


PROGRAMMING IN C++

COURSE CONTENT



UNIT – I

PRINCIPLES OF OBJECT ORIENTED PROGRAMMING

2

- 1.1 Procedure oriented Programming
- 1.2 Object oriented programming paradigm
- 1.3 Basic concepts of Object Oriented Programming
- 1.4 Advantages of Object Oriented Programming
- 1.5 Object Oriented Languages
- 1.6 Applications of Object Oriented Programming
- 1.7 C++ Concepts
- 1.8 Structure of C++ program
- 1.9 Applications of C++
- 1.10 Basic Data types in C++
- 1.11 User defined Data types
- 1.12 Derived Data types
- 1.13 Defining Constants
- 1.14 Declaration of variables and Dynamic initialization of variables
- 1.15 Reference variables
- 1.16 Operators in C++
- 1.17 Scope Resolution Operators
- 1.18 Member dereferencing Operators
- 1.19 Memory Management Operators and Manipulators
- 1.20 Type cast Operator

INTRODUCTION

- Object oriented programming is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming method by incorporating the best of structured programming features with several powerful new concepts.
- C++ is a superset of C.

PROCEDURE ORIENTED PROGRAMMING

- Conventional programming using high level languages such as COBOL, FORTRAN and C is known as procedure oriented programming (POP).
- In this approach, problem is viewed as a sequence of things to be done such as Reading, Calculating, Printing.
- Number of functions are written to complete the task.
- It consists a list of instructions and organize these instructions into group known as functions.

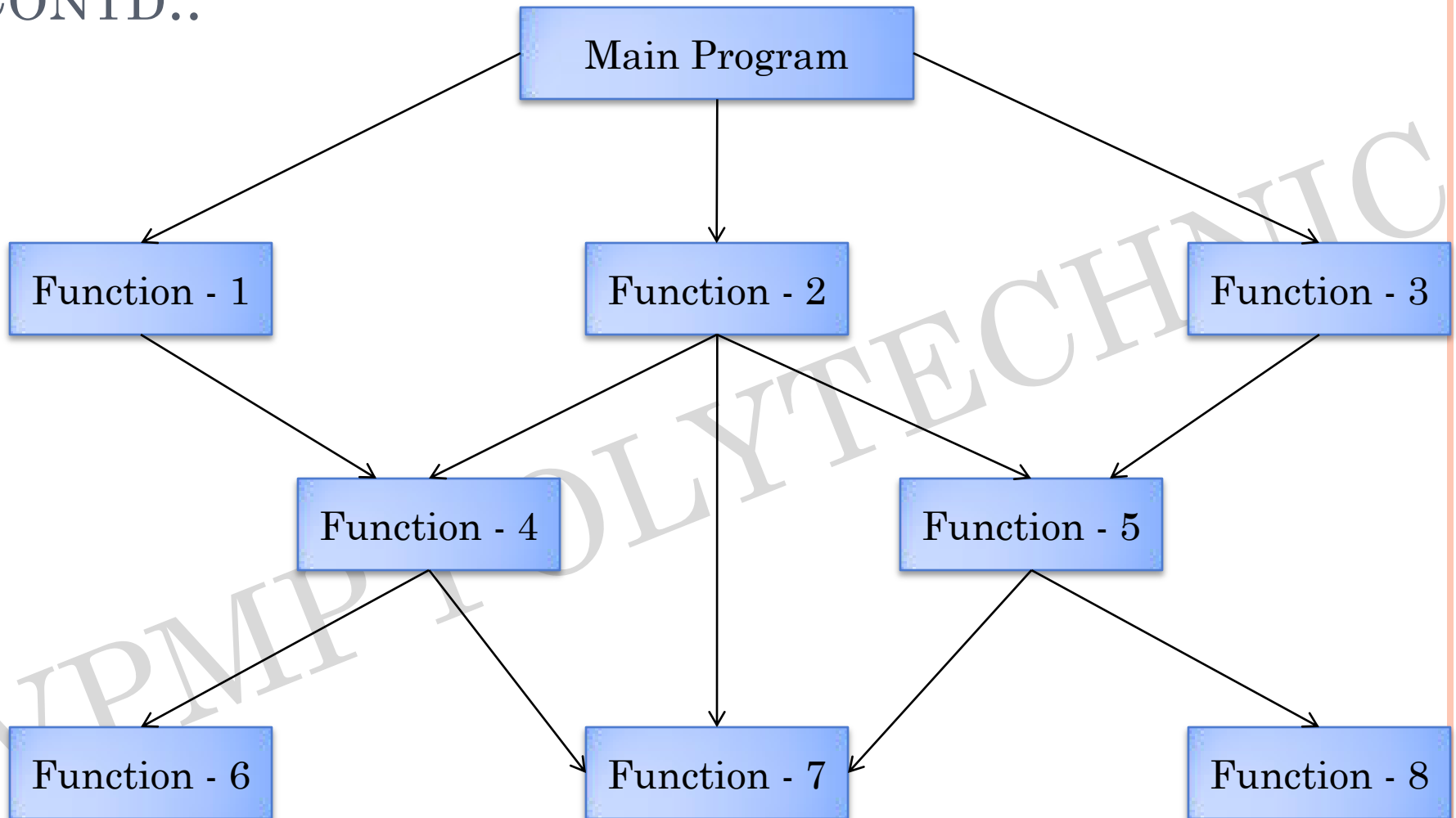
○ Disadvantage of POP:

- To develop a large program, it is very difficult to identify what data is used by which function.
- It does not model(solve) real world problem very well.

○ Characteristics Exhibited:

- Emphasis is on doing things (algorithms)
- Large programs divide into smaller programs known as functions.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- All the function share global data.
- Uses top-down approach in program.

CONTD..



Structure of the procedure oriented programs

OOP PARADIGM

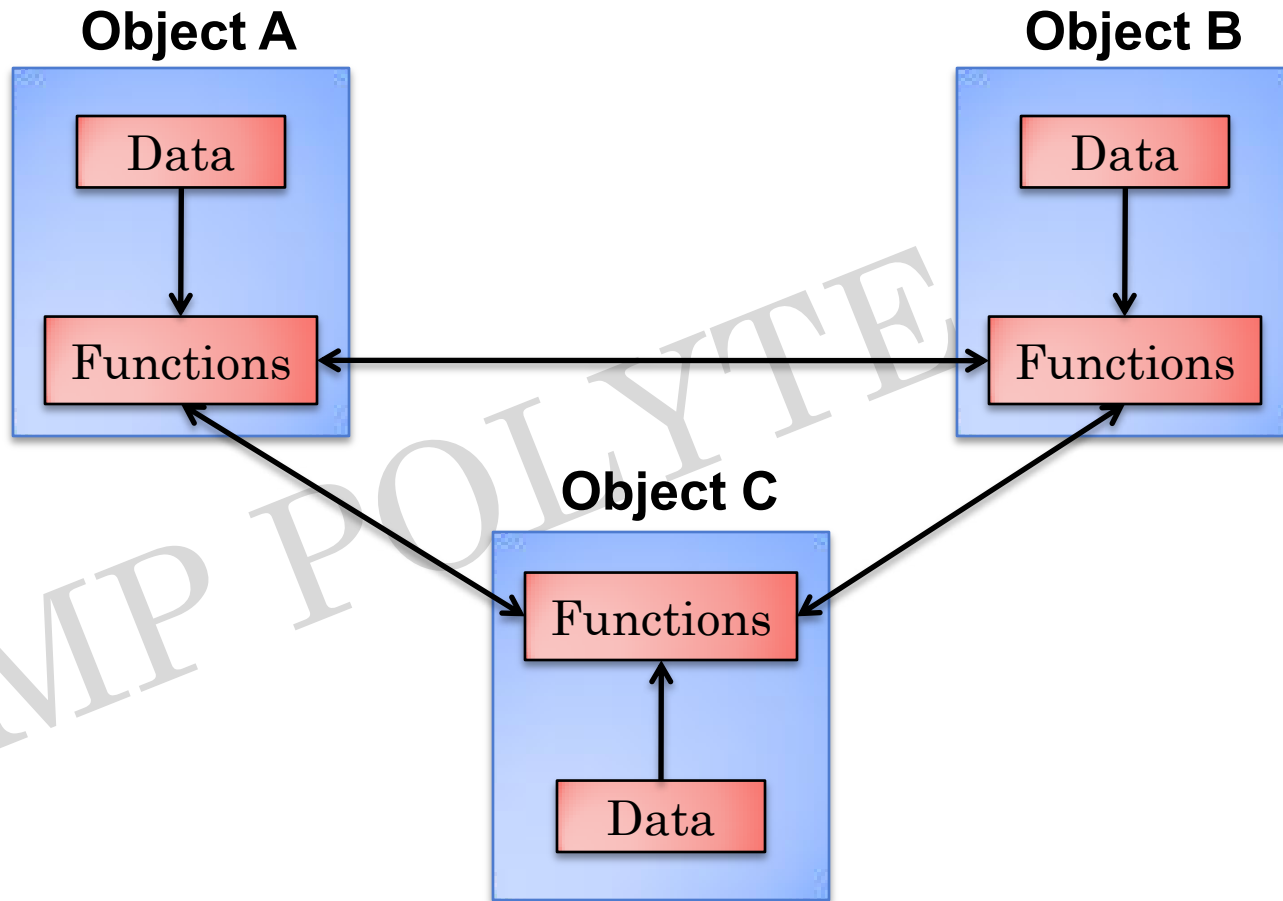
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside function.
- OOP allows decomposition of a problem into a number of entities called **objects** and then builds data and functions around these objects.

CONTD..

Features of OOP:

- Emphasis on data rather than procedure.
- Programs are divided into objects.
- Data structures are designed such that they characterize the objects .
- Function that operate on the data of an object are tied together in the data structure.
- Data is hidden and can't be accessed by external function.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.

CONTD..



Organization of data and function in OOP

BASIC CONCEPTS OF OOP

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

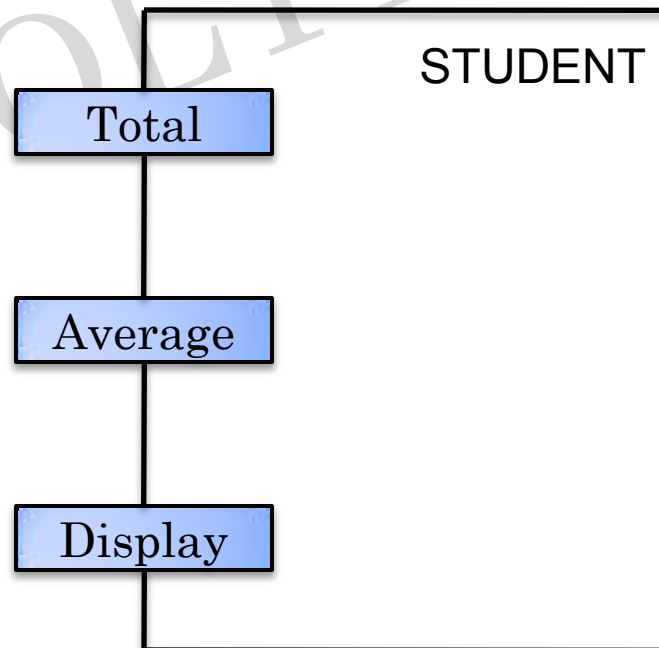
OBJECTS

- Objects are the basic run-time entities in the object oriented system.
- They may represent a data or items that the program has to handle.
- Each object contains the data and code to manipulate the data.
- Objects can interact without having to know details of each other's data or code.

CONTD..

- Representation of object by example:

Here, STUDENT is an object and Total, Average and Display are the data of that object.



CLASSES

- Objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.
- Each object is associated with the data of type class with which they are created.
- So, class is a collection of objects of similar type.
- For example, tiger, lion and panther are members of the class animal.
- Class are user-defined data type and behave like the built-in types of a programming language.
- Example:

animal tiger;

it will create an object tiger of the class animal.

DATA ABSTRACTION AND ENCAPSULATION

- The wrapping up of data and functions into a single unit (called class) is known as **encapsulation**.
- It is a most striking feature of class.
- 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 an interface between the object's data and the program.

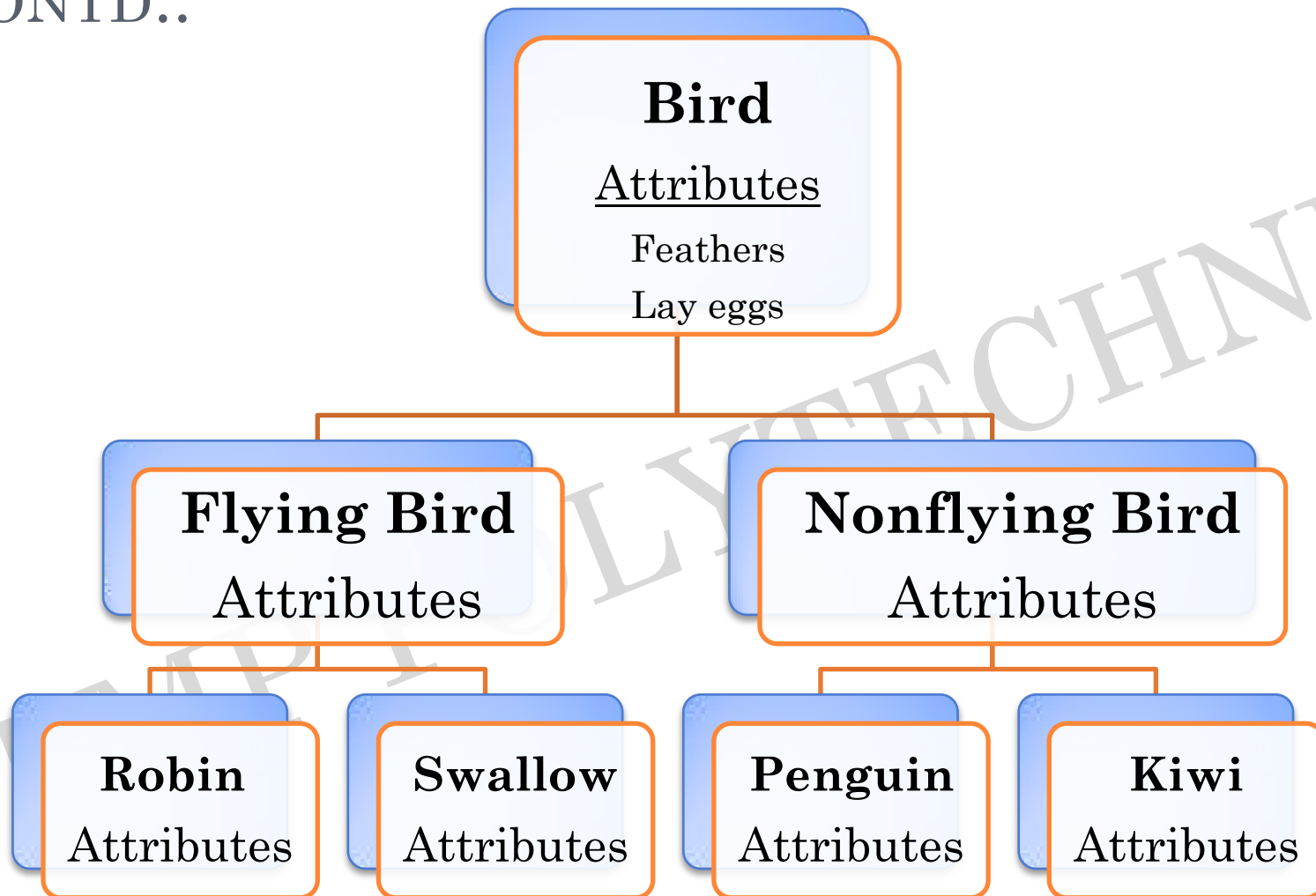
CONTD..

- **Abstraction** refers to the act of representing the essential features without including the background details.
- Classes use the concept of abstraction and are defined as a list of attributes and functions to operate on these attributes.
- They encapsulate all the essential properties of the objects that are to be created.
- The attributes are sometimes called data members because they hold information.
- The functions that operate on those data are sometimes called methods or member functions.

INHERITANCE

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- It supports the concept of hierarchical classification.
- It provides the idea of reusability.
- With the help of inheritance we can add the additional features to an existing class without modifying it.
- It is possible by deriving a new class from existing one.
- The new class will have the combined features of both the classes.

CONTD..



Example of Inheritance

POLYMORPHISM

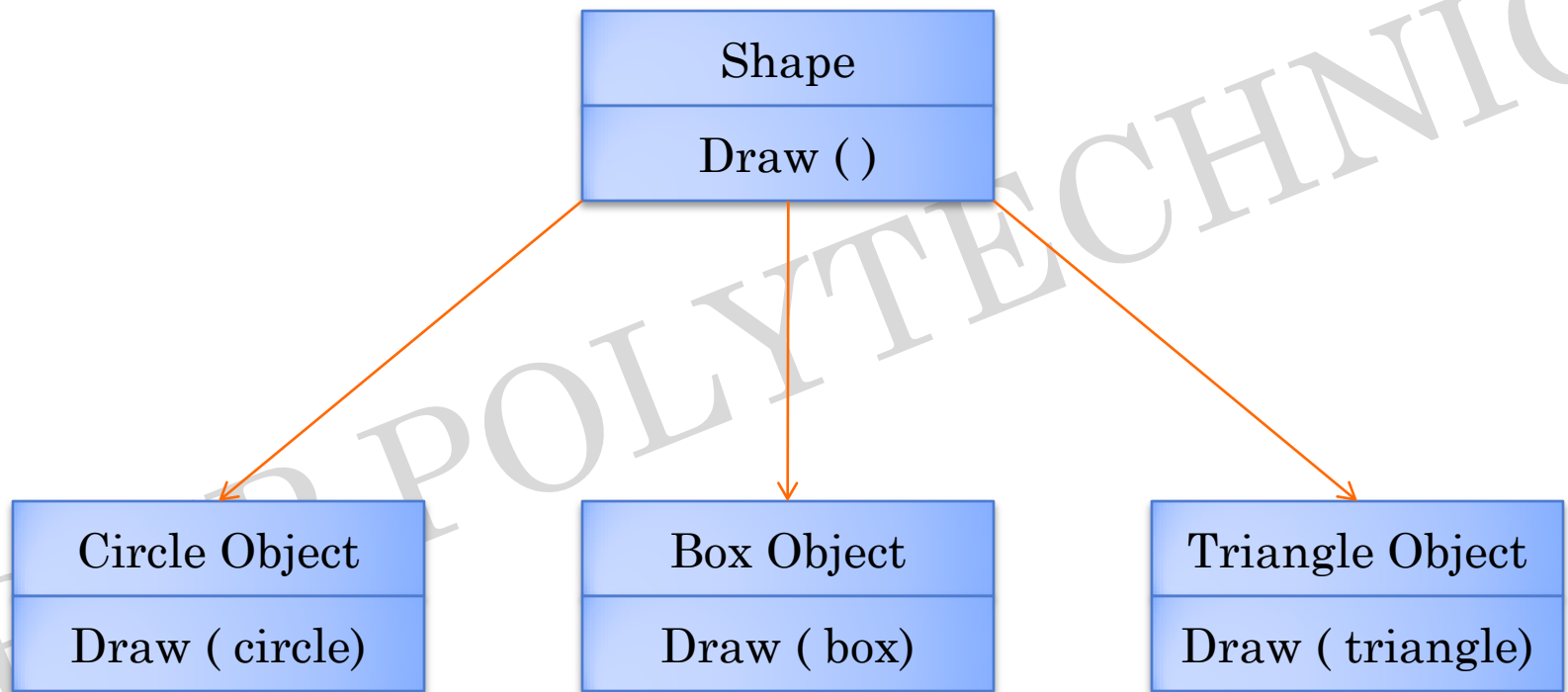
- Polymorphism means the ability to take more than one form.
- An operation may exhibit different behaviors in different instances.
- The behavior depends upon the types of data used in the operation.
- **Example:** The operation of addition.

for, two numbers the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

CONTD..

- The process of making an operator to exhibit different behavior in different instances is known as **operator overloading**.
- Same, using a single function name to perform different types of task is known as **function overloading**.
- Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.

CONTD..



Example of Polymorphism

DYNAMIC BINDING (LATE BINDING)

- Binding means the linking of procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.
- It is associated with polymorphism and inheritance.

MESSAGE PASSING

- An object oriented program consists of a set of objects that communicate with each other.
- **Steps for Message Passing:**
 1. Creating classes that define objects and their behavior.
 2. Creating objects from class definition,
 3. Establishing communication among objects.
- Objects communicate with one another by sending and receiving information.

CONTD..

- 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 the name of the object, the name of the function(message) and the information to be sent.
- **Example:**

employee. salary(name)

object message information

BENEFITS OF OOP

- Through inheritance, we can eliminate the redundant code and extend the use of existing classes.
- Save the development time and higher productivity.
- Through data hiding, the programmer can build secure program that cannot be modified by code in the other part of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is easy to partition the work in a project based on objects.

CONTD..

- The data centered design approach enables us to capture more details of a model in implementable form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

OBJECT ORIENTED LANGUAGES

- The languages should support several of the OOP concepts to claim that they are object-oriented.
- Depending upon the features they support, they can be classified into the two categories.
 1. Object-based programming language
 2. Object-oriented programming language.

CONTD..

○ Object-based programming language:

It supports encapsulation and object identity.

Major features required for object-based programming are :

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

They don't support inheritance and dynamic binding.

Ada is a example of object-based programming language.

CONTD..

- **Object-oriented programming language:**

object-oriented programming incorporates all of object-based programming features along with two additional features, inheritance and dynamic binding.

Language that support these features include C++, Smalltalk, Object Pascal and java.

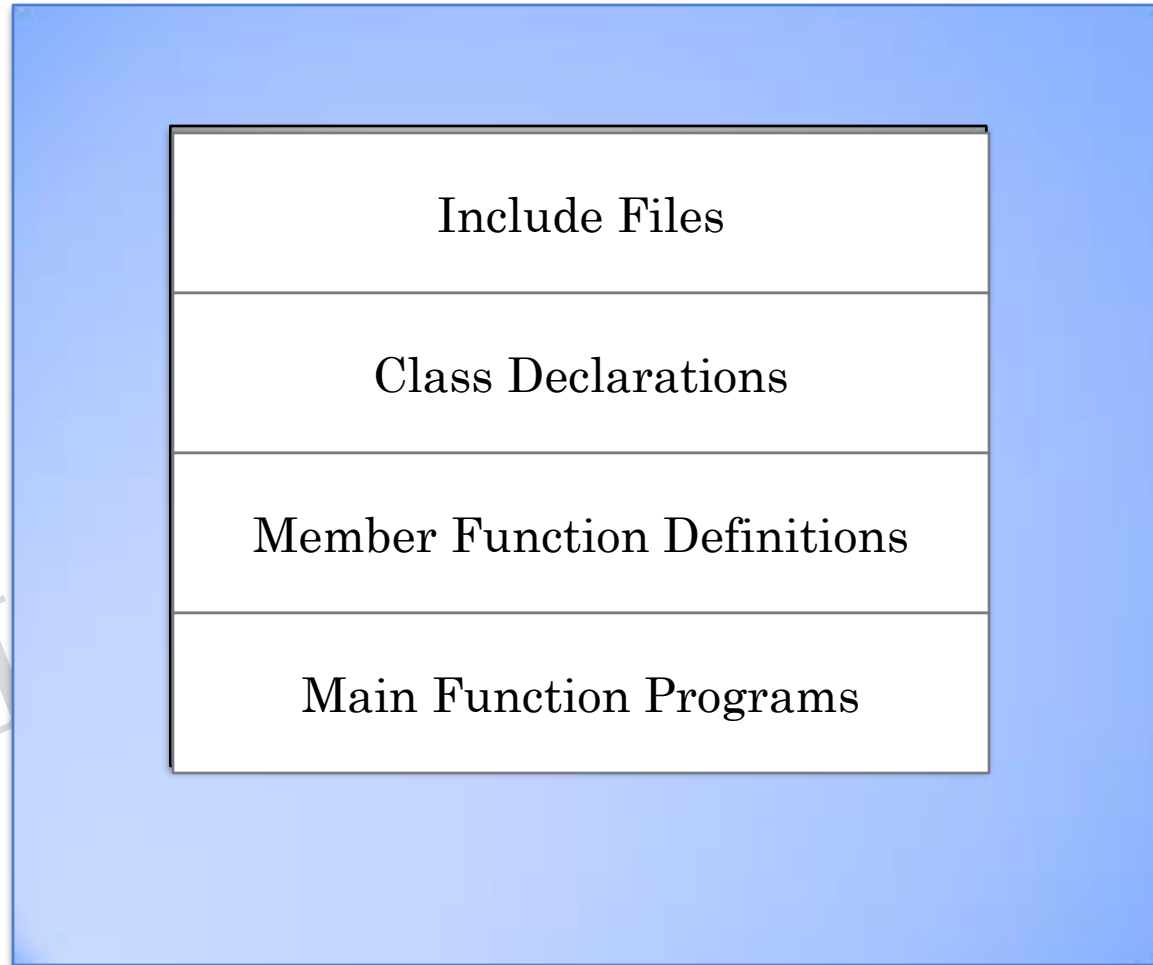
APPLICATION OF OOP

- Real-time systems.
- Simulation and modeling.
- Object-oriented databases.
- Hypertext, hypermedia and experttext.
- AI and expert systems.
- Neural networks and parallel programming.
- Decision support and office automation systems.
- CIM/CAM/CAD systems.

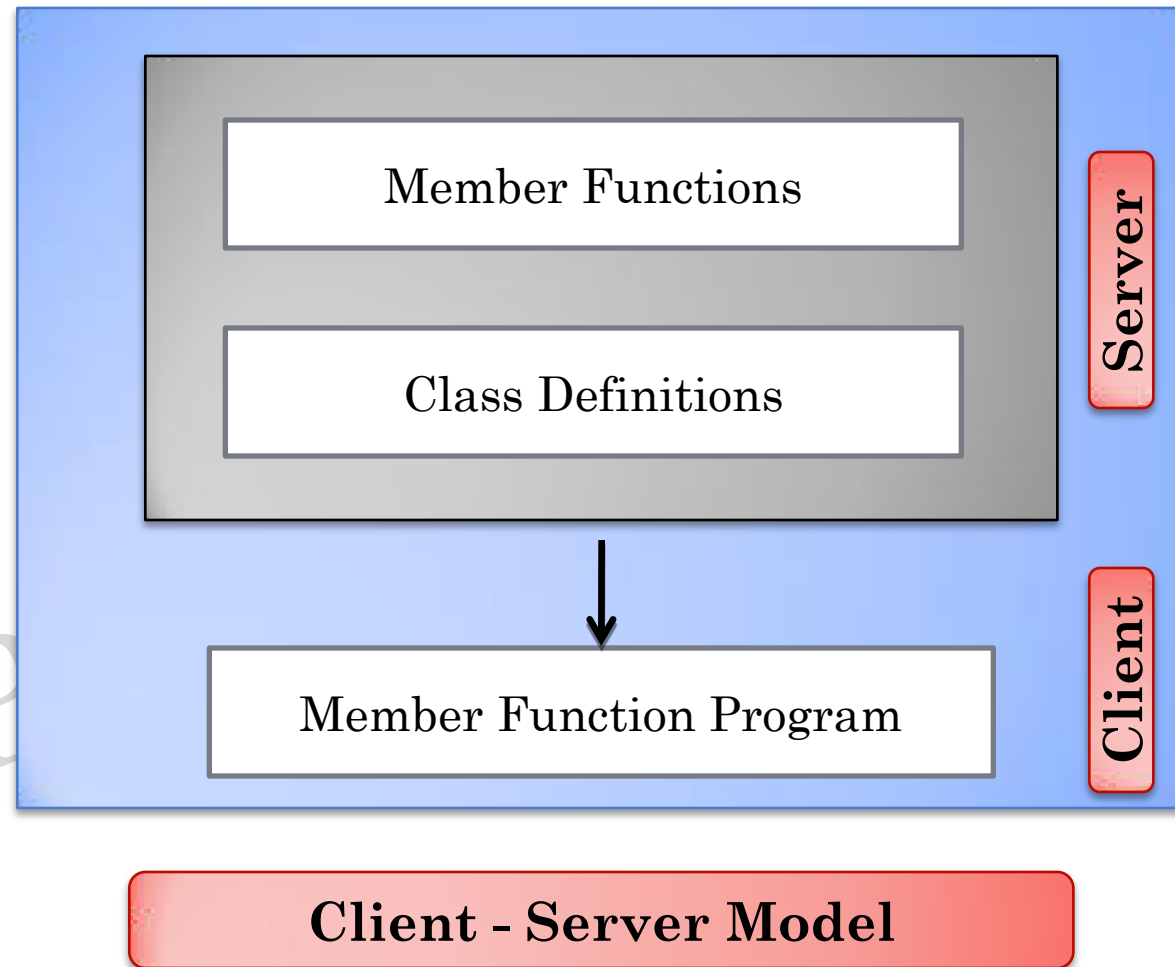
C++ CONCEPT

- C++ is an object-oriented programming language.
- It is developed by Bjarne Stroustrup at AT&T bell laboratories in 1980.
- C++ is a superset of C.
- Almost all C programs are also C++ programs.
- Added facilities in C++ are classes, inheritance, function overloading and operator overloading.

STRUCTURE OF C++ PROGRAM



CONTD..



CONTD..

- In the client-server model, the class definition including the member functions constitute the server provides services to the main program known as client.
- The client uses the server through the public interface of class.

TOKENS

- The smallest individual units in a program are known as tokens.
- In C++ there are five types of tokens.
 - Keywords
 - Identifiers
 - Constants
 - Strings
 - operators

KEYWORDS

- Keywords are explicitly reserved identifiers and cannot be used as names for the variables or other user-defined program elements.
- Example:
auto, class, return, int, friend, public, private, protected, wchar_t, bool, etc...

IDENTIFIERS

- **Identifiers** refer to the names of variables, functions, arrays, classes, etc created by programmer.
- **Rules for identifiers:**
 - Only alphabetic characters, digits and underscore are permitted.
 - Name cant start with digit.
 - Uppercase and lowercase both are distinct.
 - Keyword cant be used as a variable name.

CONTD..

- A major difference between C and C++ is a limit on the length of name.
- In C, there is only 32 characters are significant.
- While, in C++ there is no limit on the length of name.

CONSTANTS

- **Constants** refer to fixed values that do not change during the execution of program.

- **Example:**

- 123 // decimal integer
- 25.56 // floating point integer
- 037 // octal integer
- 0x2 // hexadecimal integer
- "C++" // string constant
- 'A' // character constant
- L'ab' // wide-character constant

CONTD..

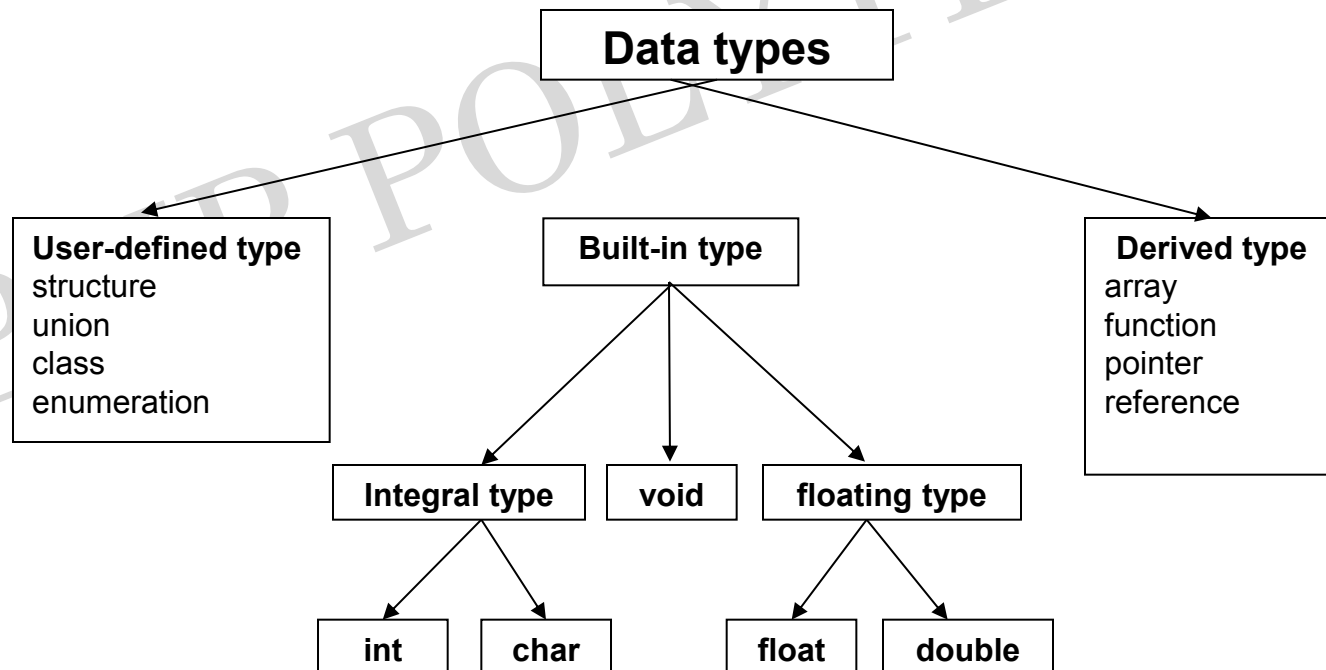
- The **wchar_t** type is a wide-character literal.
- It is intended for character sets that cannot fit a character into a single byte.
- Wide-character literal begin with the letter **L**.

APPLICATION OF C++

- For handling very large programs.
- For development of
 - Editors
 - Compilers
 - Databases
 - Communication systems
 - Complex real-life application systems
- For to build special object-oriented libraries which can be used later by the programmers.

BASIC DATA TYPES

- In C++, there is basically three types of data types are available. This is shown in the figure.
- The void data type does not return any



USER-DEFINED DATA TYPES

- **structure and union:**

- In C, structure and union both are the user-defined data types.
- Structure means collection of logically related elements of different data type.
- Union is same as structure.
- But, there is a difference between a structure and union.
- In structure, we can use all the elements at the same time. Where in union we can use the only one element at one time. After the use of one element we can use another element. So, in union all the elements share a common space. Where, in structure all the elements have its own space.

- **class:**

- A class is collection of object.
- In C++, structure is known as class. But it has some difference.
- In structure, we can declare only a variable. Where in class we can declare a variable (data member) as well as function (member function).

- **enumerated:**

- An enumerated data type provides a way for attaching names to numbers.
- The enum keyword is used to create a new data type.
- The enum keyword automatically enumerates a list of words by assigning them values 0, 1, 2 and so on.

❑ Syntax:

```
enum newdatatype
{
    Value1,
    Value2,
    Value3
};
```

• Example:

```
enum day
{
    Mon,
    Tues,
    Wed
};
```

- In the above example, Mon has value 0, Tues has value 1 and Wed has value 2.
- Value of Mon, Tues and Wed is not changed during the execution of the program.

DERIVED DATA TYPES

- **Arrays:**

- An array is fixed size sequenced collection of elements of same data type.
- Syntax: data-type array-name[size];
- Example: int a[10];
 - In above example, a is an array which stores 10 different elements of integer data type.
 - By default, the index of array is start from 0.

- **Functions:**

- Function is a group of instruction to perform a specific task.
- There is mainly two types of functions are available.
 - Library function
 - User defined function
- Advantage of the function is that we can reduce the code. And eliminate the redundant code of the program.

- **Pointers:**

- A pointer is a one type of variable which holds the address of another variable.
- Syntax of pointer declaration:

data-type *variable-name;

- Example:

```
int *p;      // pointer variable p is declared
p=&no;       // address of no is assigned to p
*p=25;       // 25 assigned to no through p
```

DEFINING CONSTANTS

- Symbolic constants can be created by two ways.
 - Using the qualifier constant
 - Using enum keyword.

- Using constant,

- Example:

Constant int size=10;

OR

Constant size = 10;

- Both the statements are same. It creates a constant
- variable size which has constant value 10. And it cannot be changed during the execution of the program.

DECLARATION OF VARIABLES

- There is some difference between C and C++ regarding to the declaration of the variable.
- In C, all the variables are declared at the beginning of the scope.
- Where in C++, we can declare a variable anywhere before they are used in the executable statements.
- Syntax of variable declaration: **data-type variable-name;**
- Example: **int i;**
- In C++, we can also declare a variable in the initialization section of the for loop statement which is not possible with C.

- Example:

```
void main()
{
    int sum=0;
    for (int i=1;i<10;i++)
    {
        sum = sum + i;
    }
    cout<< "Sum of 1 to 10 is :" << sum;
}
```

REFERENCE VARIABLES

- A reference variable is a one type of variable.
- It provides an alias or alternative name for a previously defined variable.
- Suppose, we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable.
- Syntax :
data-type & reference-name = variable-name;

- Example:

```
int a=15,b=25,total;
```

```
total = a + b;
```

```
int & sum = total;
```

```
cout<<total;           // it will print 40
```

```
cout<<sum;             // it will print 40
```

```
sum = 20;
```

```
cout<<total;           // it will print 20
```

```
cout<<sum;             // it will print 20
```

- A reference variable must be initialized at the time of declaration.

OPERATOR IN C++

- C++ has rich set of operators all C operators are valid in C++.
- In addition, C++ introduce some new operators. Some new operators:-
 - `::` scope resolution operator
 - `::*` pointer-to-member declarator
 - `->*` pointer-to-member operator
 - `.*` pointer-to-member operator
 - `delete` memory release operator
 - `new` memory allocation operator

SCOPE RESOLUTION OPERATOR

- Local variable: A variable which is declared inside the function or block is called as the local variable for that function and block.
- Global variable: A variable which is declared outside the function or block is called as the global variable for that function and block.
- In C, to access the global version of the variable inside the function and block that is not possible.
- In C++, the global version of variable is accessed inside the function and block using the scope resolution operator.

- Scope resolution operator is used to uncover a hidden variable.
- **Syntax:**

::variable-name

- **Example:**

```
int a = 10;
void main()
{
    int a = 15;
    cout<< " Value of a = " << a<<endl;
    cout<< " Value of a = " << ::a;
}
```

Output:

Value of a = 15

Value of a = 10

MEMBER DEREFERENCING OPERATORS

- The Member dereferencing operators use in the class, the following the member dereferencing operators use for different purpose.
- ::* To declare a pointer to the member class.
- .* To access a member using object and pointer of the class.
- ->* To access member using a pointer of the object and the pointer of the member.

MEMORY MANAGEMENT OPERATORS

- In C, `calloc()` and `malloc()` function are used to allocate memory dynamically at runtime.
- Similarly, it uses the function `free()` to free dynamically allocated memory.
- In C++, the unary operators `new` and `delete` are used to allocate and free a memory at run-time.
- An object can be created by using `new` and destroyed by using `delete` operator.
- The `new` operator can be used to create objects of any type.

- Syntax:
pointer-variable = new data-type;
- Example:
int *p = new int;
float *q = new float;
- Using the new operator we can also initialize the memory.
- Syntax:
Pointer-variable = new data-type(value);
- Example:
int *p= new int(25);
float *q = new float(10.5);
- The new operator can be used to create a memory space for any data type including user-defined types such as arrays, structures and classes.

- Syntax:

`pointer-variable = new data-type[size];`

where, size specifies the number of elements in the array.

- Example:

`int *p = new int [10];`

which create a memory space for an array of 10 integers.

- When a data object is no longer needed, it is destroyed to release the memory space for reuse.

- Syntax:

`delete pointer-variable;`

- Example:

`delete p;`

`delete q;`

- To free a dynamically allocated array we can use the delete operator.

- Syntax:

`delete [size] pointer-variable;`

The size specifies the number of elements in the array to be freed.

- Example:

`delete [] p;`

MANIPULATORS

- Manipulators are operators that are used to format the output.
- Most commonly used manipulators are endl and setw.
- The endl manipulator is used in the output statement when we want to insert a linefeed.
- It has the same effect as using the newline character “\n”.

Example:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a=15,b=420,c=3542;
    cout<< “a =”<<a<<endl;
    cout<< “b =”<<b<<endl;
    cout<< “c =”<<c<<endl;
    getch();
}
```

Output:

```
a = 15
b = 420
c = 3542
```

- ❑ The setw manipulator is used to format the output.
- ❑ When we use the setw manipulator the output is formatted in right justified.
- ❑ Before using the setw manipulator we have to include the header file iomanip.
- ❑ Syntax:

setw(width)

Where, width specifies the field width.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a=15, b=420, c=3542;
    cout<< "a ="<<setw(4)<<a<<endl;
    cout<< "b ="<<setw(4)<<b<<endl;
    cout<< "c ="<<setw(4)<<c<<endl;
    getch();
}
```

Output:

```
a =  15
b = 420
c =3542
```

TYPE CAST OPERATOR

- Casts can be used to convert objects of any scalar type to or from any other scalar type.
- Explicit type casts are constrained by the same rules that determine the effects of implicit conversions.
- Example

```
float x = 3.1;
```

```
int i;
```

```
i = (int)x;
```

```
//the value of I is now 3
```

Types cast operators

- Casts can be used to convert objects of any scalar type to or from any other scalar type.
- Explicit type casts are constrained by the same rules that determine the effects of implicit conversions.
- Example

```
float x = 3.1;
```

```
int i;
```

```
i = (int)x;
```

```
//the value of I is now 3
```

