SYLLABUS

UNIT-I -> C++ Basic : overview - C++ functions : simple function - call by value and call by reference inline functions, macro versus inline function. overloading of function default arguments, friend functions, virutal functions

UNIT-II -> Pointer and I/o and file management - Pointers in C++, Pointers and objects, this pointer, Virutal and pure virutal functions. Streams - Concepts of stream, cin and cout objects, C++ Stream clases, unformated input & output, manipulated file stream, C++ file stream class, file management functions

UNIT-III -> Java basic; class and objects - string - Inheritance - polymorphyism and interfaces - regular expressions - exspection handling

UNIT-IV - Java GUI, file stream and concurrency - GUI development using swing - I/o streams and object serialization - Generic collections - Concurrency -

Thread states and life cycles - Thread Synchronization Java networking

UNIT-V -> Client side essentials - Java script objects and Function - JQuery accessing dom elements using Javascript and JQuary objects - Javascript event handling - XML DOM - AJAX enabled riched internet applications with XML and JSON - Dynamic asics and manipulation of web pages using Javascript and JQUARY - Web speach API - Speech synsthesis markup language

OOPS

C++ overview

* C++ is general purpose, middle level language to perform high level process

* C++ was developed by B.Jarne stroustrup in the year 1983

* Extension of c programming language is c++

Why c++?

* High level performance software
* client server architecture
* system | Desktop application
* Database software.

C++ program structure

```
#include<iostream.h>
using namespace std
void main()
{
 cout<<"Hello C++";
 getch();
}
```

output

Hello C++

user input and output

cin >> => get input from the user

>> => Extraction operator

cout << => Display the output in the screen.

<< => insertion operator

Program

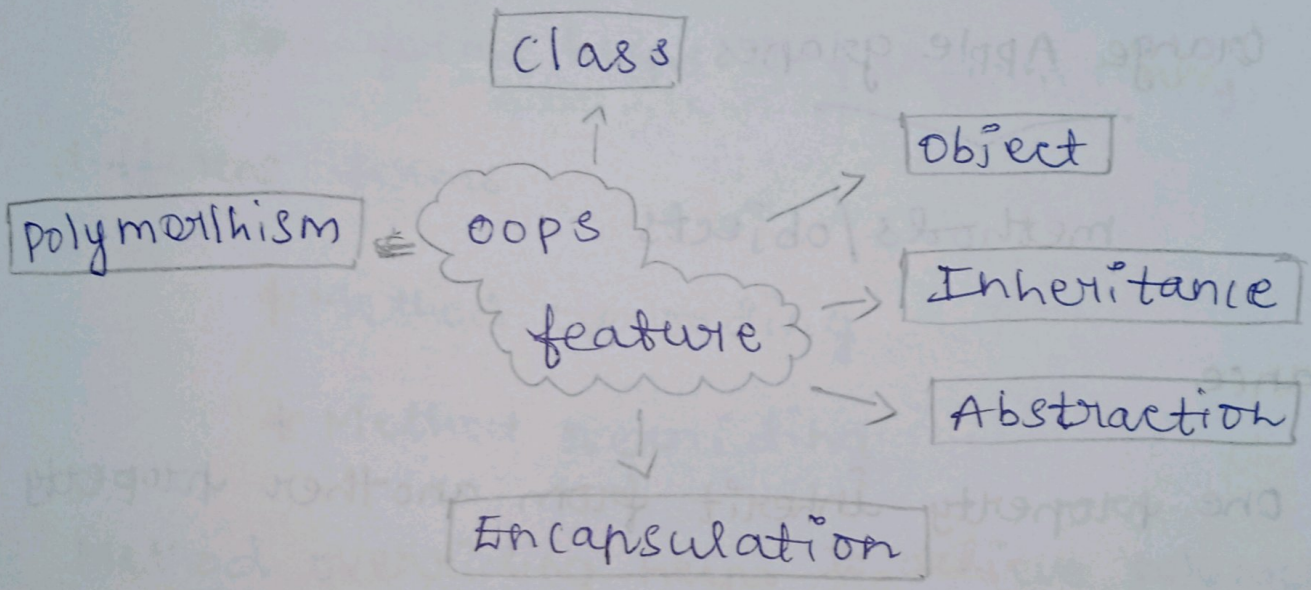
```
#include<iostream.h>
void main()
{
    int x;
    cout<<"Enter any number:";
    cin>>x;
    cout<<"Given number is"<<x;
    getch();
}
```

OUTPUT:

```
Enter any number: 5
Given number is 5
```

Oops feature

Oops stands for object oriented programming. which is aims to implement real world entities such as object, class, code reusuability.



Object

Object is an instance of class object has states and behaviour states means name, color, height behaviour means example dog -> barking, walking

Example : girl => description - states

!! name, color, height, weight voice,

Speaking => behaviour

Class: Blueprint of an object / methods

Example (fruits)

Fruits

Orange Apple grapes

methods / objects

Inheritance

One property inherit from another property called inheritance

Parents

Child 1 child 2

Encapsulation

wrapping up of data into class

Example. capsule medicine

Abstraction

Hiding irrelavant data/data hidden called abstraction.

Polymorphism

One forms helps to perform many different forms.

* Method overloading
* Method overriding

Method overriding helps to achieve polymorphism.

Functions in c++

Function can be defined as a block of code that perform a task when it is called.

Advantages

=> Code Reusability

=> Code optimization

Code Reusability

By creating a function, we don't need to write the code again and again

Code optimized

Code will be optimized and the logic can be implement. we no need to repeat the logic again and again

Types of Function

* library function
* User defined function

library function:

library function is a standard function available in stream.

User-defined function

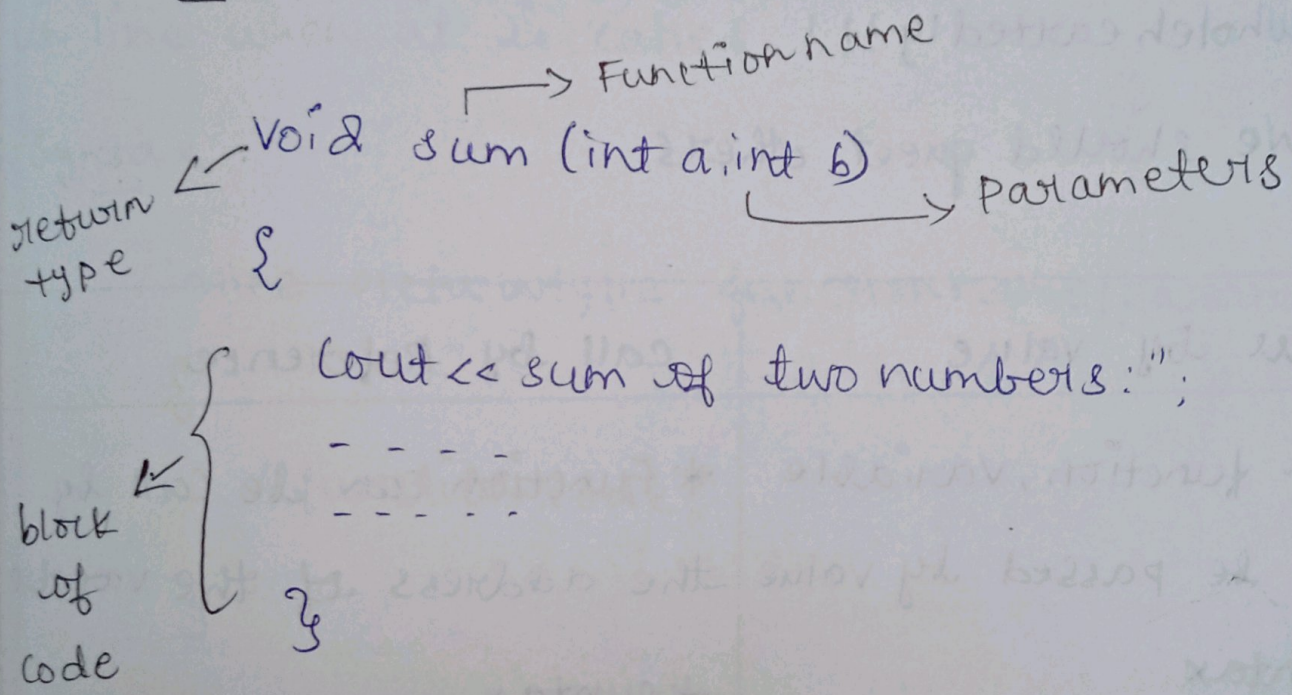
User ourself can create a function called user-defined function.

Simple function

Syntax return type function name (Parameters...)

```
{
    code to be execute
    ----
}
```

Example



Program:

```
#include<iostream.h>
void greet()
{
cout<<"wholeheartedly !!!\n we should greet
        others";
}
```

```
void main()
{
    greet(); // call the function
    getch();
}
```

output:

wholeheartedly!!!

We should greet others.

call by value	call by Reference
* In function, variable can be passed by value * syntax `void fun(int a, int b)` variable *Actual and formal arguments allocated memory allocation *It need more memory space	* Function can be call by the address of the variable * syntax `void fun(int &a, int &b)` call by address *Actual and formal arguments store in different location * It store in less memory

* It is less efficient * Pointer is absent	* It is more efficient * Pointer is present

Inline function:

Inline function is a function that is expanded in line when it is called.

Syntax

```
inline returntype functionname(parameters)
{
   //function code;
}
```

Example

```
#include<iostream.h>
inline int add (int a,int b)
{
   return(a+b);
}
void main
{
  cout<<" Addition of 2 number is:"<<add(5,10)
} getch();
```

output

Addition of 2 number is 15

Advantage of inline function

* It require less yeild of code for implementing inline function
* Function overheads doesn't occur

Disadvantage

* It is not capable of using for complex set of code.
* Binary execution file also become large.

Inline versus Macro

Inline Function	Macro
* These are functions provided by C++	* Macros are preprocessor directives
* Inline keyword is used to declare the function as inline	* #define is used to declare the macro
* It can be defined inside or outside the class	* It cannot be declared inside the class

* Inline functions are parsed by the compiler	* Macros are expanded by the c++ preprocessor
* Inline function can access the data member of the class.	* Macros cannot access the data member of the class
* Compiler replaces the function call with the function code	* C preprocessor replaces every occurrance of macro template with its ~~for~~ corresponding definiton
* Inline function follows strict Parameter type checking	* Macro does not follows parameter type checking
* Inline function may or may not be expanded by the compiler. Its depends upon the compiler's decision wheather to expand the function inline or not	* Macros are always expanded
* Can be used for debugging a program	* Cannot be used for debugging as they are expanded at pre-compile time

* inline int sum(int a, int b)
{
return (a+b);
}
	#define SUM (a,b) (a+b)

C++ function overloading:

* In C++, two functions can have the same name if the number and/or type of arguments passed is different. ~~These functions~~

* These, functions having the same name But different arguments are known as overloaded function. For example // same name different arguments

```
int test(){}
int test(int a) {}
float test(double a){}
int test(int a, double b){}
```

Here, all four functions are overloaded functions. Notice that the return types of all these

four functions are not the same. or

*overloaded functions may or may not have different return types. but they must have different arguments. For example: Error code

```
int test(int a) {}
double test(int b) {}
```

Program:

```
#include<iostream.h>
void print(int i)
{
 cout<<"Here is int"<<i<<endl;
}
void print(double f)
{
  cout<<"Here is float"<<f<<endl;
}
void main()
{
   print(10);
   Print(10.8);
   getch();
}
```

Output:

Here is int 10
Here is float 10.8

Default arguments in C++

In a function, arguments are defined as the values passed when a function is called. values passed are the source, and the receiving function is the destination.

Definition

A default argument is a value in the function declaration automatically assigned by the compiler if the calling function does not pass any value to that argument.

Characteristics for defining the default arguments

*The values passed in the default arguments are not constant. These values can be overwritten if the value is passed to the function. If not, the previously declared value retains

* During the calling of function, the values are copied from left to right.

* All the values that will be given default value will be on the right.

Example

* void function (int x, int y, int z=0)

  Explanation - The above function is valid. Here z is the value that is predefined as a part of the default argument.

* void function (int x, int z=0, int y)

  Explanation - The above function is invalid. Here z is the value defined in between, and it is not accepted.

Program

```
#include <iostream.h>
#include <conio.h>
int sum(int x, int y, int z=0, int w=0)
                                    // Here there are two values
                                    in the default argument
{
return(x+y+z+w);
}
int main()
{
   cout << sum(10,15) << endl; // x=10, y=15, z=0, w=0
   cout << sum(10,15,25) << endl; // x=10, y=15, z=25, w=0
   cout << sum(10,15,25,30) << endl; // x=10, y=15, z=25, w=30
   return 0;
}
```

output

```
25
50
80
```

Friend Function

A friend function can be given a special grant to access private and Protected members.

Friend function can be

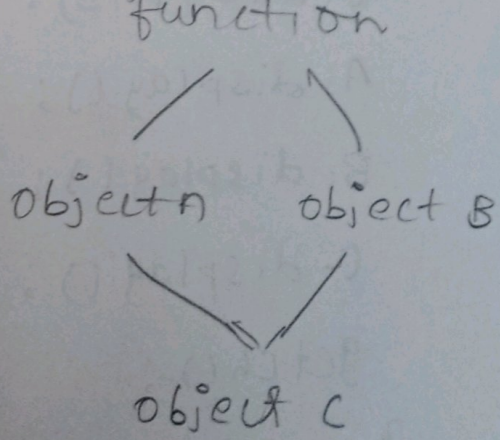
* A member of another class
* A global function

Syntax

```
class class_name
{
  friend datatype function name (arg1, arg2...)
}
```

Example

```
#include<iostream.h>
#include<conio.h>
class friends
{
  Public:
    int number;
    Void getdata()
    {
      cout<<"Enter a number:";
    } cin>>number;
```



```
    void add (friends a, friends b)
    {
       number = a.number + b.numbers;
    }
    void display()
    {
    cout<<"In Number is:"<<number;
    }
}
void main()
{
  friends A,B,C;
  clrscr();
  A.getdata();
  B.getdata();
  C.add(A,B);
  A.display();
  B.display();
  C.display();
  getch();
}
```

Virtual function

A virtual function is a member function in the base class that we expect to redefine in derived classes.

Syntax:

```
virtual return type function name (argument)
{
-----------
-----------
}
```

Example

```
Class Base {
    Public:
        virtual void print () {
            // code
        }
};
```

```
Class Derived : Public Base {
    Public :
        void Print () {
            // code
        }
};

int main () {
    Derived derived1 ;
    Base * base1 = & derived1 ;
    base1 -> print () ;
    return 0 ;
}
```

Print() of derived Class is called because print() of base Class is virtual