Introduction

program

Set of instrction performing specific task

programing language

Machine → 0's & 1's

Assembly → 0 & 1's and nemonic
high level ADD. SUB, MUL. DIV

high level language

→ English type

→ midlle level language

programing language.

→. C, c++, Java, c#

OOP

→ object oriented programing.

→ modeling data

→ it is an approach to develop s/w prog.

→ c++, .Net. java

OOPs

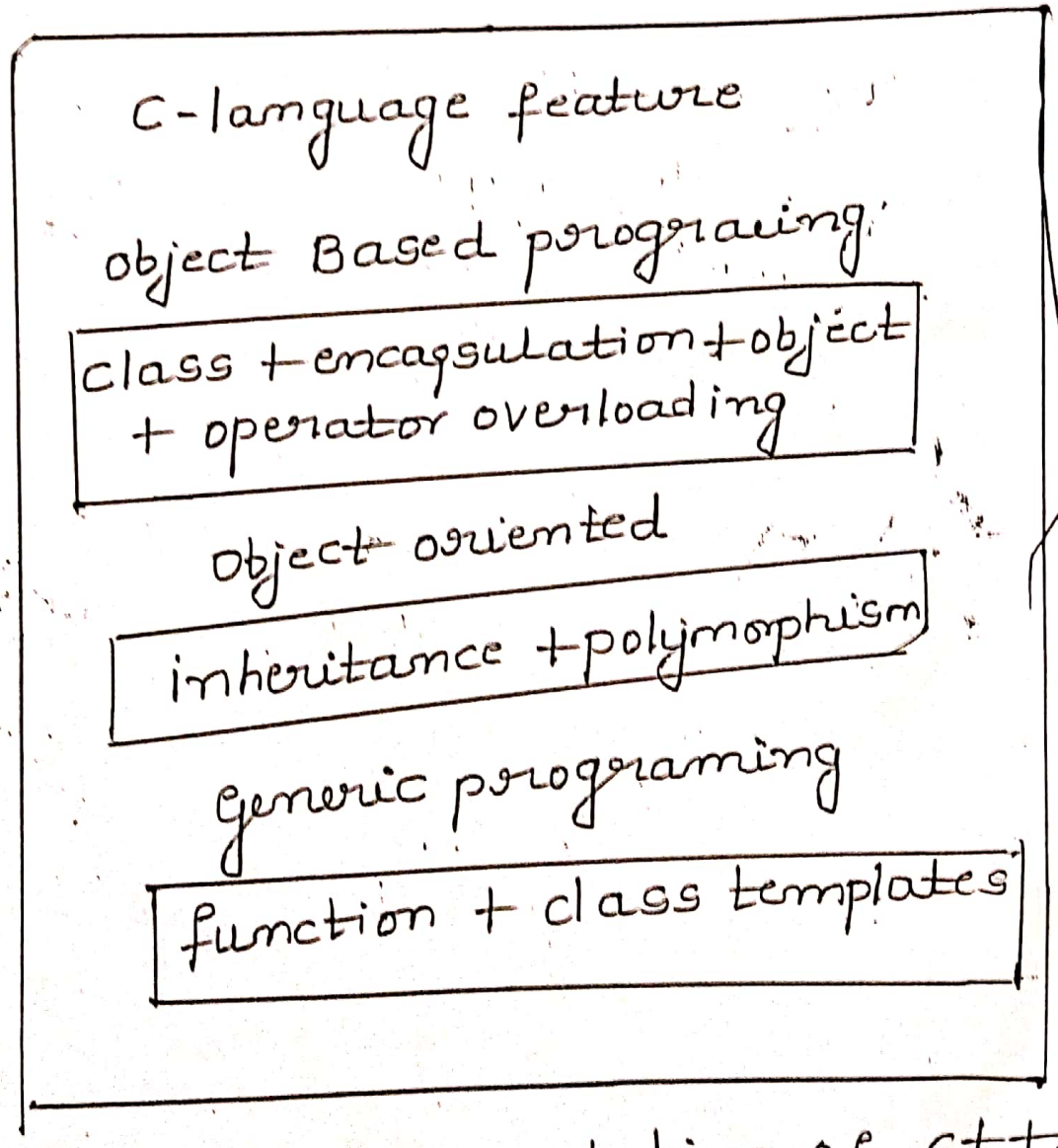
→ c++ is an extension of C

→ c++ is an object oriented programing language.

→ It was developed by Bjarne Stroustrup at AT and T Bell laboratories in Murray Hill, New Jersey USA, in the early 1980's.

→ c++ is an extension of c

object Based
object oriented
generic programming language.



pictorially representation of c++.

→ oops was introduce to overcome flaw in the procedural approach to programing such as

→ Reusablity

→ maintabilty

→ increasing the complexity

→ Application of oops

→ real time system

→ hypermedia

→ hypertext

→ AI

→ object oriented database.

Features of oops

Encapsulation is a defining feature of C++ using which data and code are kept together [bind] This ensures that the data is accessible to internal function and hence prevents data corruption

The data and related function are enclosed into a container called a clas

→ Inheritance is another important fe of object oriented approach.

Both data and associated functio can be inherited from Base class it is derived class. The Base class can given more common and general characteristics where as its derived class can be given specific charcteris This provides code reusability.

→ polymorphism is another defining feature of object oriented programing approach. it refers to a phenomenon in which one entity exists in many forms.

→ In c++ functions, operators can be made to exhibit polymorphic behaviour

In c++, There are 2 types of polymorphism.

→ static

→ dynamic

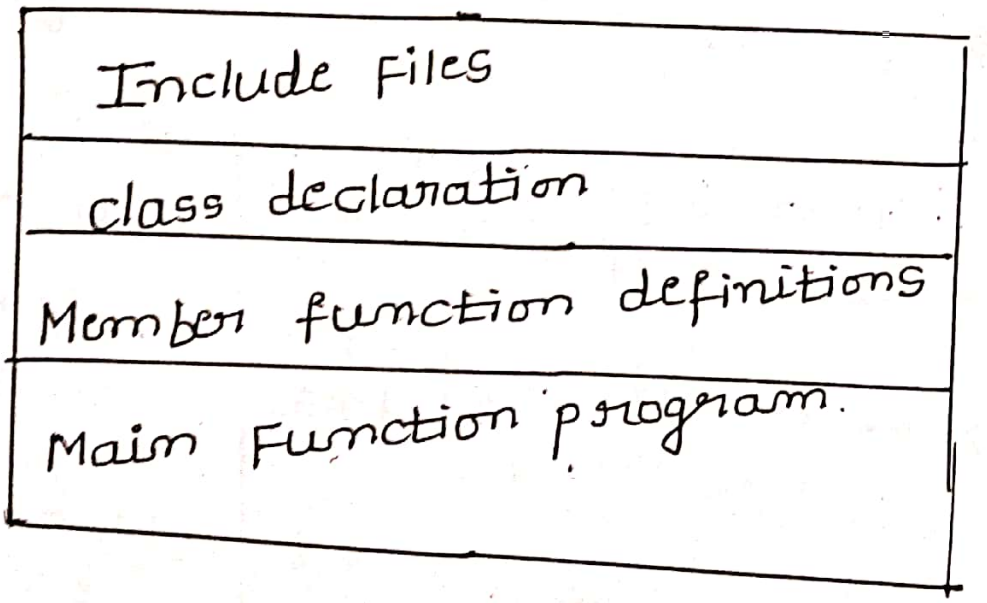
Comparison of C++ with C

C	C++
it is procedure oriented language	it is object oriented language
→ They run faster	They run slower
→ does not support polymorphism, Inheritance	support polymorphism Inheritance.
→ data is not secure	data is secure
→ Top-down approach	bottom-up approach
→ printf & scanf statement used.	cout and cin statement used.
→ :: does not use.	:: scope resolution operator used in c++
→ #include<stdio.h> header files used.	# include <iostream> header files used.
→ does not support constructor, pointers.	support pointers, constructor, destructor
→ function is a fundamental entity.	object is a fundmental entity.
→ doesnot support class and object	support class and object

(6) Simple c++ program

```
#include <iostream>
using namespace std;
int main()
{
  cout<<"C++ is better than C.\n";
  return 0;
}
```

Structure of c++ program



C++ EXAMPLE PROGRAM

Write a c++ program to read two number from the keyboard and display their average an the screen.

Average of two Numbers

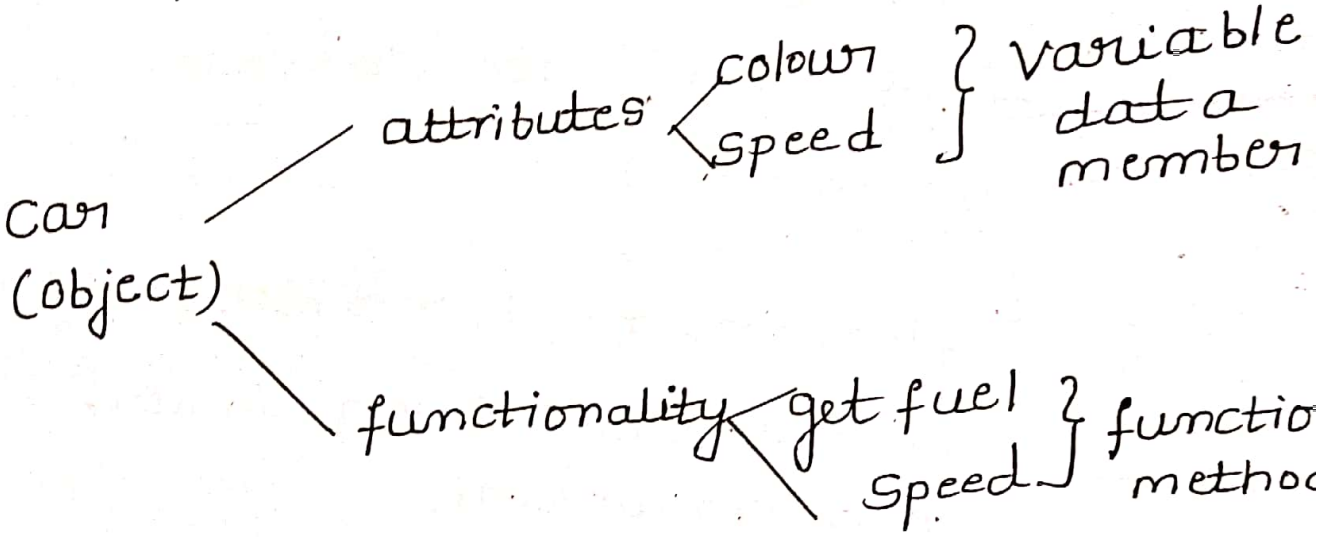
```
#include <iostream>
using namespace std;
int main()
{
  float number1, number2, sum, averag
  cout << Enter two number: ";
  Cin >> number1;
  Cin >> number2;
  Sum = number1 + number2;
  average = Sum/2;
  Cout << "Sum=" << Sum << "\n";
  Cout << "Average=" << Average << "\n";
  return 0;
}
```

O/P Enter two number
6.5 7.5
Sum = 14
Average = 7

Class and objects

class → collection of objects
Blueprint of an objects.

objects → instance of class
it is a piece of code which represent the real life Entity.



Class	objects
→ class is a data type	→ instance of class
→ generate object	it gives life to class.
→ does not's occupy memory location.	it occupies memory locations.
→ can't be manipulated.	can be manpulated.

Class in C++ provides the programmer the facility to bind data and function class are created using the keyword class.

→ The class declaration helps the programmer to create a new data type.

→ The data present inside the class is known as data members. The functions present inside the class is known as function members.

→ The general form of the class is as shown below

```
class class_name
{
access-specifier:
        data and functions
access-specifier
        data and functions
};
```

(10)

```
#include <iostream>
class simple_Interest
{
 private : float p, t, r, si;
 public : void set_data()
          {
            p = 100;
            t = 8.5;
            r = 3.5;
          }
         void compute()
         {
          si = (p*t*r)/100;
         }
          void disp()
          {
            cout<<si;
          }
};
 void main()
 {
   simple_Interest x;
    x.set_data();
    x.compute();
    x.disp();
}
```

In the above example [illegible]
members set_data(), compute() and disp() are member function. only these member functions would have exclusive right to access private data member

→ A non member function like main() cannot directly access data members since they have been declared as private using "private" access specifier. since the member function have been declared as public they can be accessed from main function.

→ The keyword private and public are called as the "access specifier" since they control The access to the data members of a class. Normally data members are placed under private category & member functions under public category.

→ variable of the class are called as "objects" The amount of memory allocated to an object depends upon data members of the class.

(iv) → All the member function sh
invoked by using an object as
in the above Example.

→ The classes and objects are the cor
concept of object oriented programin
That ensures data security and
data encapsulation.

Access members in c++

Access members in c++ are used for controlling the access of members of a class. They are also called as "access specifers" or "access modifiers".

In c++. There are 3 access members.

→ private
→ public
→ protected

private access members are accessible to anly member functions. In other words non member functions cannot access private members.

→ public access members are accessible to members functions as well as non member function.

→ protected access members are accessible to member functions of base class and derived class.

Structure and class are Related

→ class are syntactically similar to Struct.

→ one major difference blw class and Struct.

all the members are private in class
all the members are public in Structu

→ Struct and class both are equivalent.

→ c++ contain two keywords Struct & class

→ user defined data type.

```
Struct person
{
  char name[50];
  int age;[
  float salary;
};
```

(14)

difference b/w Class and struct in c++ ?

Class	structure
→ Keyword for the declaration: class	struct
→ reference type data.	value type data
→ object is created on the heap memory	stack memory
→ class can be inherited	can't be inherited
→ default access specifier private.	public
→ generally used for large amounts of data	smaller amounts of data
→ it can have all the types of constructor and destructor.	only parameterized constructor.

write a c++ program to assign data to members of a structure variable and display it.

```
#include<iostream>
using namespace std;
Struct person
  {
    char name [50];
    int age;
    float salary;
  };
int main ()
{
  person p1;
  cout << "Enter full name: ";
  cin.get(p1.name, 50);
  cout << "Enter age: ";
  cin >> p1.age;
  Cout << "Enter salary: ";
  cin >> p1.salary;
  cout << "\n Displaying Information."
                << endl;
 cout << "Name: " << p1.name << endl
 cout << "age: " << p1.age << endl;
 cout << "salary: " << p1.salary;

  return 0;
    }
```

(16) union and classes with

→ user defined data type

→ Union contain data member variable include constructor & destructor.

Syntax

```
union uniontypename
{
 type member-name;
}union-variable;
```

Example

```
#include <iostream>
union Emp
{int num;
 double sal;
};
int main()
{
  Emp value;
 value.num=2;
cout<<" Employee number :: "<<value.num
<<"\n Salary is :: "<<value.sal<<endl
 value.sal2000.0;
cout<<" Employee number :: "<<value.num
<<"\n salary is :: "<<value.sal<<
                    endl;
return 0;
}
```

Friend Functions

Friend Functions are non member functio which has a special right to access the private data of class.

→ A friend function has access to all private and protected members of the class for which it is a friend.

→ A friend function is created using "friend" keyword & should be placed with in a class to which it is intende to be a friend.

Example.

```
#include <iostream.h>
Class exmp
{
 private: int i;
          int j;
 public   : exmp()
          {
            i=10;
            j=20;
          }
```

```
    friend void sum();
};
void sum()
{
    int c;
    exmp x;
    c=x.i+x.j;
    cout<<c;
}
    void main()
    {
      sum();
    }
```

As noticed from the above example. Even though sum is a non member function it is able to access private data of the class exmp this is possible because sum() function has been made friend of class exmp.

Advantages of using friend functions

→ Friend functions can be useful when we are overloading certain types of operators.

→ Friend functions make the creation of some types of I/O functions easier

→ Friend functions may be desirable in cases as two or more classes may contain members that are interrelat relative to other parts of the program

Inline Functions

→ c++ proposes a new feature called Inline function.

→ An inline function is a function that is Expanded in line when it is invoked that is the compiler replaces the function call with the corresponding function code.

syntax:

```
inline function-header
{
    function-body
}
```

Adavantage:

→ Inline functions are very important create very efficient code.

(21) Inline function Example

```
#include <iostream>
using namespace std;
inline float mul (float x, float y)
{
  return (x*y);
}
inline double div (double p, double q)
{
   return (p/q);
}
int main()
{
  float a=12.345;
  float b=9.82;
  cout<< mul (a,b) <<"\n";
  cout<<'div (a,b) <<"\n";
  return 0;
}
```

Some of the situations where inline Expansio may not work are:

→ For function returning values, if a loop a switch or a goto exists.

→ if inline functions are recursive.

→ if functions contain static variables.

→ For functions not returning values, if a return statement exists.

Constructors

→ constructor is a special member function whose task is to initialize the objects of its class. it is special because its name is the same as the class name.

→ constructor is invoked whenver an object of its associated class is created.

Characteristics.

→ They should be declared in the public section.

→ They cannot be inherited.

→ constructor cannot be virtual.

→ They are invoked automatically when the objects are created.

3types

→ default → not take any argument

→ parameterized → take one or more argumer

→ copy → copy object into another object.

Parameterized Constructors

→ The constructors that can take ar
are called parameterized constru

features

→ constructor can be overloaded

→ Explicitly defined by the progra

Example:

```
#include <iostream>
using namespace std;
class integer
{
    int m, n;
public:
  integer (int, int);
   void display (void)
{
cout<<"m ="<<m <<"\n";
cout <<"n ="<<n <<"\n";
}
};
integer:: integer (int x, int y)
{
   m=x; n=y;
}
```

```
int main()
{
integer int1(0,100);
integer int2= integer(25.75);
cout <<"\n OBJECT1 "<<"\n";
int1.display();
cout<<"\n OBJECT2"<<"\n";
int2.display();
return 0;
}
```

(23)

(25) Scope Resolution operator

The scope resolution operator is denoted by :: it can be used for following purpose.

1) defining member function outside the Class

```
#include <iostream>
Class simple-interest
{
 private : float p;
           float t;
           float r;
           float si;

  public : void set_data();
           void compute();
           void disp();
};
void simple-interest :: compute()
 {
   si = (p*t*r) /100;
 }
void simple-interest ::disp()
 {   cout<<si;
 }
```

6y. As noticed in the above example. the member functions are only declared in the class however they are defined outside the class using scope resolution "::" operator.

2) it can be used to access global variable. in such cases where local and global variabl have same name.

```
int i;
void f()
{
  int i;
  ::i=10;
}
```

As noticed in the above example, The globa and local variables have same name. Therefore using "::" operator global variabl is accessed.

Static member function which is used to a

A static data member

Example:

```
#include <iostream>
using namespace std;
Class Demo
{
 private:
    Static int x;
    Static int y;        } data memb
 public:
 Static void print()     } member function
 {
 cout << "value of x:" << x << endl;
 cout << "value of y:" << y << endl;
 }
};
int Demo :: X=10;
int Demo :: y=20;        } Static data member intilializatio
  int main()
  {
    Demo OB;
    cout << " printing through
                Object name:" << endl;
    OB.print();
    cout << " printing through class
          name:" << endl;
    Demo:: print(); return 0; }
```

Function Overloading [5marks]

- Function Overloading means 2 or more functions in a program can have same name but differ in the number of arguments or data type of arguments.
- The compiler automatically decides about the appropriate function to be executed by comparing the arguments and their data types used in function call.

Eg: int function_1(int x)
int function_1(int x, float y)
Here function_1 is overloaded twice.

Need for function overloading. (Advantages)

(i) Less effort for the programmers to remember different functions names at the function call.
(ii) Code maintenance is easy.
(iii) Easy interface between objects and functions.
(iv) Easy to understood the flow of information and easy to debug.

Restrictions on function overloading.

(i) The overloaded function must differ in either in no of argument or data type of arguments.
(ii) If type-def is used for declaring user defined names for variables then the function is not considered as overloaded.

LAB: WAP to find area of square, rectangle and Δ^{le} using function Overloading

```
#include <iostream.h>
#include <iomanip.h>
class area
{ private: ar, ar1, ar2;
  public: void Ar(int a)
          { cout<<"Enter side of square"<<endl;
            cin>>a;
            ar = a*a
            cout<<"Area"<<"ar<<endl;
          };
void Ar(int l, int b)
{ cout<<" Enter l & b"<<endl;
  cin>>l>>b;
  ar1 = l*b. cout<<"Area is"<<ar
```

* What is function overloading? Explain with example.

```
cin>> x >>y>>
s= (x+y+z)/2;
ar2 = sqrt (s*(s-x)*(s-y)*(s-z));
cout << ar2 << endl;
};
void main
{
Area A1;
A1 Ar (int a);
A1. Ar(int l, b);
A1. Ar(int s, x, y, z);
};
```

—OR—

```
#include <iostream.h>
#include <iomanip.h>
class foverloading
{
  public : float area(float a)
           {
            return (a*a);
           }
           float area(float a, float b)
           {
            return (a*b);
           }
           float area (float a, float b, float c)
           {
            float s;
             s=(a+b+c)/2;
            return sqrt (s*(s-a)*(s-b)*(s-c));
};
void main()
{
  foverloading F1;
  float x,y,z;
  int choice;
  cout<<"Enter the choice 1,2, or 3"<<endl;
  if (choice==1)
  {
   cout<<"Enter the sides of the square"<<endl;
   cin>>x;
```

Copy Constructor.

It is a parameterised constructor using which one object can be copied to another object. Copy constructor is used in following situations.

(i) The initialisation of an object by another object of the same class.

(ii) Return of objects

existing object.
- when a new object is created and the existing object is passed as parameter to it.
- when an object is passed to a function as non reference parameter.

LAB:
WAP to find the sum of the series $1+x+x^2+x^3+\ldots+x^n$ using copy constructor

```
#include <iostream.h>
#include <iomanip.h>
class series
{
private: int n,i;
         float x, sum;
public: series (int n, float x)
        {
          sum = 1;
        }
        float calculate()
        {
          for (i=1; i<=n; i++)
            sum = sum+1+pow(x,i);
            return (sum);
        }
};
void main()
{ int a, float b;
  cout << "Enter the value of a and b" << endl;
  cin >> a >> b;
  Series S1 (a,b);
  Series S2 = S1;
  S2. calculate ();
  cout << "sum of series from object 1 = " << S1. calu
                                                ();
  cout << "sum of series from object 2 = " << S2. calcula

};
```

function which is automa