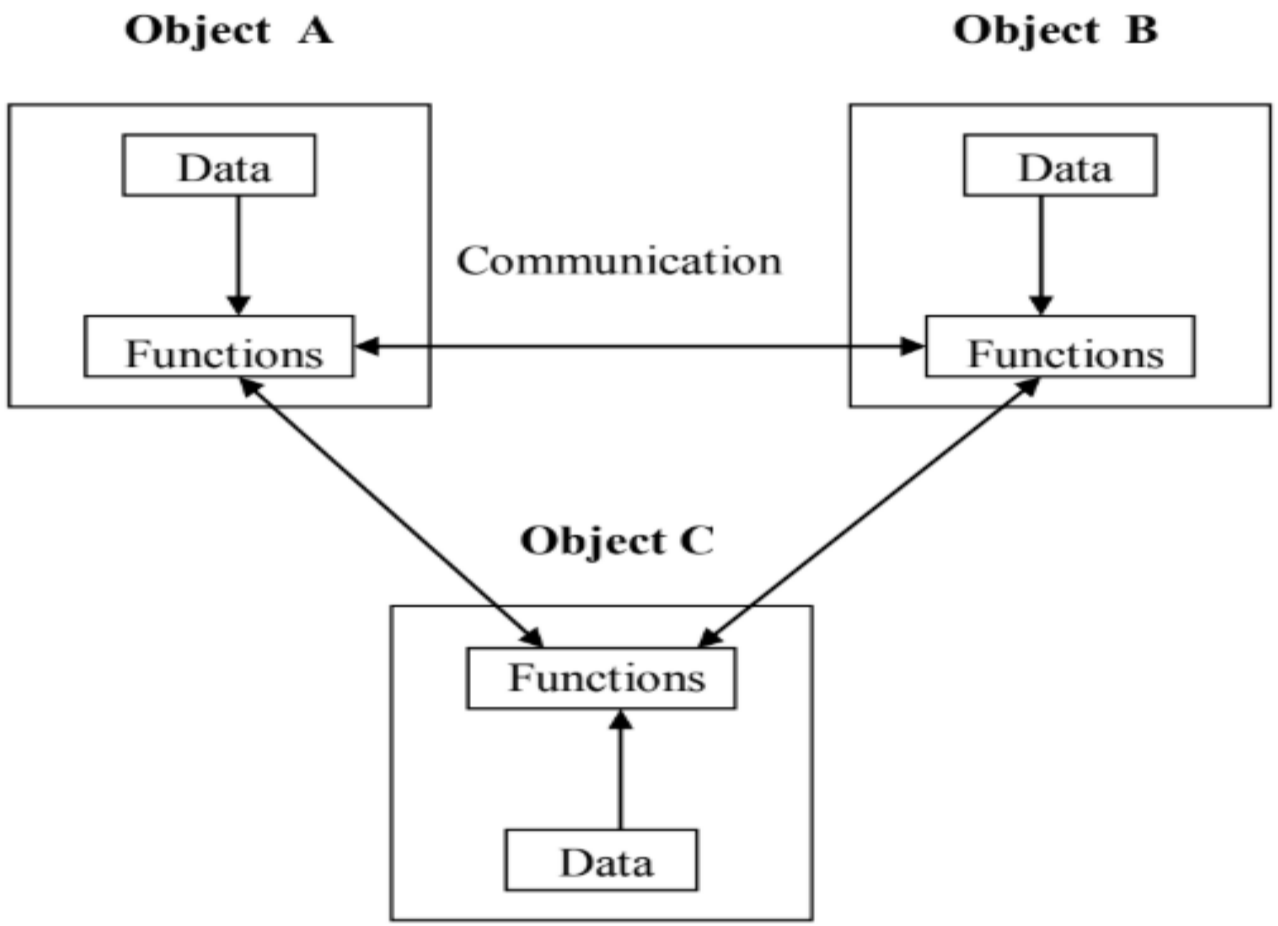
- OOP allows decomposition of a problem into a number of entities called o*bject*s and then builds data and functions around these objects.

- Data of an object can be accessed only by the functions associated with that object.

- However, functions of one object can access the functions of other objects.

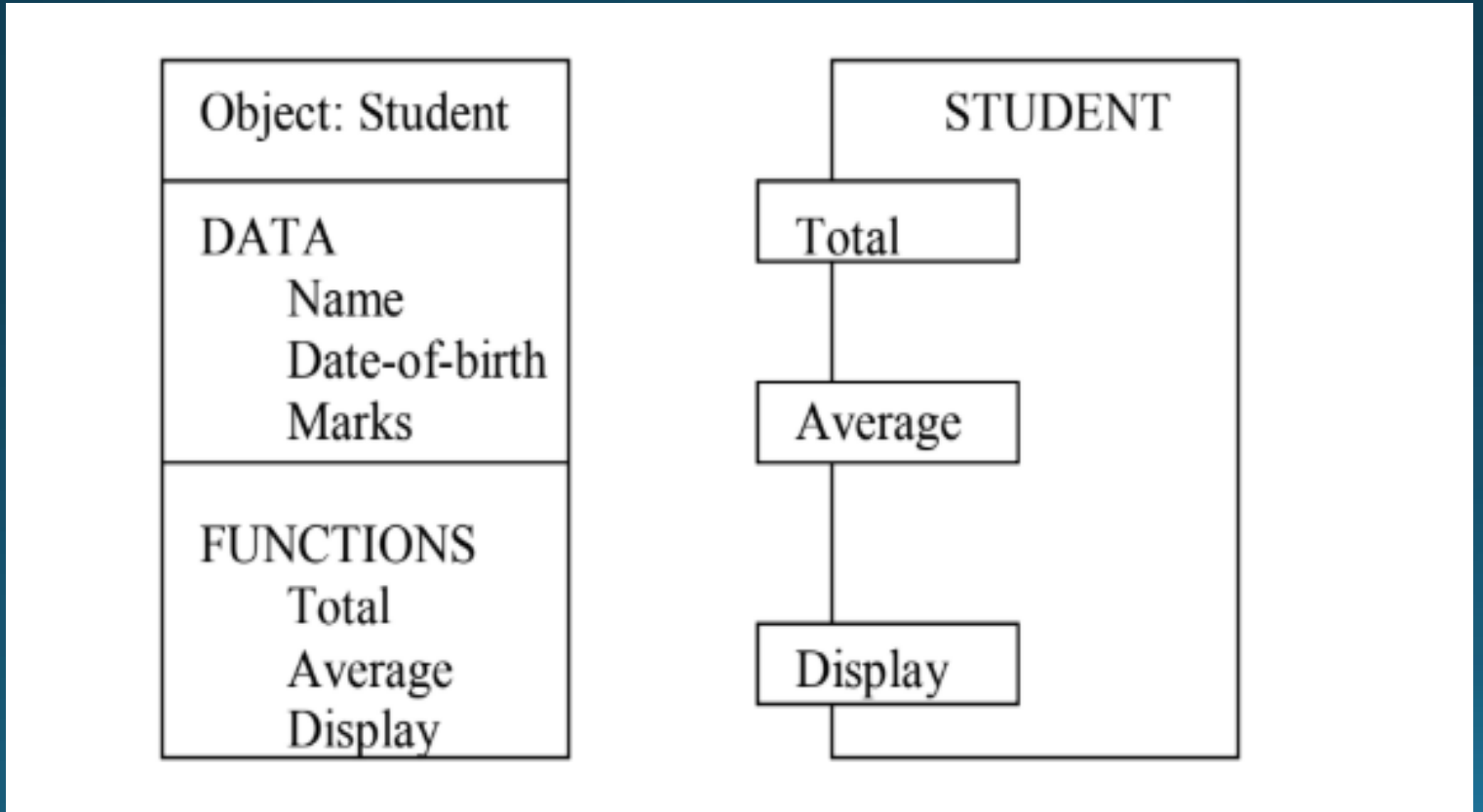
Object Oriented Programing

"Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand".



For example if student is considered as an object, then
1. **DATA** related to student (Name, Date-of-birth and Marks) and
2. **FUNCTIONS** (calculating the total,calculating average and displaying the result) are **combined** to form a student **object**

Object : Student



Some characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- 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.

Some of the striking features of object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the 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.
- New data and functions can be easily added whenever necessary.
- *Follows bottom-up* approach in program design.

BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING

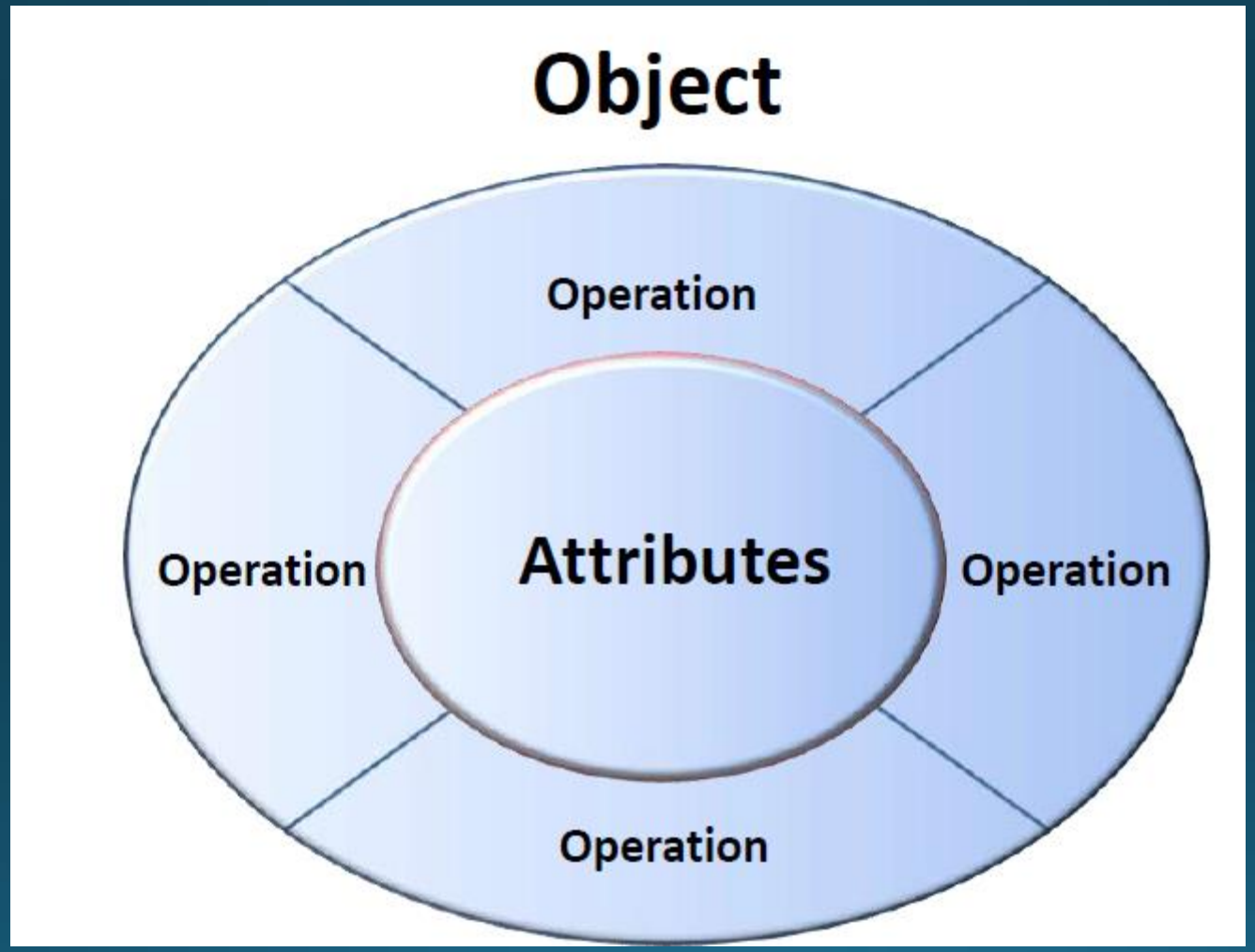
1. Objects
2. Classes
3. Data abstraction and encapsulation
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

OBJECTS

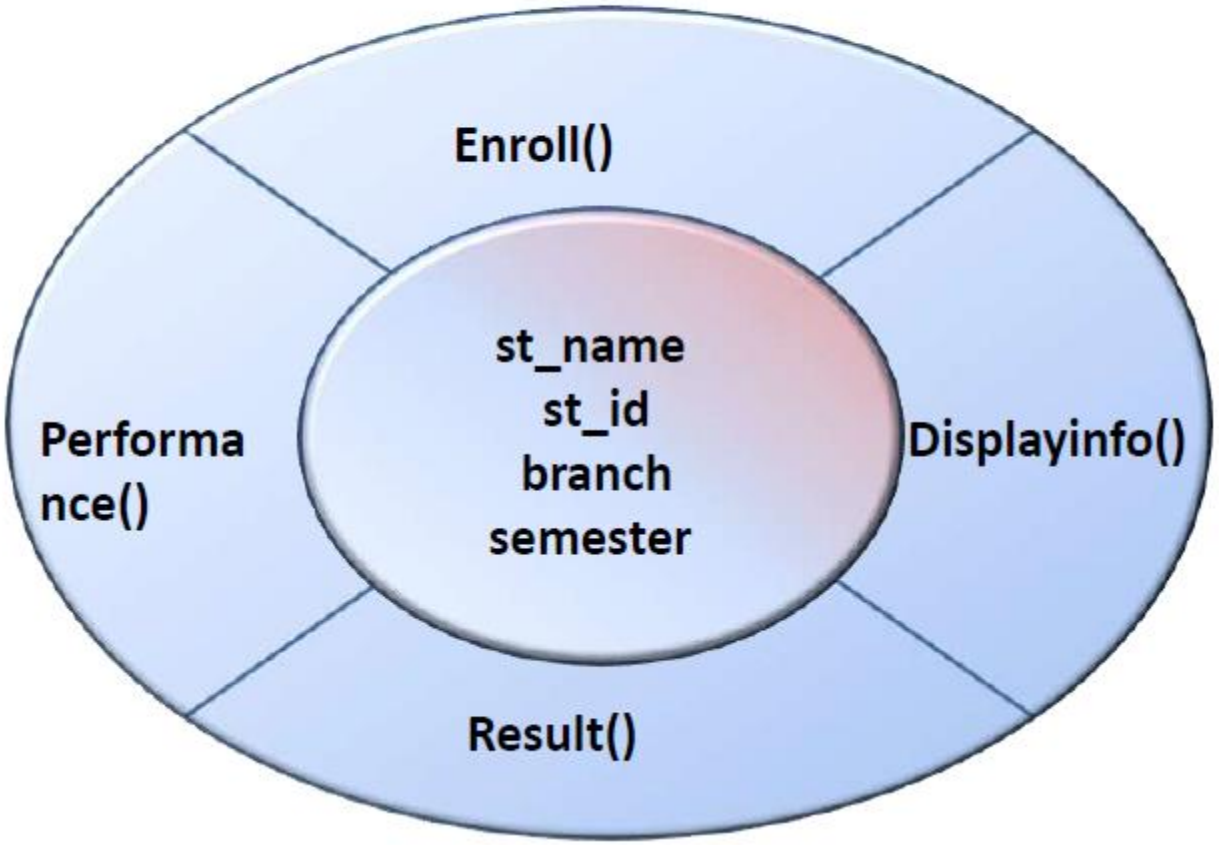
- Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program must handle.

- The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

- The term objects means a combination of data and program that represent some real word entity.


Object
Operation
Operation
Attributes
Operation
Operation

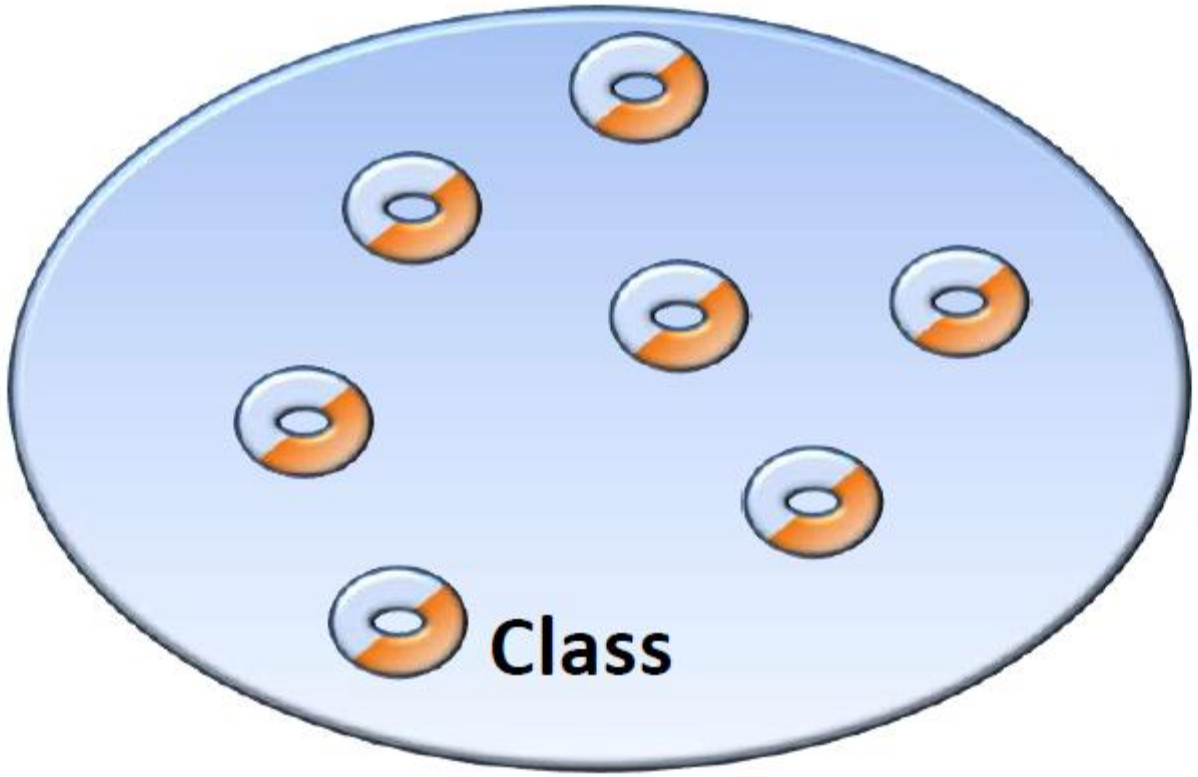
Example: StudentObject

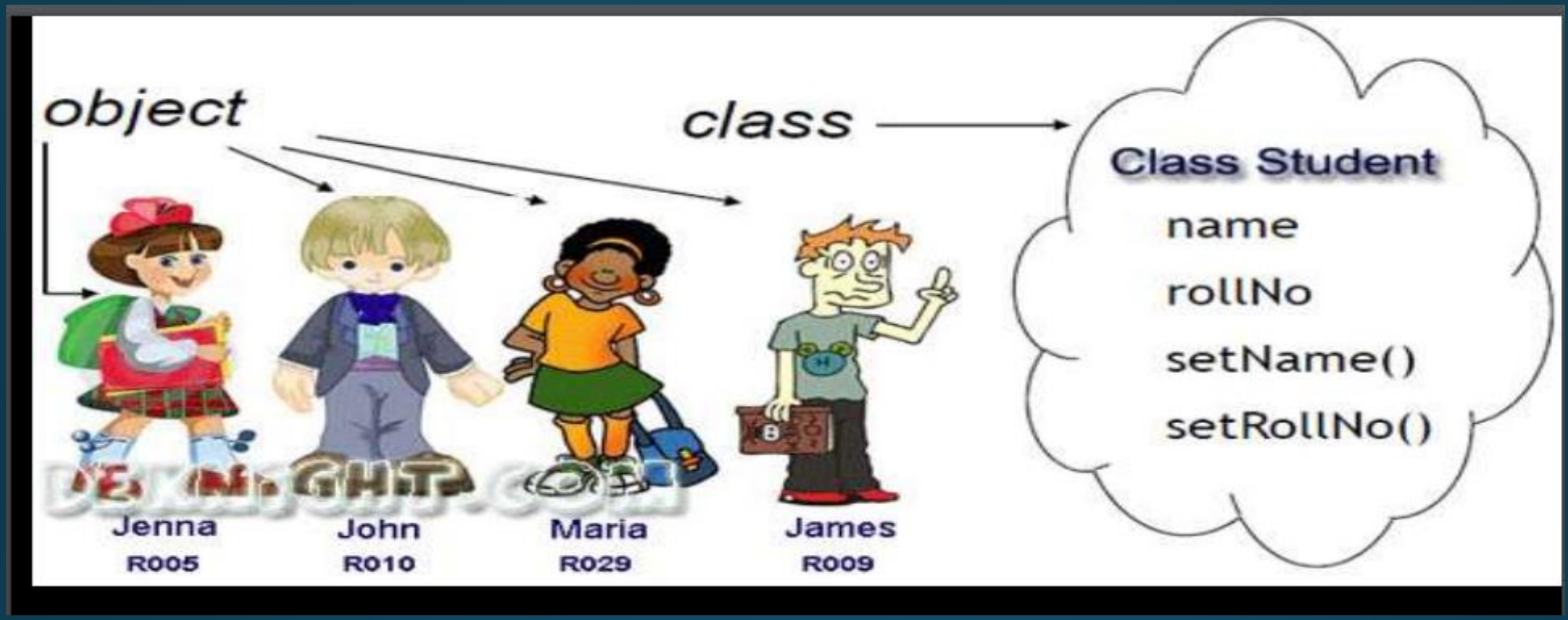


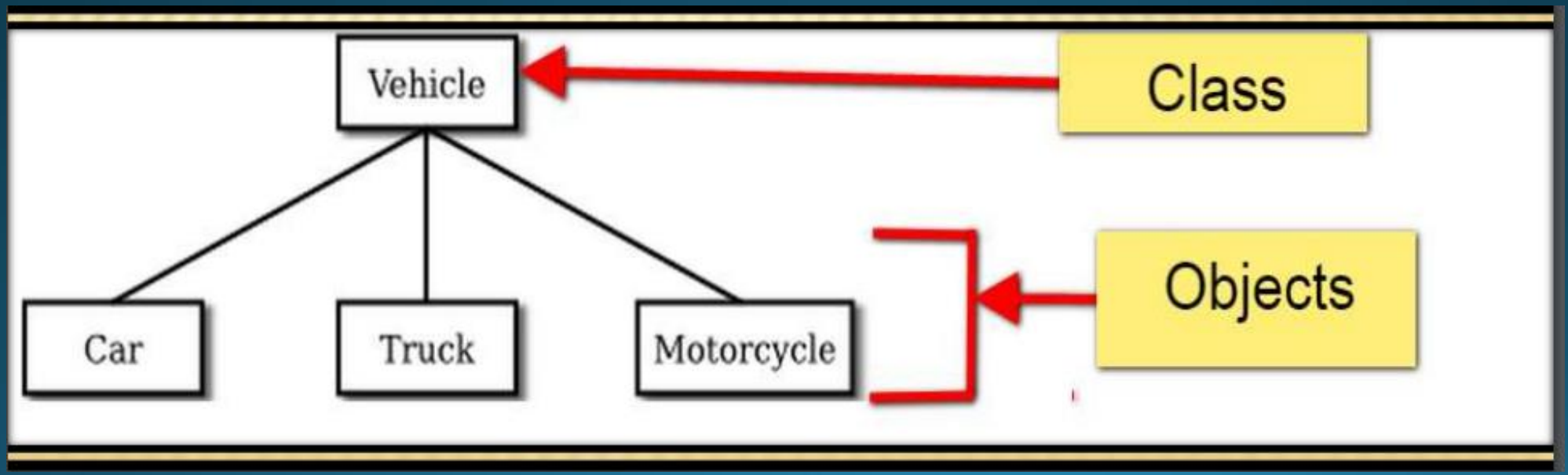
- **CLASS:**
- A group of objects that share common properties for data part and some program part are collectively called as class.
- In C ++ a class is a new data type that contains member variables and member functions that operate on the variables.

Class

- Class is a collection of **similar objects**.




object
class
Class Student
name
rollNo
setName()
setRollNo()
Jenna
R005
John
R010
Maria
R029
James
R009


Vehicle
Class
Car
Truck
Motorcycle
Objects

	Class	Object
1	Class is a container which collection of variables and methods.	object is a instance of class
2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.

DATA ABSTRACTION :

- Abstraction refers to the act of representing essential features without including the back ground details or explanations.

- Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

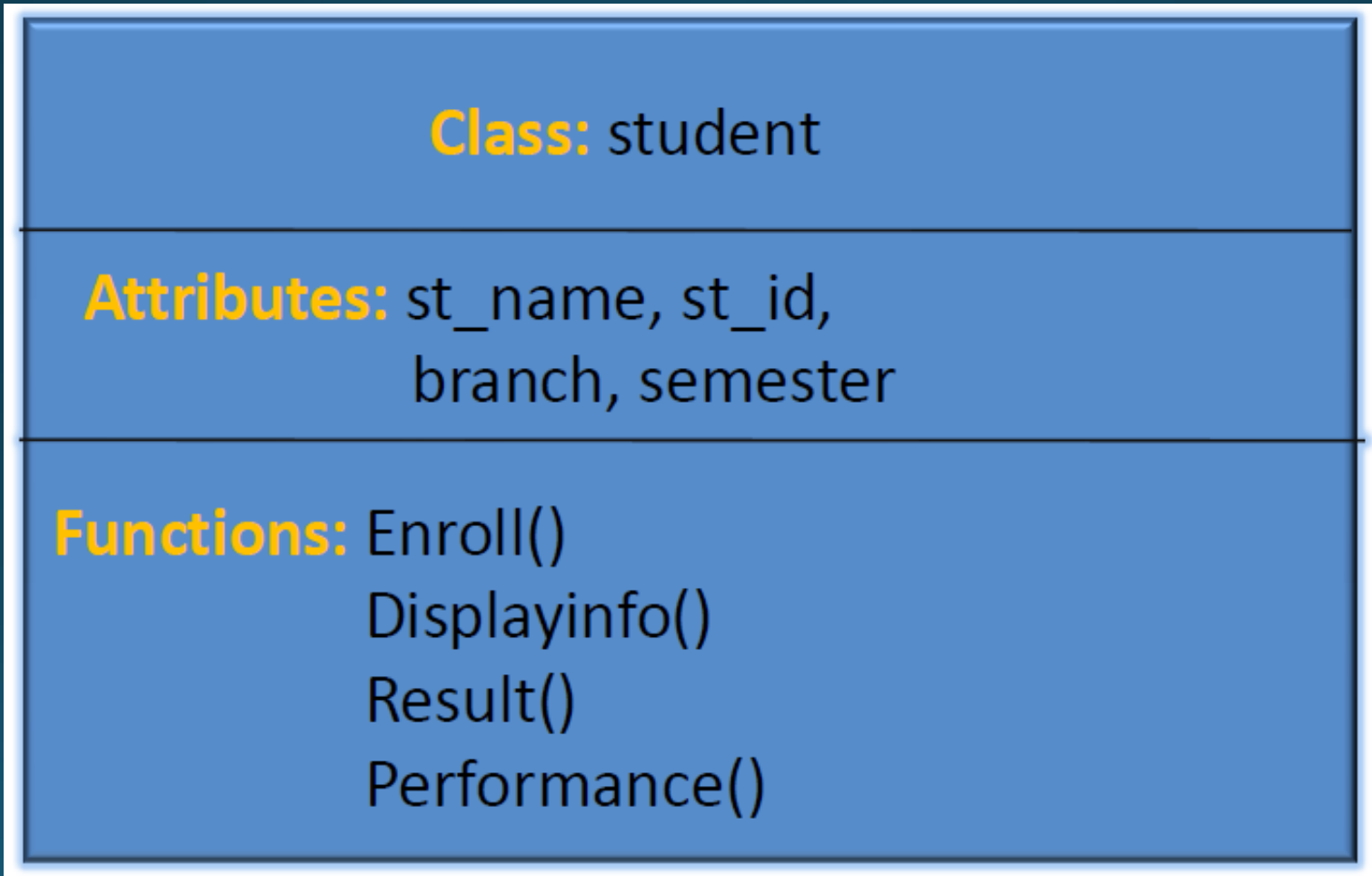
Real life example of Abstraction

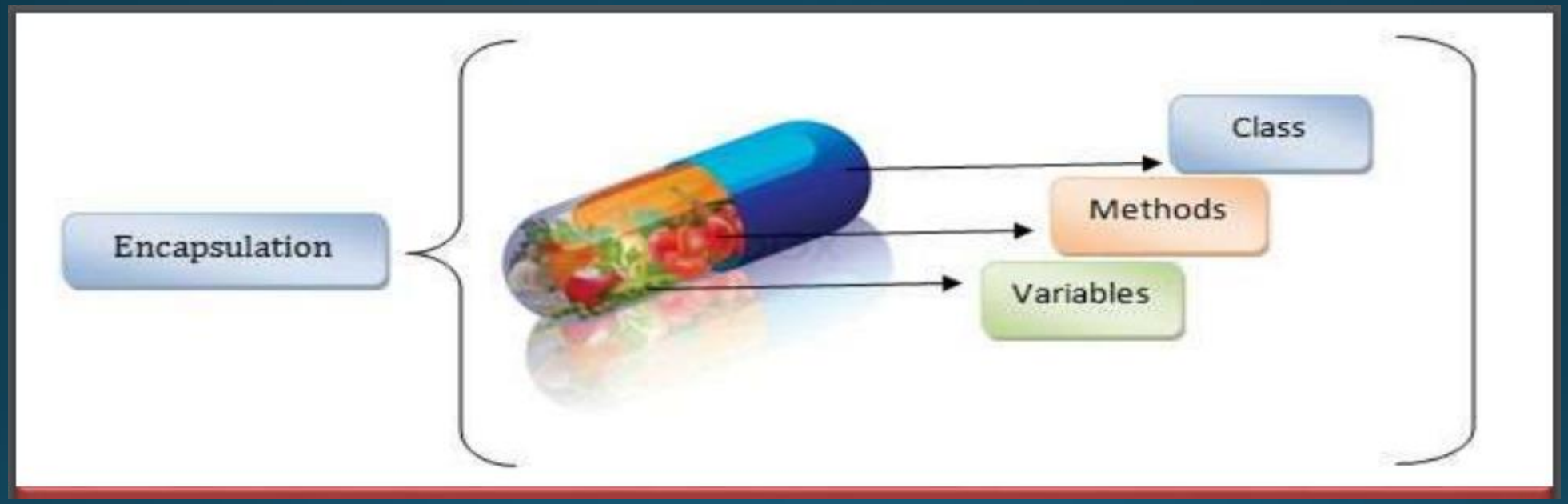
Abstraction shows only important things to the user and hides the internal details for example when we ride a bick, we only know about how to ride bick but can not know about how it work ? and also we do not know internal functionality of bick.

- **DATA ENCAPSALATION :**
- The wrapping up of data and function into a single unit (called class) is known as encapsulation.
- The data is not accessible to the outside world and only those functions which are wrapped in the class can access it.
- These functions provide the interface between the objects data and the program.

Encapsulation

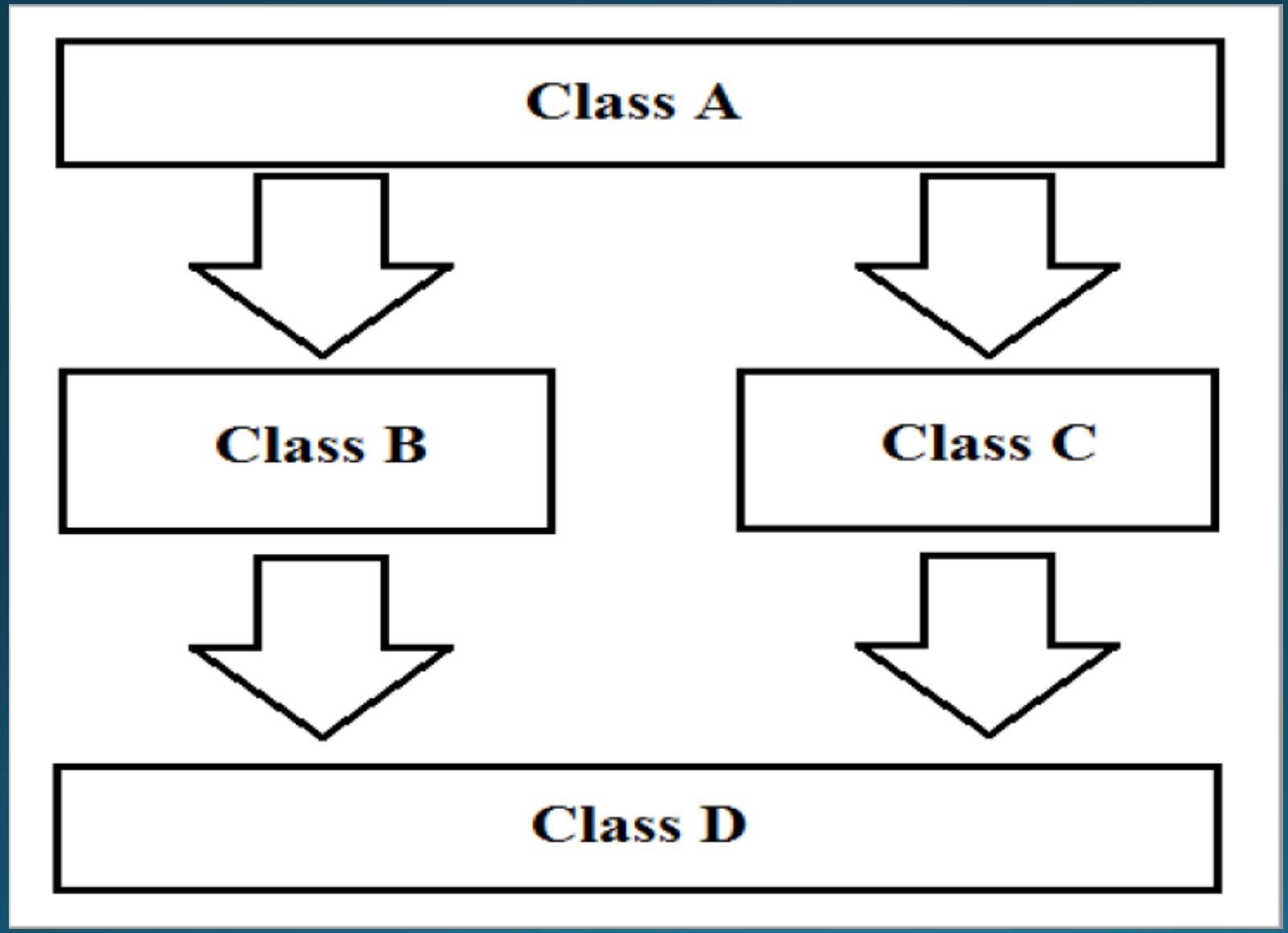
"Mechanism that associates the **code** and the **data** it manipulates into a single unit and keeps them safe from external interference and misuse."



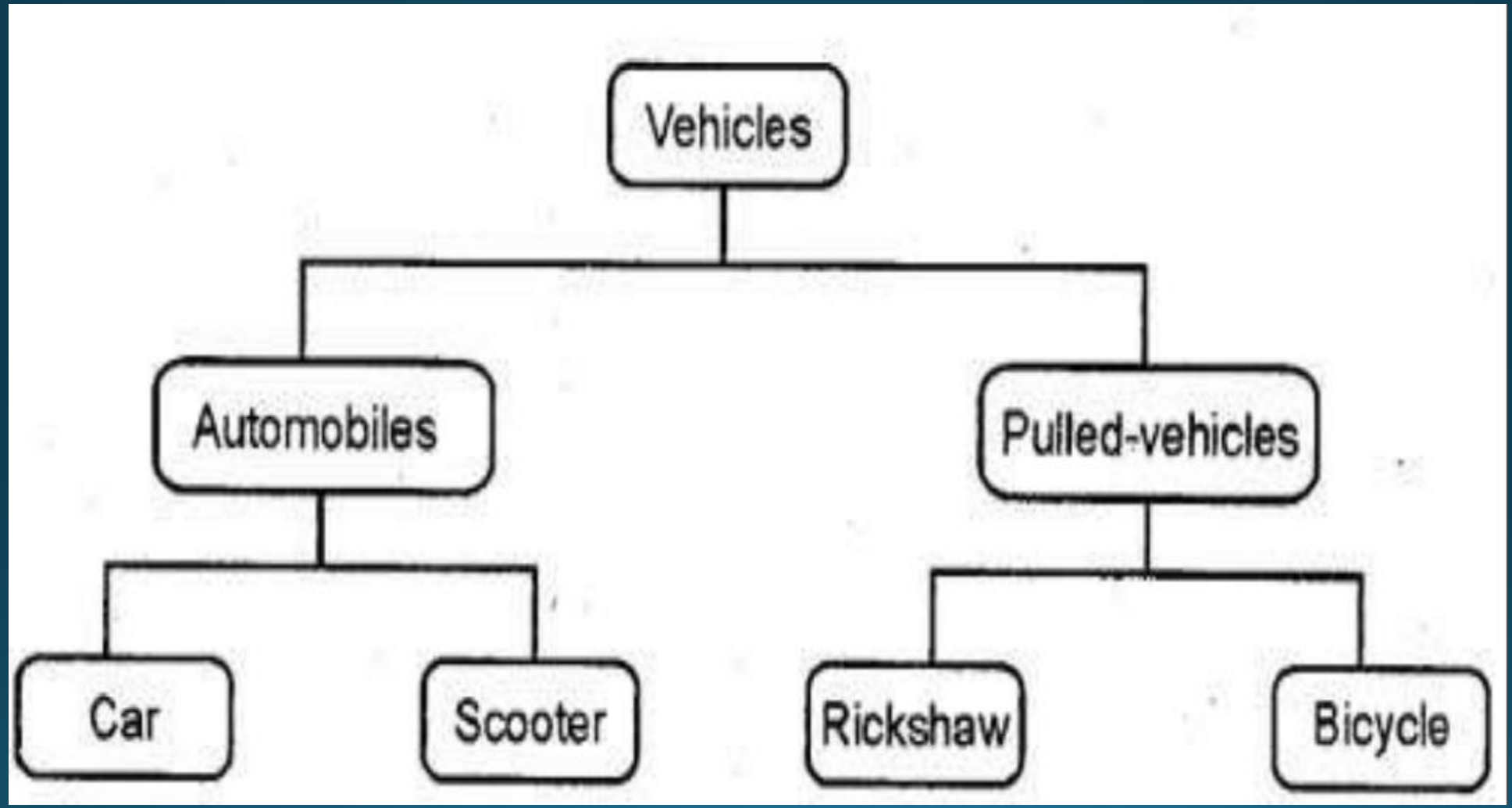

Encapsulation
Class
Methods
Variables

- **INHERITENCE :**
- Inheritance is the process by which objects of one class acquire the properties of another class.
- In the concept of inheritance provides the idea of reusablity.
- This mean that we can add additional features to an existing class with out modifying it.
- This is possible by desining a new class will have the combined features of both the classes.

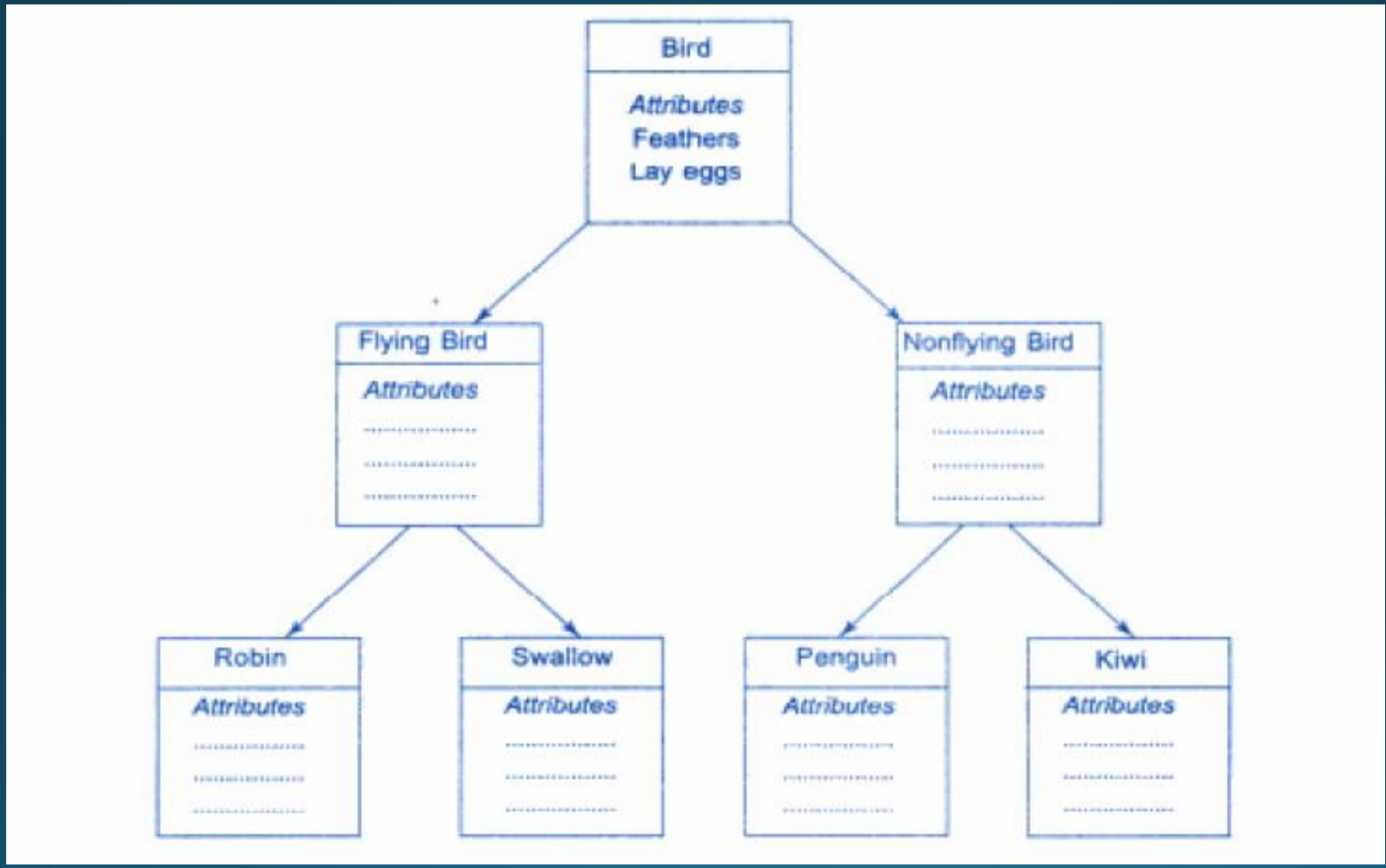
INHERITENCE



INHERITENCE



INHERITENCE



POLYMORPHISIM:

- Polymorphism means the ability to take more than one form.

Meaning of Polymorphism

Poly → Multiple

Morphing → Actions

- A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used.

POLYMORPHISIM Cont.

- Overloading may be operator overloading or function overloading.
- It is able to express the operation of addition by a single operater say ‘+’. When this is possible you use the expression x + y to denote the sum of x and y, for many different types of x and y; integers , float and complex no. You can even define the + operation for two strings to mean the concatenation of the strings.

Real life example of polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.

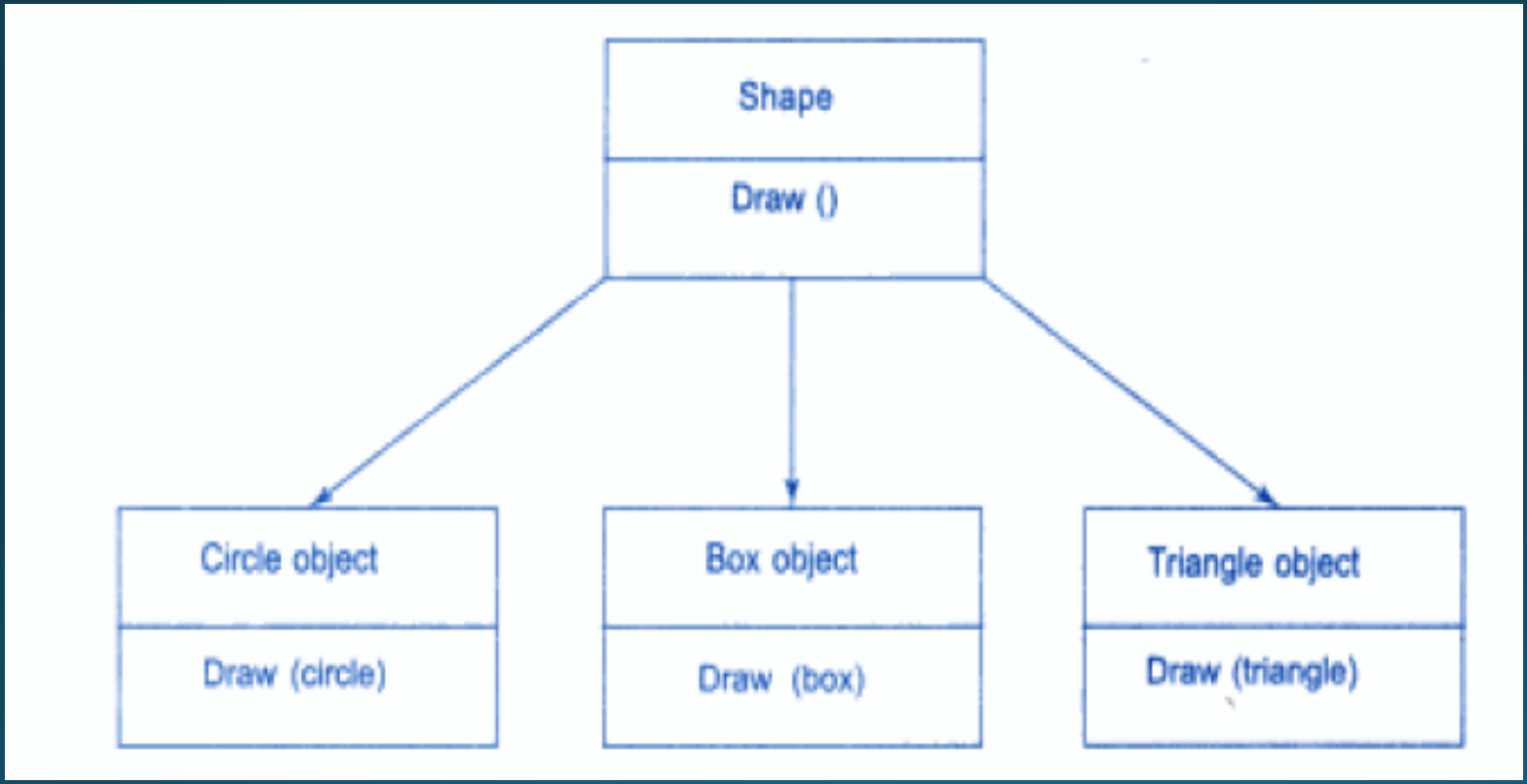
In Shopping malls behave like Customer

In Bus behave like Passenger

In School behave like Student

At Home behave like Son

POLYMORPHISIM



BINDING A FUNCTION CALL:

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.

```
void show();      /* Function declaration */
main()
{
      .....
      show();           /* Function call */
      .....
}
void show()       /* Function definition */
{
      .....


     .....        /* Function body */
     .....
}
```

DYNAMIC BINDING :

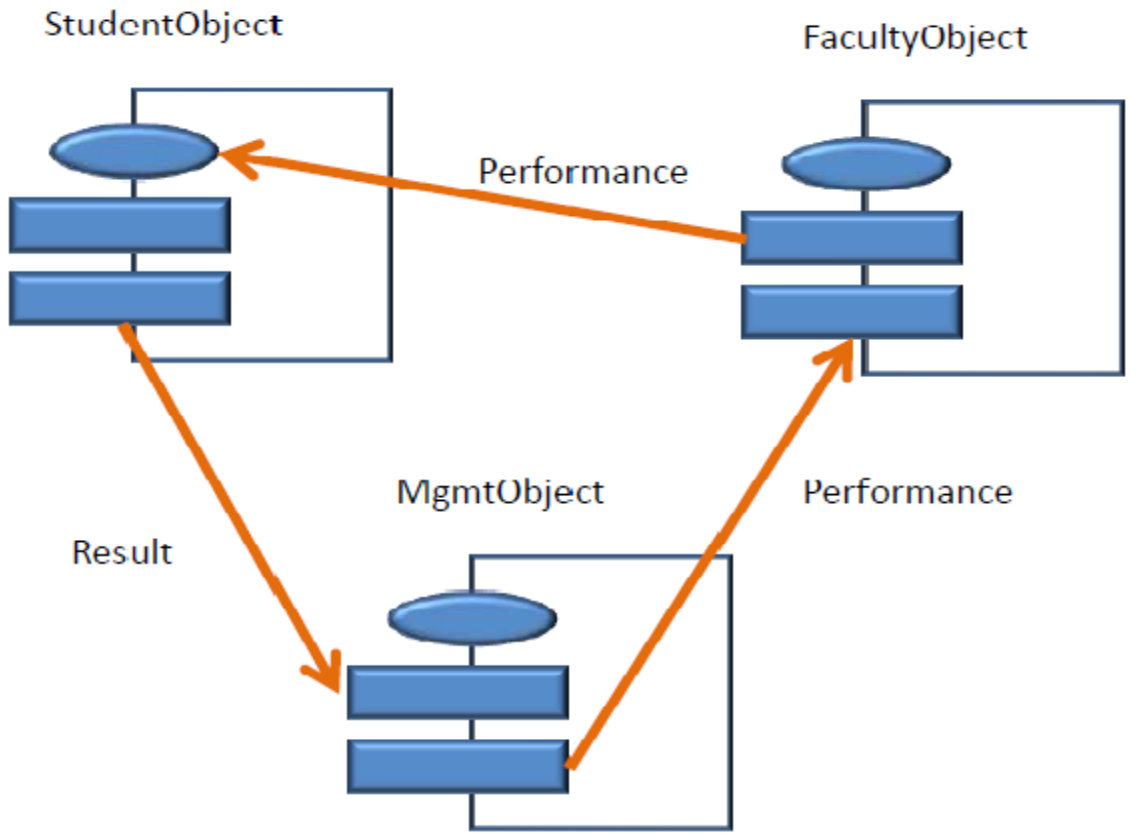
- Dynamic binding means the code associated with a given procedure call is not known until the time of the call at run-time.

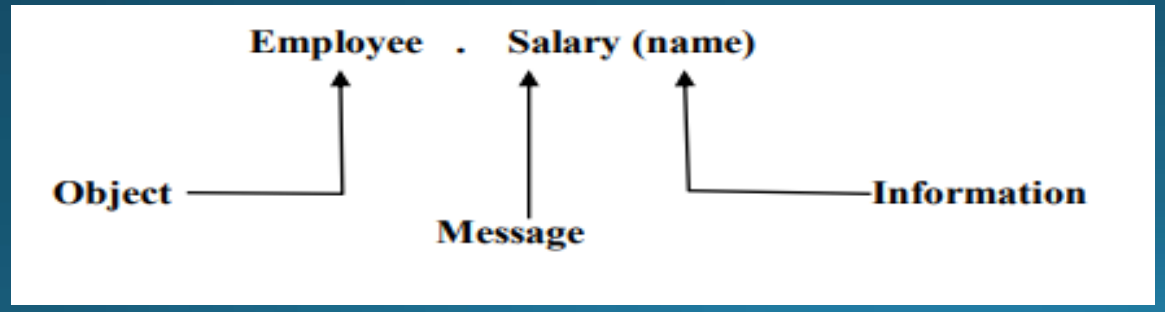
- It is associated with a polymorphic reference depends upon the dynamic type of that reference.

MESSAGE PASSING :

- An object oriented program consists of a set of objects that communicate with each other.
- A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result.
- Message passing involves specifying the name of the object, the name of the function (message) and information to be sent.

Message Passing





BENEFITS OF OOP:

1. Through inheritance redundant code can be eliminated. and extend the use of existing classes.

2. Code reusability leads to saving of development time and higher productivity.

3. The principle of data hiding helps the programmer to build secure programs that can’t be invaded by code in other parts of the program.

BENEFITS OF OOP Cont.

4. It is possible to have multiple instances of an object to co-exist with out any interference.

5. It is easy to partition the work in a project based on objects.

6. Object-oriented systems can be easily upgraded from small to large systems.

BENEFITS OF OOP Cont.

7. Message passing techniques for communication between objects makes the interface description with external systems much simpler.

8. Software complexity can be easily managed.

APPLICATION OF OOP:

The most popular application of oops up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using oop techniques.

The promising areas for application of oop includes.

1. Real – Time systems.
2. Simulation and modeling
3. Object oriented databases.

APPLICATION OF OOP Contd.

4. Hypertext, Hypermedia and Expertext.

5. Artificial intelligence and expert systems.

6. Neural networks and parallel programming.

7. Decision support and office automation systems.

8. CIM / CAM / CAD system.

Object Oriented Languages

1. Object-Based programming Languages

Support : 1. Data Encapsulation
2. Data Hiding & Access Mechanism
3. Automatic Initialization & Clear-up of objects
4. Operator Overloading

Not support: 1. Inheritance &
2. Dynamic Binding

Example: Ada

2. Object-Oriented programming Languages

Object-Based features+ Inheritance + Dynamic Binding

Examples: Simula, Smalltalk80, Objective C, C++, Eiffel etc.,

Introduction to C++

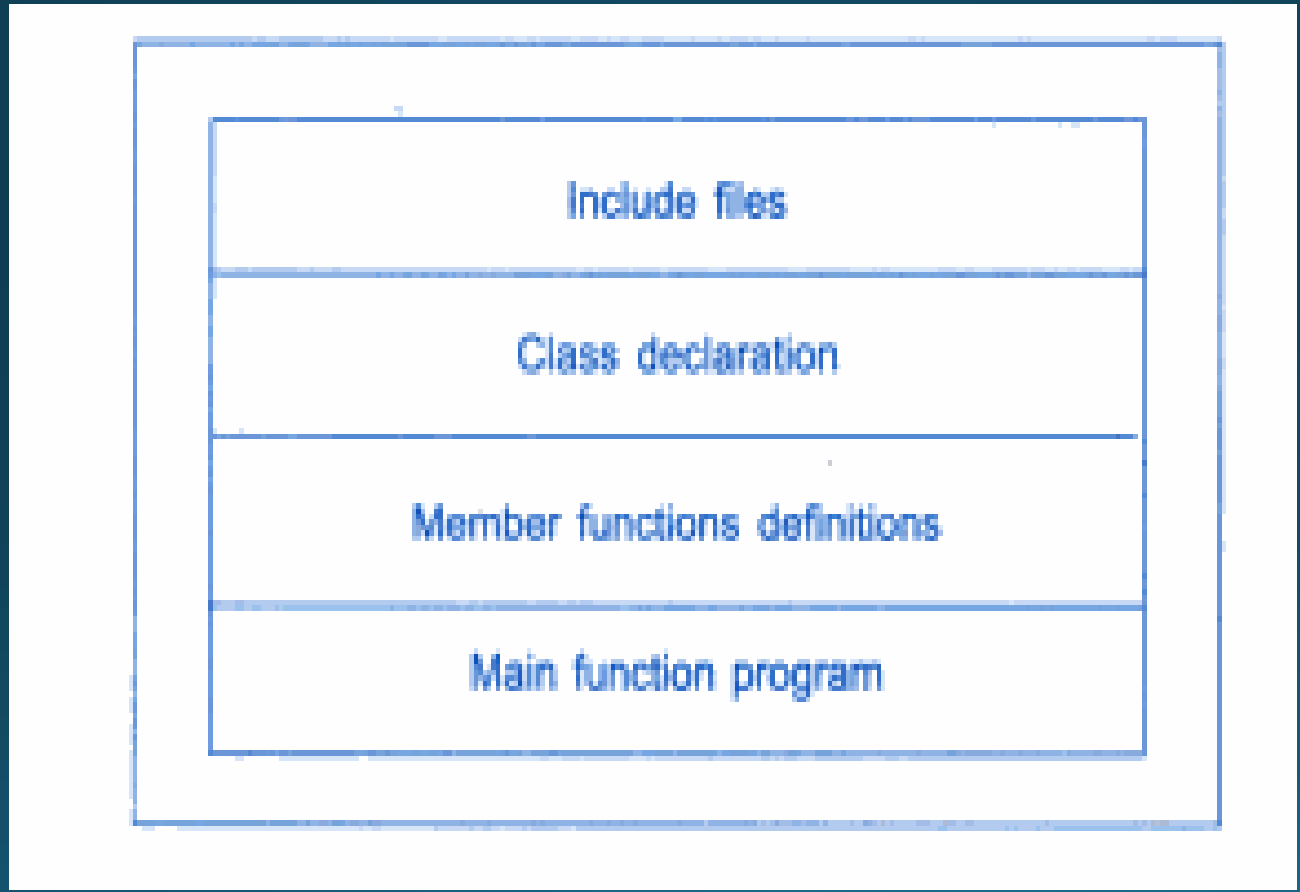
C++ is an Object-Oriented Language.

It was developed by Bjrane Stroustrup at AT&T Bell Labs, New Jersey, USA in the early 1980s.

It is a combination of Simula 67 and C.

It was initially called as “C with Classes” and was renamed as C++ in 1983.

Structure of a C++ program



Applications of C++

Machine-level programming - can be used in the development of editors, compliers, databases, communication system and any other real-life application

Hierarchical Structure - can be used to build special object-oriented libraries which can be used later by many programs.

Easily maintainable and expandable – when a new feature needs to be implemented, it is easy to add to the existing structure of an object.

Example C++ program without using class concept

eg.1:

```
#include<iostream.h>
void main()
{
cout<<“Welcome to C++ programming”;
}
```

output:

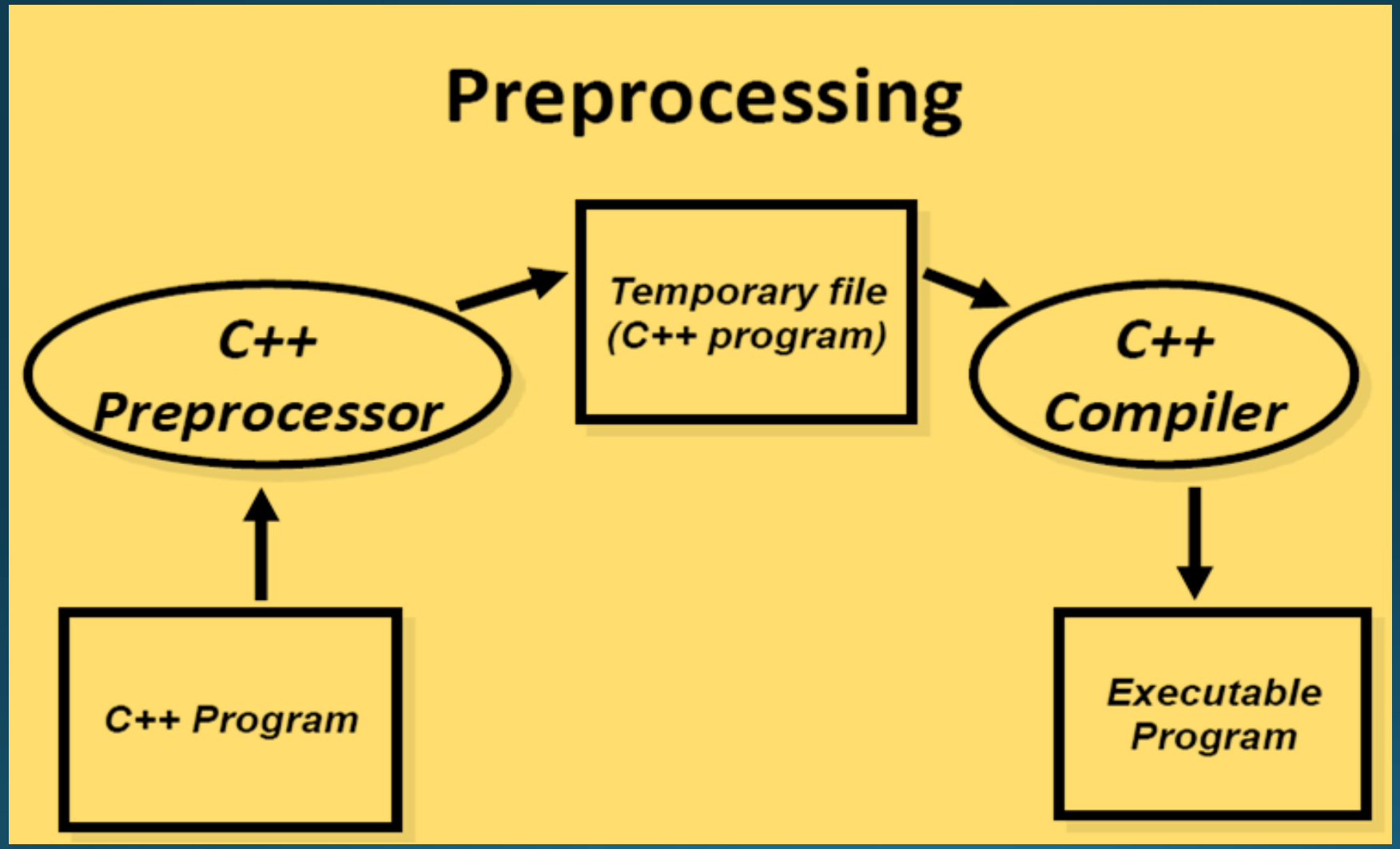
Welcome to C++ programming

eg.2:

```
// Program to add two integers      ← comment
#include<iostream.h>                ← allows access to I/O library
void main()                         ← Execution starts at main() functon
{
int a,b,c;
cout<<"\nProgram to add 2 integer values";
cout<<"\nEnter the values of integers a & b\n";
cin>>a>>b;
c=a+b;
cout<<"\nThe value of a is"<<a;
cout<<"\nThe value of b is"<<b;
cout<<"\nThe value of c is"<<c;
}                                   ← program termination as main() function closes
```

output:

```
Program to add 2 integer values
Enter the values of integers a & b
5 10
The value of a is 5
The value of b is 10
The value of c is 15
```


Preprocessing
C++ Preprocessor
Temporary file (C++ program)
C++ Compiler
C++ Program
Executable Program

Output Operation

The statement such as

cout<<"C++"; --- performs output operation.

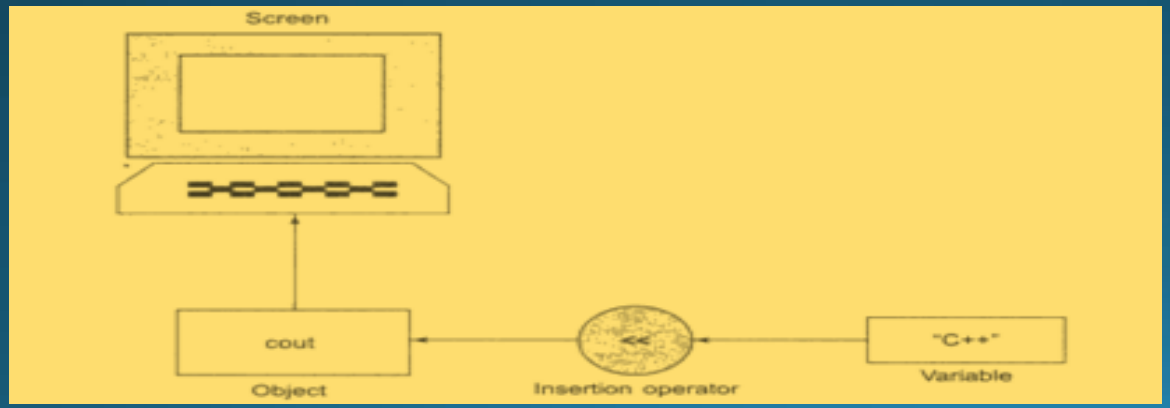
i.e., it prints the string constant given within double quotes in the output screen during execution.

cout is a predefined object that represents the standard output stream in C++.

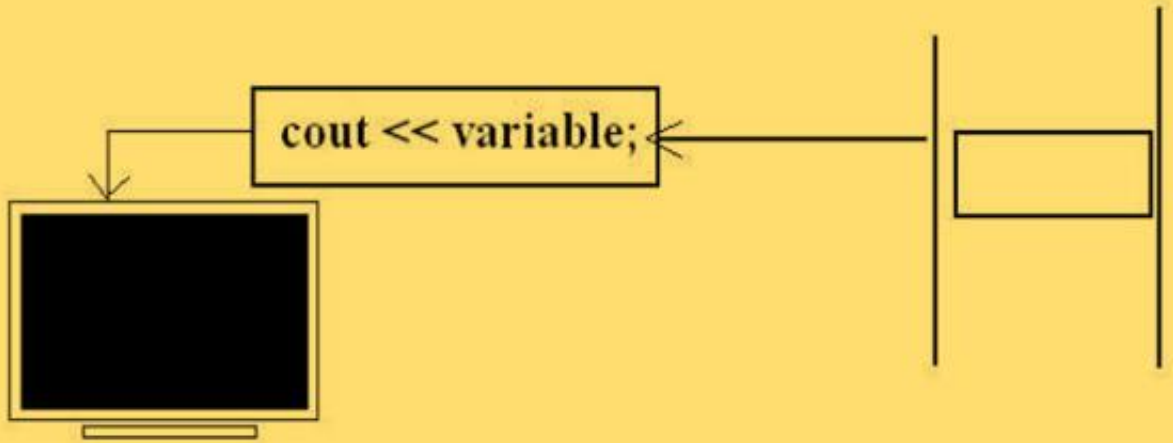
The standard output stream represents the screen.

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

It inserts or sends the contents of the variable on its right to the object on its left.



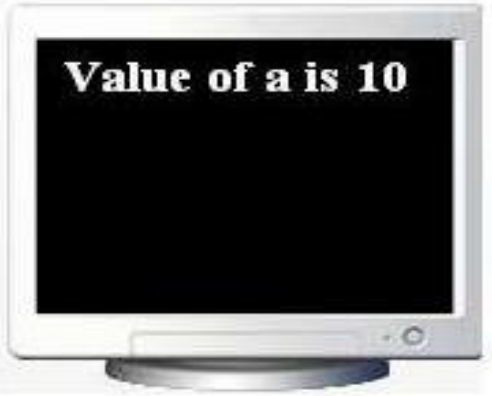
Output statement



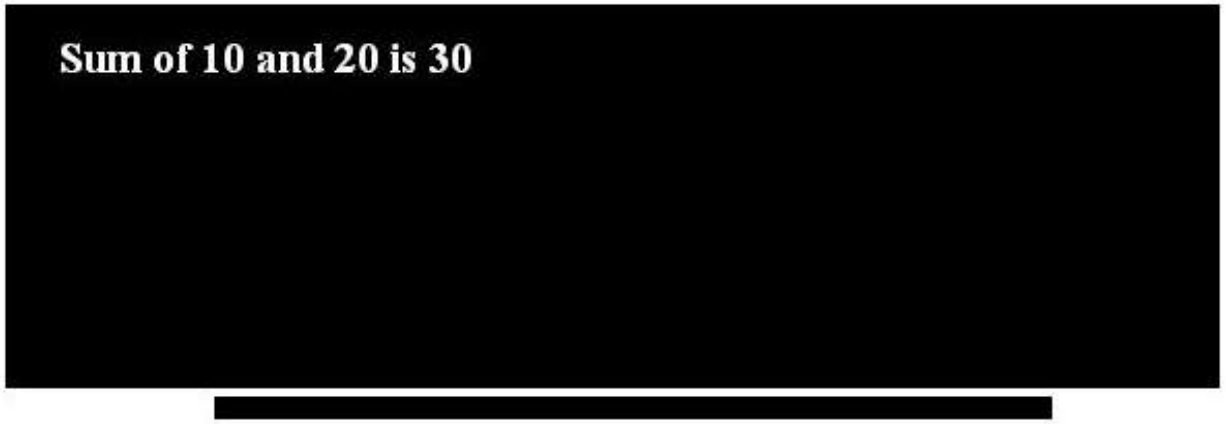
cout << "Welcome to c++";

```
a = 100;
cout << a;
```


```
a = 10;
cout << "Value of a is " << a;
```



```
a = 10;
b = 20;
c = a + b;
cout << "sum of " << a << " and " << b << "is" << c;
```




```
a = 10;
b = 20;
cout << "Sum of 2 numbers = " << a + b;
```

expression

Input Operation

The statement such as

cin>>a; --- performs input operation.

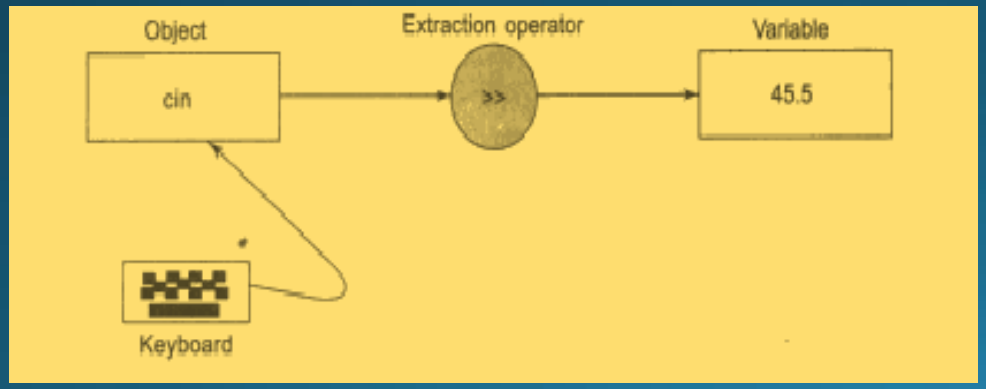
i.e., it cause the program to wait until it receives a value for the variable during execution time.

cin is a predefined object that represents the standard input stream in C++.

The standard input stream represents the keyboard.

The operator >> is called the extraction or get from operator.

It extracts or gets the value from the keyboard and assigns it to the variable on its right.



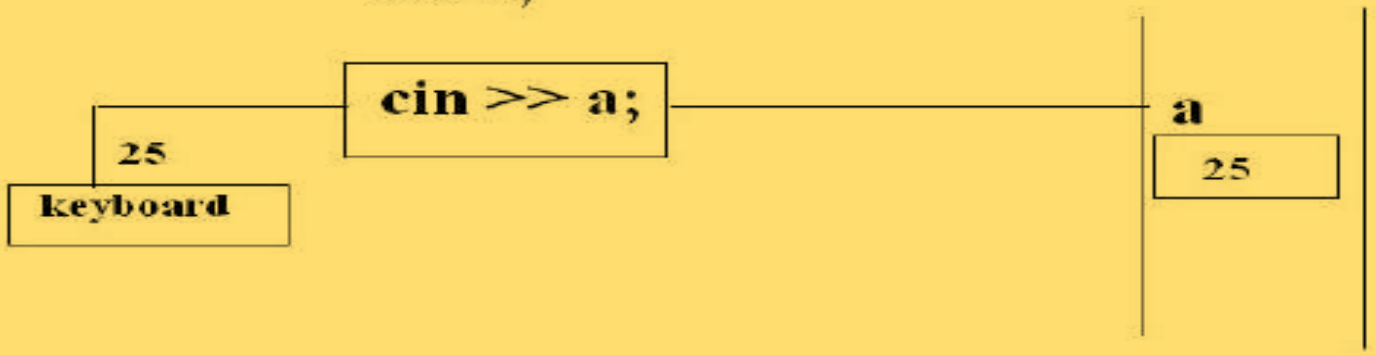
C++

Input and output of c++

```
cin >> variable ;
```

example:

```
int a;
cin >> a;
```



C++ Tokens:

Tokens are the smallest indivisible units in any programming language. C++ tokens are:

1. Reserved Words - Keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

RESERVED KEYWORDS

*Keywords are reserved words defined to perform a specific task.

*C++ has 48 keywords.

*All keywords are in lower case.

RESERVED KEYWORDS

delete	boolean	Break	Enum
case	volatile	Catch	Char
const	continue	Default	Do
else	asm	Extern	Union
float	for	Auto	Unsigned
if	inline	Register	Class
int	template	Long	Double
virtual	operator	Signed	goto
Protected	public	Sizeof	Return
Static	Struct	this	new
Friend	Throw	Typedef	private
try	Switch	while	short

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

*User-defined words

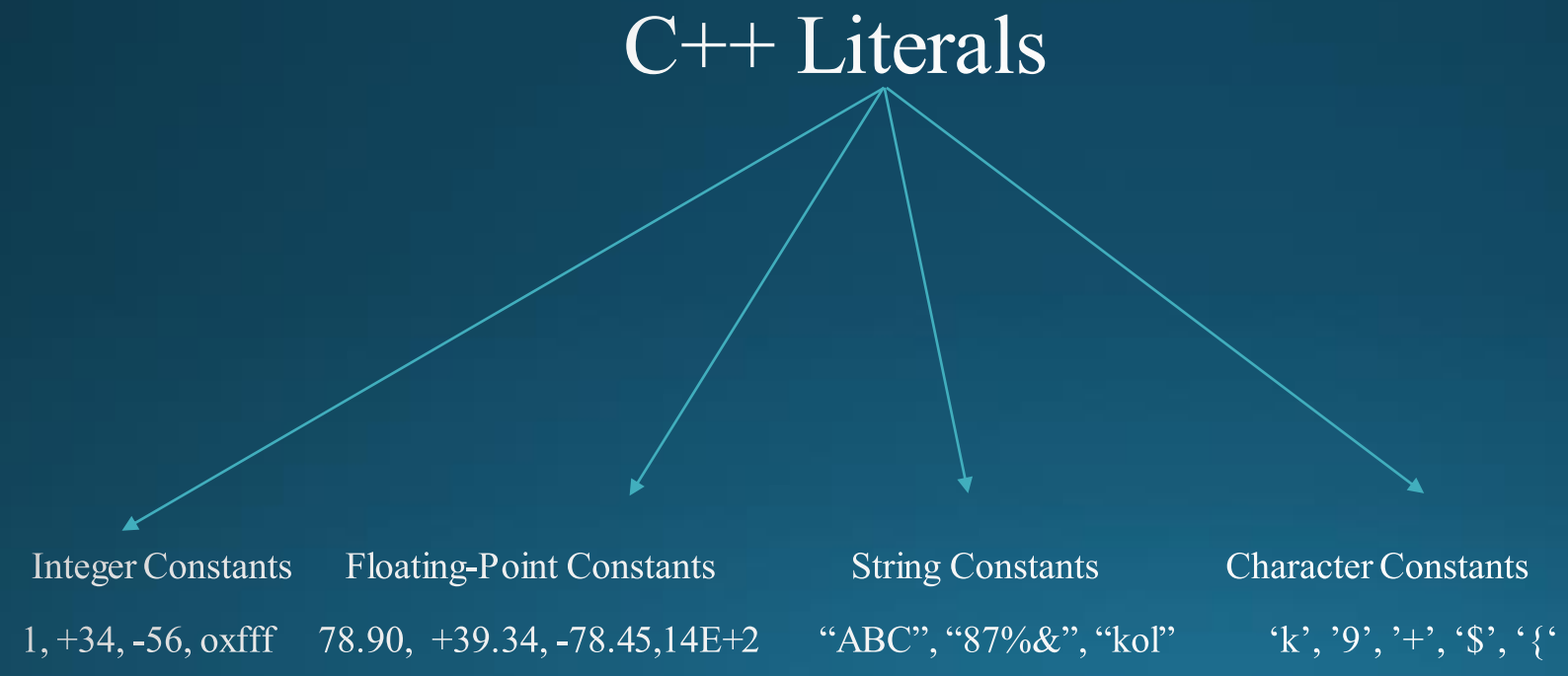
*Used to identify some program elements such as variable name, function name, class name, etc.,

*C++ rules for constructing identifiers

- alphabets, digits and underscore
- should not start with a digit
- case sensitive – ‘N’ is not same as ‘n’
- any length
- declared anywhere in the program
- keywords cannot be used as identifiers

LITERALS

Sequence of characters(identifiers) that represents constant values



LITERALS (Symbolic Constants)

1. using **const** keyword

```
const int k=234;
const float pi=3.143;
```

default type for const is int – i.e., const a=456; means a is an integer constant

2. using **enum** keyword – for declaring a set of integer constants

Egs. Without using tag names:

(I) enum{x,y,z}; means

```
const int x=0;
const int y=1;
const int z=2;
```

(II)enum(x=398,y=234,z=123); implies

```
const int x=398;
const int y=234;
const int z=123;
```

An enumerated data type is a user defined type which provides a way for attaching names to number.

The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and so on.

This facility provides an alternative means for creating symbolic constants.
Example:

```
enum shape{ circle,square,triangle}
enum colour{red,blue,green,yellow}
enum position  {off,on}
```

In C++, the tag names shape, colour, and position become new type names. That means we can declare new variables using the tag names.

Example:

```
Shape ellipse;                  //ellipse is of type shape
colour background ;             // back ground is of type colour
```

ANSI C defines the types of enums to be ints.

In C++,each enumerated data type retains its own separate type.

This means that C++ does not allow an int value to be automatically converted to an enum.

Example:

```
colour background =blue; //valid
colour background =7; //error in c++
colour background =(colour) 7;//ok
```

How ever an enumerated value can be used in place of an int value.

Example:

```
int c=red ;                    //valid, colour type promoted to int
```

By default, the enumerators are assigned integer values starting with 0 for the first enumerator, 1 for the second and so on. We can also write

```
enum color{red, blue=4,green=8};
```

in this case red=0, blue=4 & green=8

```
enum color{red=5,blue,green};
```

in this case red=5, blue=6 & green=7

3. using pre-processor directive **# define**

```
#define max 100
#define size 50
#define pi 3.14
```

#define is the pre-processor directive in C++

It should be used in the beginning of the program like #include

No need to give any datatype to the symbolic constant

No need for assignment operator =

; is not needed to terminate the pre-processor directive

OPERATORS

- Is a symbol that takes operands and operates on it – 1. unary 2. binary 3. ternary

1. Arithmetic operators → +, -, *, /, %
2. Relational operators → <, <=, >, >=, ==, !=
3. Logical operators → &&, ||, !
4. Assignment operator → =
5. Shortcut operators/
   Arithmetic Assignment operators → +=, *=, *=, /=, %=
6. Increment/Decrement operators → ++, --
7. Conditional operators → ? :
8. Bit-wise operators → <<, >>
9. Special Operators

Special Operators

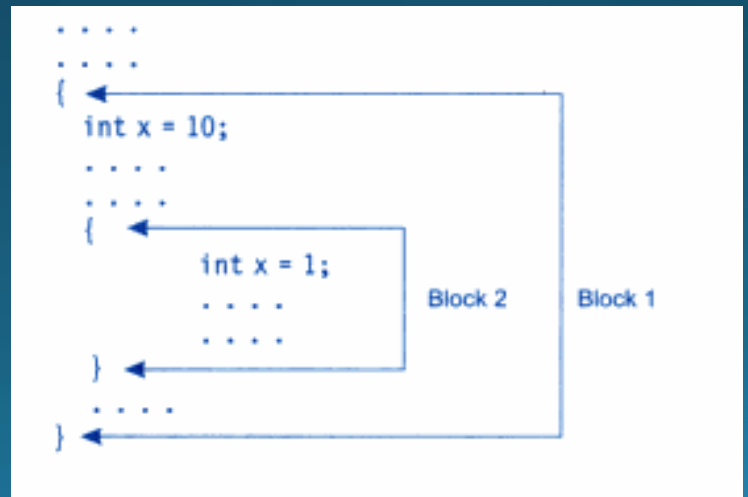
`::`	Scope resolution operator
`::*`	Pointer-to-member declarator
`->*`	Pointer-to-member operator
`.*`	Pointer-to-member operator
delete	Memory release operator
endl	Line feed operator
new	Memory allocation operator
setw	Field width operator

:: - Scope resolution operator

- two variable of same name can be used in different blocks.

```
.....
.....
{
   int x = 10;
   .....
   .....
}
.....
.....
{
   int x = 1;
   .....
   .....
}
```

- if one block is contained in another block and if there are variables of the same name in both the blocks??



In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator :: called the *scope resolution operator*. This can be used to uncover a hidden variable. It takes the following form:

```
:: variable-name
```

```
cout<<x<<"\n";          //here x=1
cout<<::x;              // here x=10
```

output:
1
10

C++ Implementation

```
class class_name
{
Attributes;//Properties      ----   data
Operations;//Behaviours     ----    functions
};

class stack
{ private:
int stackitems[10];
int tos;
public:
void init();
void push(int i);
int pop();
};
```

C++ Class Implementation

Example:

```
class student
{ private:
char st_name[30];
char st_id[10];
char branch[10];
char semester[10];
public:
void Enroll( );
void Displayinfo( );
void Result( );
void Performance( );
};
```

Example :-

```
# include<iostream.h>
class person
{ private:
char name[30];
int age;
public:
void getdata(void);
void display(void);
};
void person::getdata(void)
{
cout<<"enter name\n";
cin>>name;
cout<<"enter age";
cin>>age;          }
void person::display()
{
cout<<"\n name:"<<name;
cout<<"\n age:"<<age; }
```

```
int main( )
            {
   person a;
   a.getdata( );
  a.display( );
   return(0);
 }
```

Output:

```
enter name
arun
enter age
20
name:arun
age:20
```

Member Dereferencing Operator

C++ allows class to have any type of data and functions.

The class members can be accessed using pointer variables.

So C++ uses 3 pointer-to-member operators.

They are ::*, * and →*

Operator	Function
::*	To declare a pointer to a member of a class
*	To access a member using object name and a pointer to that member
->*	To access a member using a pointer to the object and a pointer to that member

Dynamic Memory Allocation(new) and De-Allocation(delete) Operators

An object can be created by using **new**, and destroyed by using **delete**, as and when required. A data object created inside a block with **new**, will remain in existence until it is explicitly destroyed by using **delete**. Thus, the lifetime of an object is directly under our control and is unrelated to the block structure of the program.

The **new** operator can be used to create objects of any type. It takes the following general form:

```
pointer-variable  =  new data-type;
```

```
int *p = new int;
float *q = new float;
```

Subsequently, the statements

```
*p = 25;
*q = 7.5;
```

assign 25 to the newly created **int** object and 7.5 to the **float** object.

```
pointer-variable  =  new data-type(value);
```

Here, value specifies the initial value. Examples:

```
int *p = new int(25);
float *q = new float(7.5);
```

When a data object is no longer needed, it is destroyed to release the memory space for reuse. The general form of its use is:

```
delete pointer-variable;
```

The *pointer-variable* is the pointer that points to a data object created with **new**. Examples:

```
delete p;
delete q;
```

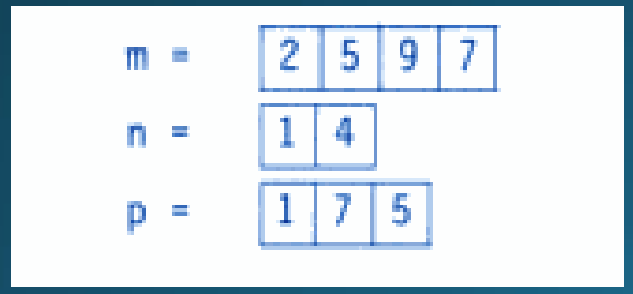
MANIPULATERS: - endl & setw

Manipulators are operator that are used to format the data display. The most commonly used manipulators are endl and setw.

The endl manipulator, when used in an output statement, causes a line feed to be insert.(just like \n). If m=2597, n=14 & p=175

Example:

```
cout<<"m="<<m<<endl;
cout<<"n="<<n<<endl;
cout<<"p="<<p<<endl;
```



setw - specify a common field width for the field to be printed.

- setw can be used to print numbers in right justified way

Example:

```
cout<<setw(5)<<"m="<<m<<endl;
cout<<setw(5)<<"n="<<n<<endl;
cout<<setw(5)<<"p="<<p<<endl;
```

```
m =  2597
n =    14
p =   175
```

Example: If sum = 345

```
cout << setw(5) << sum << endl;
```

The manipulator **setw(5)** specifies a field width 5 for printing the value of the variable sum. This value is right-justified within the field as shown below:

SEPARATORS

- Symbols used to indicate where groups of code are divided & arranged
- C++ separators:

()parentheses .. Methods, precedence in exp

{} braces .. Arrays init., block of codes, scopes

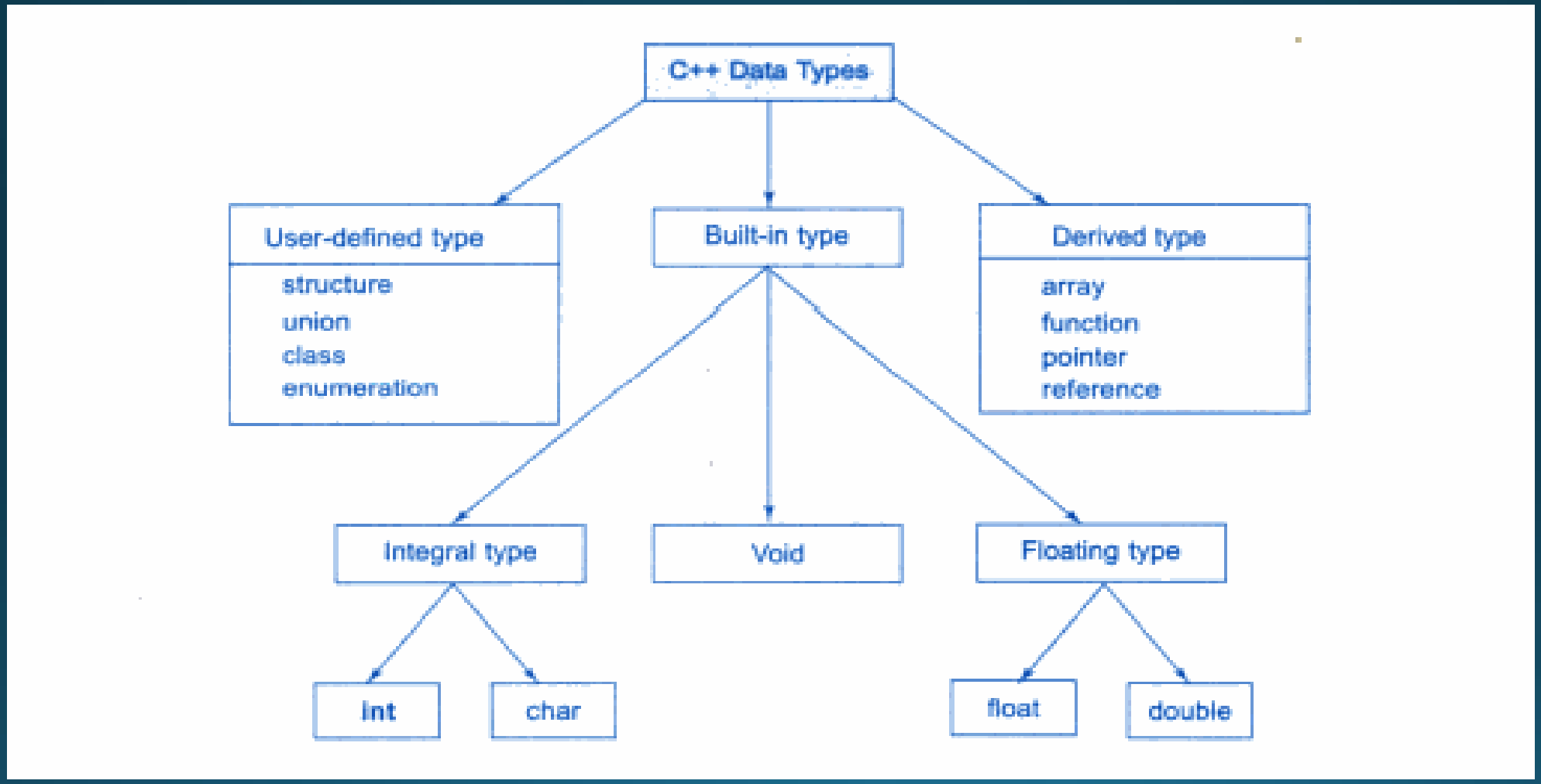
; semicolon

,comma.. Separate multiple identifiers, chain more than one stmt

. Period.. Data members, methods

[]Brackets.. Array referencing/dereferencing

Basic Data Types of C++:



Size & range of C++ basic data types:

Built-in Data types in C++ using sign & size qualifiers.

TYPE	BYTES	RANGE
char	1	-128 to127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647
signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	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

The type **void** normally used for:
1) To specify the return type of function when it is not returning any value.
2) To indicate an empty argument list to a function.

Example:
Void function(void);
Another interesting use of void is in the declaration of **generic pointer**
Example:

```
void *gp;                 //gp is generic pointer
int *ip;                  //ip is int pointer
gp=ip;                    // assigning int pointer to void type is valid in C++
```

Assigning any pointer type to a void pointer without using a cast is allowed in both C++ and ANSI C. In ANSI C we can also assign a void pointer to a non-void pointer without using a cast to non void pointer type. This is not allowed in C ++.
Example:

```
void *ptr1;
char *ptr2;
```

are valid statement in ANSI C but not in C++. We need to use a cast operator.

```
ptr2=(char*)ptr1;
```

User Defined Data Types:
Structures, Unions and Classes & enum

*User defined data types such as struct, and union are collections of heterogeneous data items.

*class is a user defined data type which can be used just like any other basic data type to declare a variable. The class variables are known as objects, which are the central focus of oops.

*Enumerated data type enum is already discussed in symbolic constant topic

Structure in C++ - keyword struct

A Structure is a user defined data type for handling a group of logically related heterogeneous data items.

i.e. It can be used to represent a set of attributes, such as : student_name , roll_no

A structure definition creates a format that may be used to declare structure variables.

Structure Declaration Syntax:

```
struct tag_name
{
data_type member1;
data_type member2;
… …
…… …..
};
```

1. The template terminated with a semicolon

2. While the entire declaration considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.

3. The tag name can be used to declare structure variables

Example:

```
struct book
{
char title[20];
char author[15];
int pages;
float price;
};
```

- The keyword struct declares a structure to hold the details of four fields, namely title, author, pages and price.

- These fields are called structure elements or members and each member may belong to different type of data.

- book is the name of the structure and also called as 'STRUCTURE TAG'.

Example:

```
struct book
{
char title[20];
char author[15];
int pages;
float price;
};
```

```
void main( )
{
    struct book b1;
    strcpy(b1.title,"Programming in c++");
    strcpy(b1.author,"Dr. E. Balagurusamy");
    b1.pages=680;
    b1.price=350.0;
    cout<<b1.title<<b1.author<<b1.pages<<b1.price;
 }
```

Union in C++ - keyword union

Union is a user defined data type similar to structures and therefore follow the same syntax as structures.

Major difference is in terms of Storage:

In structures each member has its own storage location.

In unions all members use the same location – common memory locations

Union may contain many members of different type but can handle only one member at a time.

Union Declaration Syntax

```
union union_name
{
data_type member1;
data_type mrmber2;
… … .. …..
} var1, var2, .. ;
```

Example

```
union book
{
char title[15];
char *author;
int pages;
float price;
} b1, b2, b3;
```

Derived Data Types

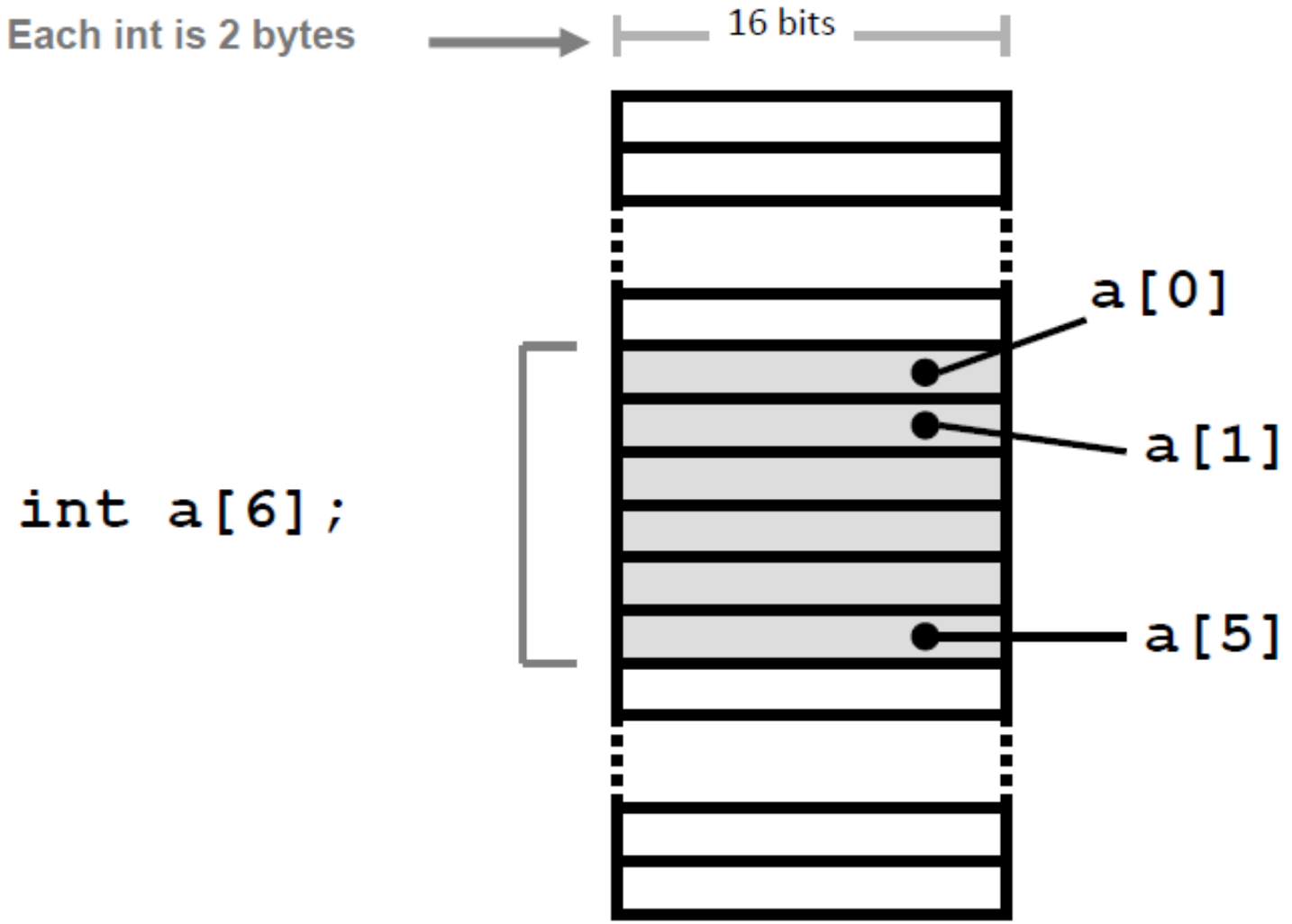
* **Arrays**
* **Pointers**
* **Functions**
* **Reference**

Array is a collection of similar data items that are stored in consecutive memory locations.

Syntax for declaring Arrays:

```
int a[6];
char name[30];
struct book b[20];
```

Memory and Arrays




```
int a[6];
```

Array Initialization

We can initialize an array :

```
int a[5] = { 1,8,3,6,12};

double d[2] = { 0.707, 0.707};

char s[] = { 'R', 'P', 'I' };
```

NOTE:
Need not have to specify a size when initializing, the compiler will count automatically.

Arrays

The application of arrays in C++ is similar to that in C. The only exception is the way character arrays are initialized. When initializing a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant. For instance,

```
char string[3] = "xyz";
```

is valid in ANSI C. It assumes that the programmer intends to leave out the null character \0 in the definition. But in C++, the size should be one larger than the number of characters in the string.

```
char string[4] = "xyz"; // O.K. for C++
```

An array printing function

Can pass an array as a parameter.
Need not have to say how big it is!

```
void print_array(int a[], int len)
{
  for (int i=0;i<len;i++)
     cout << "[" << i << "] = "
          << a[i] << endl;
}
```

2-D Array: `int A[3][4]`

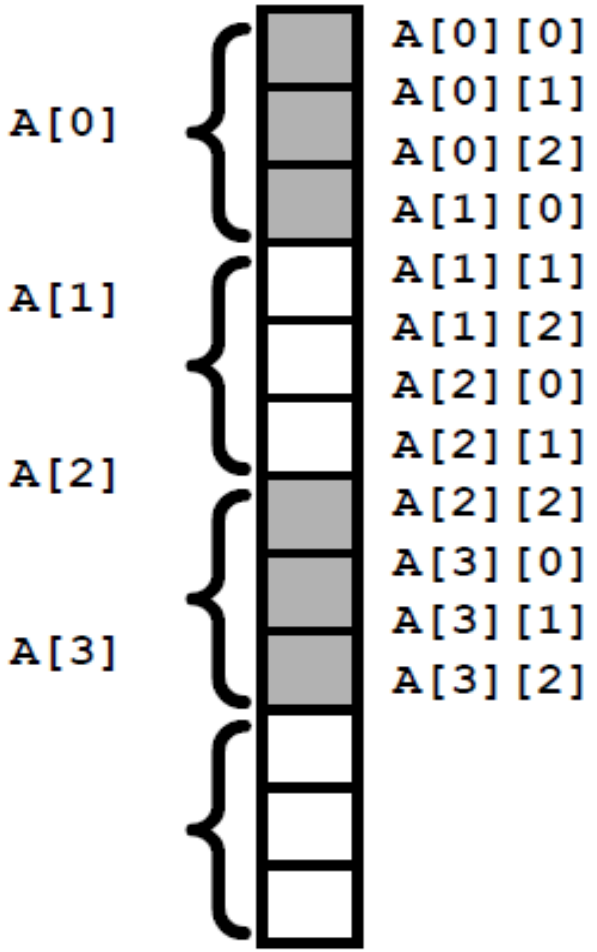
	Col 0	Col 1	Col 2	Col 3
Row 0	`A[0][0]`	`A[0][1]`	`A[0][2]`	`A[0][3]`
Row 1	`A[1][0]`	`A[1][1]`	`A[1][2]`	`A[1][3]`
Row 2	`A[2][0]`	`A[2][1]`	`A[2][2]`	`A[2][3]`

2-D Memory Organization

```
char A[4][3];
```

A is an array of size 4.

Each element of A is an array of 3 chars



2-D Array

Need not have to specify the size of the first dimension
But must include all other sizes!

```
double student_average( double g[][NumHW],
  int stu)
{
  double sum = 0.0;
  for (int i=0;i<NumHW;i++)
     sum += g[stu][i];

  return(sum/NumHW);
}
```

Pointers

*Pointers are address variable

*Variables that store the address of another variable

*Data type of the variable and its address must be the same

* *→ indirection operator (content of) is used to declare a pointer

* &→ address of operator is used to assign the address to pointer

Eg.

```
int a;
int *ptr;
ptr=&a;
```

Pointers

Pointers are declared and initialized as in C. Examples:

```
int *ip;          // int pointer
ip = &x;          // address of x assigned to ip
*ip = 10;         // 10 assigned to x through indirection
```

C++ adds the concept of constant pointer and pointer to a constant.

```
char * const ptr1 = "GOOD";    // constant pointer
```

We cannot modify the address that **ptr1** is initialized to.

```
int const * ptr2 = &m;  // pointer to a constant
```

ptr2 is declared as pointer to a constant. It can point to any variable of correct type, but the contents of what it points to cannot be changed.

We can also declare both the pointer and the variable as constants in the following way:

```
const char * const cp = "xyz";
```

This statement declares cp as a constant pointer to the string which has been declared a constant. In this case, neither the address assigned to the pointer cp nor the contents it points to can be changed.

Pointers are extensively used in C++ for memory management and achieving polymorphism.

DECLARATION OF VARIABLES:

In ANSI C all the variable which are to be used in programs must be declared at the beginning of the program.

But in C++ we can declare the variables any where in the program where it requires.

It makes the program easier to understand because the variables are declared in the context of their use.

Example:

```
main( )
{
float x;
float sum=0.0;
for(int i=1;i<5;i++)
  {
    cin>>x;
    sum=sum+x
   }
float average;
average=sum/4;
cout<<average;
return 0;
}
```

REFERENCE VARIABLES:

A references variable provides an alias name(alternative name) for a previously defined variable.

Syntax for declaring reference variable :

Datatype & reference_name=variable name;

Example:
float total=1500;
float &sum=total;

*Here **sum** is the **alternative name** for variables **total**
*Both the variables will refer to the same data object in the memory
*sum and total can be used interchangeably to represent the variable.
*A reference variable must be initialized at the time of declaration .
*The notation float & means reference to float.

Type casting operator

C++ permits explicit type conversion of variables or expressions using the type cast operator.

Traditional C casts are augmented in C++ by a function-call notation as a syntactic alternative. The following two versions are equivalent:

```
(type-name) expression  // C notation
type-name (expression)  // C++ notation
```

Examples:

```
average = sum/(float)i; // C notation
average = sum/float(i); // C++ notation
```

Expressions & their types

An expression is a combination of operators, constants and variables arranged as per the rules of the language. It may also include function calls which return values. An expression may consist of one or more operands, and zero or more operators to produce a value. Expressions may be of the following seven types:

- Constant expressions
- Integral expressions
- Float expressions
- Pointer expressions
- Relational expressions
- Logical expressions
- Bitwise expressions

An expression may also use combinations of the above expressions. Such expressions are known as *compound expressions*.

Constant Expressions

Constant Expressions consist of only constant values. Examples:

```
15
20 + 5 / 2.0
'x'
```

Integral Expressions

Integral Expressions are those which produce integer results after implementing all the automatic and explicit type conversions. Examples:

```
m
m * n - 5
m * 'x'
5 + int(2.0)
```

where **m** and **n** are integer variables.

Float Expressions

Float Expressions are those which, after all conversions, produce floating-point results. Examples:

```
x + y
x * y / 10
5 + float(10)
10.75
```

where **x** and **y** are floating-point variables.

Pointer Expressions

Pointer Expressions produce address values. Examples:

```
&m
ptr
ptr + 1
"xyz"
```

where **m** is a variable and **ptr** is a pointer.

Relational Expressions

Relational Expressions yield results of type **bool** which takes a value **true** or **false**. Examples:

```
x <= y
a+b == c+d
m+n > 100
```

When arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared. Relational expressions are also known as *Boolean expressions*.

Logical Expressions

Logical Expressions combine two or more relational expressions and produces **bool** type results. Examples:

```
a>b   &&   x==10
x==10   ||   y==5
```

Bitwise Expressions

Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits. Examples:

```
x << 3    // Shift three bit position to left
y >> 1    // Shift one bit position to right
```

Shift operators are often used for multiplication and division by powers of two.

Special Assignment Operator

Chained Assignment

```
x = (y = 10);
    or
x = y = 10;
```

First 10 is assigned to y and then to x.

A chained statement cannot be used to initialize variables at the time of declaration. For instance, the statement

```
float a = b = 12.34;            // wrong
```

is illegal. This may be written as

```
float a=12.34, b=12.34          // correct
```

Embedded Assignment

```
x = (y = 50) + 10;
```

(y = 50) is an assignment expression known as embedded assignment. Here, the value 50 is assigned to y and then the result 50+10 = 60 is assigned to x. This statement is identical to

```
y = 50;
x = y + 10;
```

Compound Assignment

Like C, C++ supports a *compound assignment operator* which is a combination of the assignment operator with a binary arithmetic operator. For example, the simple assignment statement

```
x = x + 10;
```

may be written as

```
x += 10;
```

The operator += is known as *compound assignment operator* or *short-hand assignment operator*. The general form of the compound assignment operator is:

```
variable1 op= variable2;
```

where *op* is a binary arithmetic operator. This means that

```
variable1 = variable1 op variable2;
```

Implicit Type Conversion

We can mix data types in expressions. For example,

```
m = 5+2.75;
```

is a valid statement. Wherever data types are mixed in an expression, C++ performs the conversions automatically. This process is known as *implicit* or *automatic conversion*.

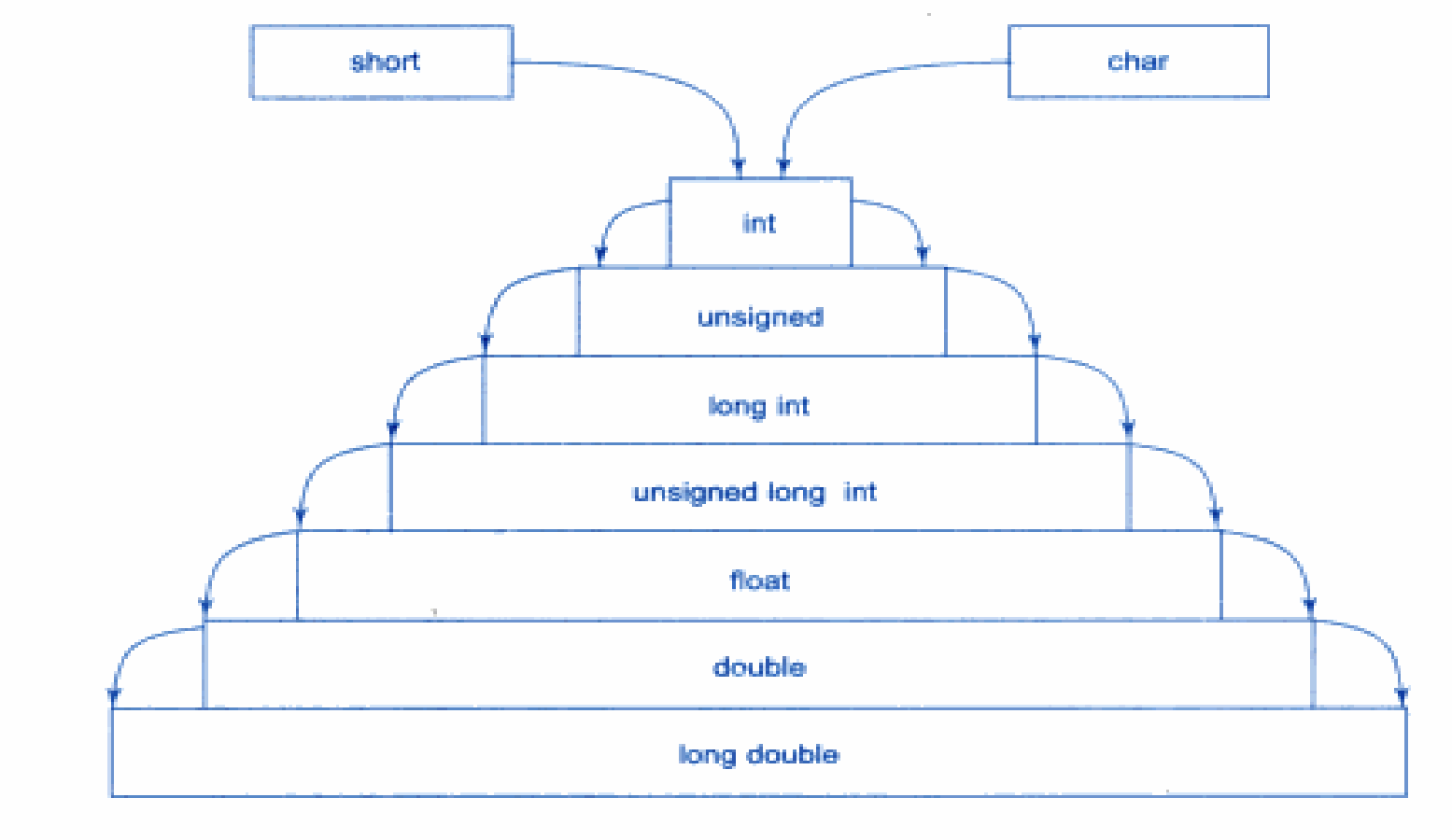
When the compiler encounters a mixed expression, it will convert one of them to match with the other, using the rule that the "smaller" type is converted to the "wider" type.
For eg.

* char to int

* int to float

* float to double

Waterfall model for implicit type conversion



Results of Mixed-mode Operations

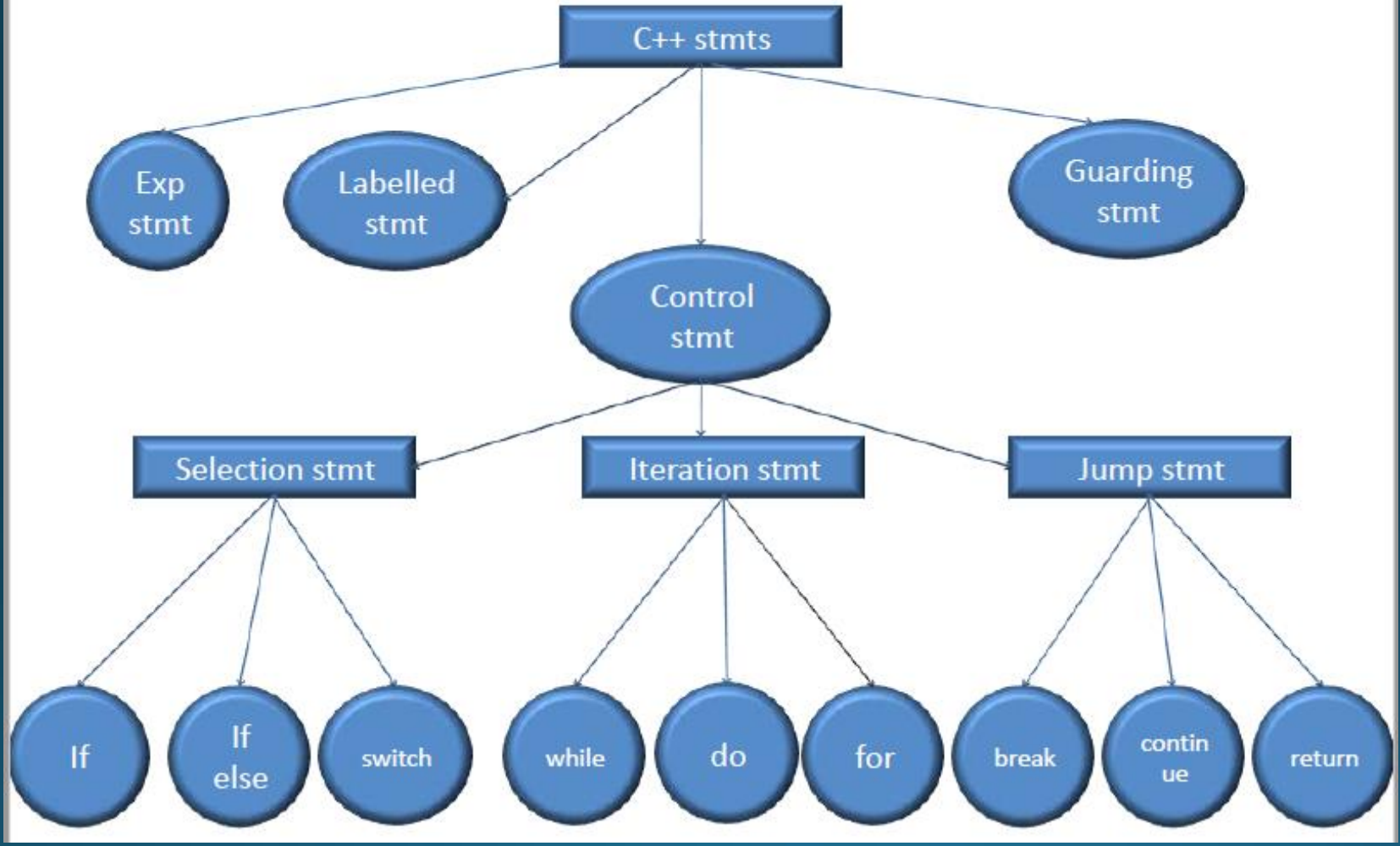
LHO \ RHO	*char*	*short*	*int*	*long*	*float*	*double*	*long double*
char	int	int	int	long	float	double	long double
short	int	int	int	long	float	double	long double
int	int	int	int	long	float	double	long double
long	long	long	long	long	float	double	long double
float	float	float	float	float	float	double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

RHO – Right-hand operand
LHO – Left-hand operand

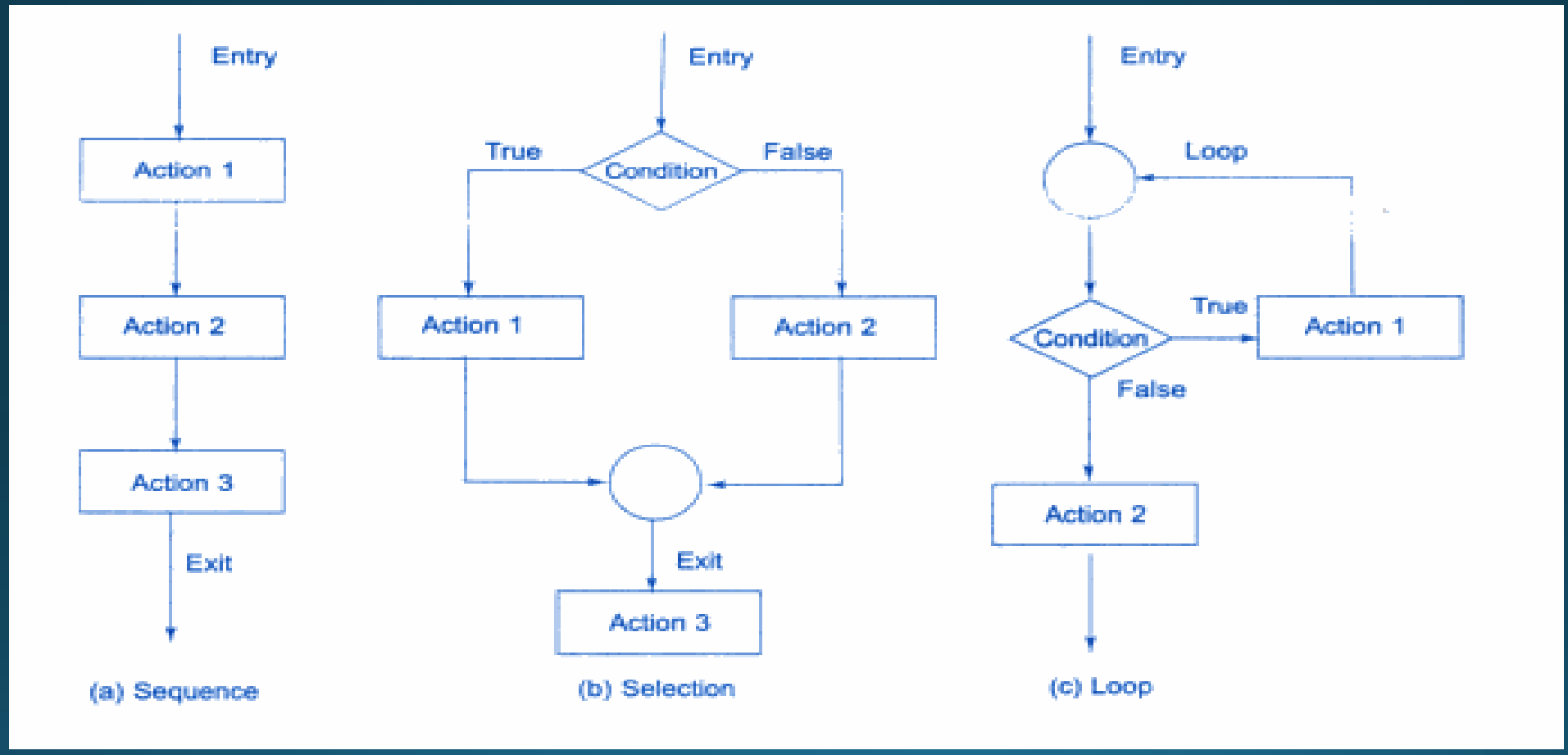
Operator Precedence & Associativity

Operator	*Associativity*
::	left to right
–> . ( ) [ ] postfix ++ postfix – –	left to right
prefix ++ prefix – – ~ ! unary + unary – unary * unary & (type) sizeof new delete	right to left
–> * *	left to right
* / %	left to right
+ –	left to right
<< >>	left to right
<< = >> =	left to right
= = !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	left to right
= * = / = % = + = =	right to left
<< = >> = & = ^= |=	left to right
, (comma)	

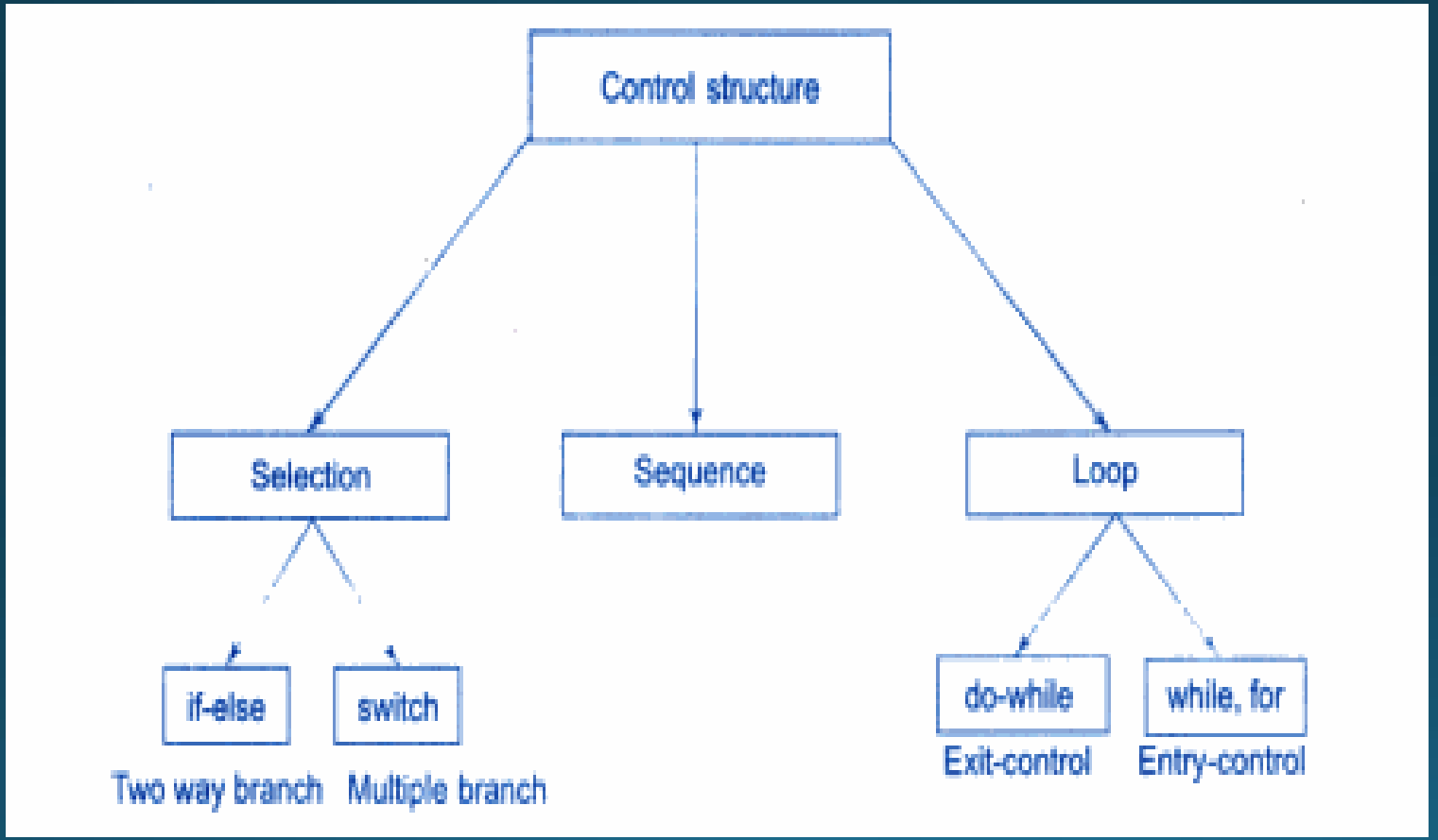
C++ STATEMENTS



Basic Flow of Control in C++ - Control Structures



Control Statements



Selection Statements

The if statement:

The if statement is implemented in three forms:

1. simple if statement
2. if..else statement
3. if..else if ladder

1. Simple if statement Syntax:

```
if (condition)
{
Statement block;
}
```

Eg.

```
#include<iostream.h>
void main()
{
int a,b;
cin>>a>>b;
if(a>b) { cout<<a<< " is greater than "<<b;}
if(b>a) { cout<<b<<" is greater than "<<a; }
}
```

Outout:

10 34

34 is greater than 10

2. if.. else statement

```
if (condition)
{Statement block1 …….;}
else
{Statement block2 …….;}
```

Eg.

```
#include<iostream.h>
void main()
{
int a,b;
cin>>a>>b;
if(a>b) { cout<<a<< " is greater than "<<b;}
else { cout<<b<<" is greater than "<<a; }
}
```

Outout:

10 34

34 is greater than 10

3. if..else if ladder

```
if (condition1)
   {Statement block1…….;}
else if(condition2)
         {Statement block2.......;}
       else if(condition3)
              ……….
            else
               {Statement blockN……;}
```

Eg.

```
#include<iostream.h>
      void main()
     {
       int a,b,c;
       cin>>a>>b>>c;
       if((a>b) && (a>c)) { cout<<a<<" is the largest number";}
       else if((b>a) && (b>c)) {cout<<b<<"is the largest number";}
             else if ((c>a) && (c>b)) { cout<<c<<"is the largest number";}
                   else { cout<<"two or more numbers are same";}
       }
```

Switch statement

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points;

```
switch(expr)
{
case 1:
    action1;
    break;
case 2:
    action2;
    break;
..
..
default:
    message
}
```

Eg. Program to build a simple calculator using switch..case statement

```
#include<iostream.h>
#include<iomanip.h>
int main()
{
char oper;     int n1,n2;
cout<<“Enter  an operator ( +, -, *, / ):”;
cin>>oper;
cout<<“Enter  two numbers:  “<<endl;
cin>>n1>>n2;
switch(oper)
{
    case ‘+’:
                cout<<n1<<  “ + ”<<n2<<“ = “<<n1+n2;
          break;
case ‘-’:
                cout<<n1<<  “ - ”<<n2<<“ = “<<n1-n2;
          break;
case ‘*’:
                cout<<n1<<  “ * ”<<n2<<“ = “<<n1*n2;
          break;
case ‘/’:
                cout<<n1<<  “ / ”<<n2<<“ = “<<n1/n2;
          break;
default:
                cout<<“ The operator is not correct”;
}
return(0);
}
```

Output 1:

Enter an operator (+, -, *, /):
+
Enter two numbers:
34 45
34 + 45 = 79

Output 2:

Enter an operator (+, -, *, /):
-
Enter two numbers:
34 45
34 - 45 = -11

Output 3:

Enter an operator (+, -, *, /):
*
Enter two numbers:
30 40
30 * 40 = 1200

Output 4:

Enter an operator (+, -, *, /):
/
Enter two numbers:
35 5
35 / 5 = 7

Output 5:

Enter an operator (+, -, *, /):
$
Enter two numbers:
5 8
The operator is not correct

Repetitive Statements-Iterative Statements

* repeating a set of instructions until a condition is satisfied (or) for ‘n’ number of times

* needs a control variable that controls the loop(repetition)

* control variable(loop variable) should be
  1. initialized
  2. Altered
  3. used for terminating the loop

* two types of loops
  1. entry-control loop eg.while
  2. exit-control loop eg. do..while

Repetitive Statements-Iterative Statements in C++

1. The while statement:

Syntax:

```
while(condition)
{
Statements;
}
```

2. The do-while statement:

Syntax:

```
do
{
Statements;
}while(condition);
```

3. The for loop:

Syntax:

```
for(initialization expression1;termination expression2;step expression3)
{
Statements;
Statements;
}
```

Example Program to add 10 integers using while statement

```
#include<iostream.h>
void main()
{
int n,i,sum;
i=0;
sum=0;
while(i<10)
{
cin>>n;
sum=sum+n;
i=i+1;
}
cout<<“The sum of the given numbers = “<<sum;
}
```

Example Program to add 10 integers using do..while statement

```
#include<iostream.h>
void main()
{
int n,i,sum;
i=0;
sum=0;
do
{
cin>>n;
sum=sum+n;
i=i+1;
} while(i<10);
cout<<“The sum of the given numbers = “<<sum;
}
```

Example Program to add 10 integers using for statement

```
#include<iostream.h>
void main()
{
int n,i,sum;
sum=0;
for(i=0;i<10;i++)
{
cin>>n;
sum=sum+n;
}
cout<<“The sum of the given numbers = “<<sum;
}
```

Unconditional Branch Statements – Jump Statements

1. break;
2. continue;
3. return
4. goto

1. **break;** //program to count the positive inputs(within 10 inputs),terminate on first 0 or negative no.

```
void main()
{
int x,count=0;
for(int i=0;i<10;i++)
{
cin>>x;
if(x<=0) break;
else count++;
}
cout<<count;
}
```

2. **continue;** //program to skip the negative or zero inputs(within 10 inputs)

```
void main()
{
int x,count=0;
for(int i=0;i<10;i++)
{
cin>>x;
if(x<=0) continue;
else count++;
}
cout<<count;
}
```

3. **return** – transfers control to the calling function.

4. goto

syntax goto label_name;

Eg.Program to find the sum of 10 positive integers

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
int k,i,sum=0;
for(i=0;i<10;i++)
{
getinput: cin>>k;
          if (k<=0) goto getinput;
          sum=sum+k;
}
cout<<"The sum of  the given  10 positive  intergers = "<<sum;
}
```

REFERENCES:

1.E. Balagurusamy, “Object Oriented Programming with C++”, Fourth edition, TMH, 2008.

2. LECTURE NOTES ON Object Oriented Programming Using C++ by Dr. Subasish Mohapatra, Department of Computer Science and Application College of Engineering and Technology, Bhubaneswar Biju Patnaik University of Technology, Odisha

3. K.R. Venugopal, Rajkumar, T. Ravishankar, “Mastering C++”, Tata McGraw-Hill Publishing Company Limited

4. Object Oriented Programming With C++ - PowerPoint Presentation by Alok Kumar

5. OOPs Programming Paradigm – PowerPoint Presentation by an Anonymous Author