3.

```
#include<bits\stdc++.h>
Using namespace std;
#include<string>
class member{
    char name[20], address[40];
    double number;
    int age;
  Public;
    int salary;
    Void input;
  { Cout<<endl;
    Cout<<"name"<<endl;
    cin.getline(name,20);
    cout<<"age:"<<endl;
    cin>>age;
    Cout<<"phone number"<<endl;
    cin>>number;
    Cout<<"Address"<<endl;
    Cin.getline(address,40);
    Cout<<"salary:"<<endle;
    Cin>>salary;
}
```

```
    Void display()
    { cout<<endl;
      cout<<"name"<<Name<<endl;
      cout<<"Age"<<age<<endl;
      cout<<"phone number"<<number<<endl;
      cout<<"Address"<<address<<endl;
      cout<<"salary"<<salary<<endl;
    }
};
class employee : Public member{
  char specialization[20],department[20];
  Public:
  Void input()
  {
    cout<<"\n\t Enter employee details\t\n";
    member::input();
    cout<<"specialization"<<endl;
    cin.getline(specialisation,20);
    cout<<"Department"<<endl;
    cin.getline(department,20);
  }
  Void display()
  {
    cout<<"\n\t Displaying employeedetails\t\n";
    member::display();
    cout<<"specialisation"<<specialisation<<endl;
    cout<<"Department"<<Department<<endl;
  }
  Void salary(){
  cout<<"\n salary of member is"<<salary<<endl;
};
```

```
class manager : publil member {
    char specialisation [20], department [20];
Rubic:
    Void input()
{
  cout<< "\n\t enter manager details \t\n";
  member :: input ();
  cout <<"specialisation" <<endl;
  cin.getline (specialisation, 20);
  cout<<" Depavtment" <<endl;
  cin.getline (department, 20);
}
  Void display()
{
  cout <<"\n\t displaying manager details \n\t";
  member :: display ();
  cout <<" specialisation "<< specialzation <<endl;
  cout<<" Depavtment" <<department<< endl;
}
  Void Print salary()
    { cout <<" In salary of the member is " <<salary<<
                                          end;
    }
};
 int main()
{
 employer e;
  manager m;
  e. input();
  m. input ();
  e. display();
  e. Print salary();
  m: display ()
  m. Prit salary ();
}
```

2-

```
#include<bits\stdc++.h>
Using namespace std;
Void sort(int a[], int a_size)
{
    int lo = 0;
    int hi = a_size -1;
    int mid = 0;
   while (mid<=hi){
       Switch (a[mid]){

        case 0;
           swap(a[lo++], a[mid++]);
           break;
        case 1;
           mid++;
           break;
         case 2:
           swap(a[mid], a[hi--]);
           break;
       }
    }
}
  Void Print array (int a[], int a_size)
  {
      For (int i=0; i<a_size; i++)
          cout<<a[i]<<" ";
  }
  int main()
{ int a[]={1, 2 2 0 0 1 2};
  int n = size of(a)/sizeof(a[0]);
```

```
    Sort(a,n);
    cout<<"   ";
    Printarray(a,n);
    return 0;
}
```

Polymorphism :- it means the same entity behaves differently in different scenarios.

eg:-
```
int a=6;
int b=6;
int sum=a+b;
```

```
String Firstname = "great";
String lastname = "learning";
String name = Firstname+lastname;
```

in example the '+' operator can perform two functions in first it has performed as sum in second it has performed concatenation.

Types of Polymorphism :-

1- Complile Time polymorphism
2- Runtime Polymorphism

CompileTime polymorphism :-

in it, a function is called at the time of Program Compilation. we call this type of polymorphism as early binding or static binding. in Compile there are two type of Poly morphism

1- Function overloading :- It means one Function can perform many tasks. in the function over loading function will call at the time of Program Compilation. it is an example of compile polymorphism

2- Operator overloading :- it means defining additional tasks to operators changing is actual meaning. we do this by using operator Function it is used defind data types

Runtime Polymorphism:-

In it Functions are called at the time the Program execution. it is known as late binding or dynamic binding. it is achieved by using Virtual Function and pointers. It provides slow execution as it is known at run time. it has two type of Runtime polymorphism

Function overriding:- we give the new definition to the base class Function in the derived class. at that time we can say the base has been overriden. in it more than one method has the same name with different types of parameter list.

Virtual Function:- A virtual Function is declared by keyword Virtual. the return type of virtual Function may be int, float, void. it is a member Function in the base class. A virtual Function is not static. it tells to compiler to Perform dynamic binding

eg:-
```
#include <bits/stdc++>
  Using namespace std;
 class Compiletime
{ Public:
     Void display (int x)
      { cout<<" the int is:"<<x<<endl;
      }
      Void display (char ch)
     { cout<<" the character is: "<<ch<<endl;
};  }
```

```
class RuntimeParent
{
Public;
    Void sayhello()
    {
     cout << "Hello From me" <<endl;
     }
};
class Runtime child Public Runtime Parent
{
 Public;
     Void Sayhello()
     {
       cout<<"Hello From You" <<endl;
};
```

2- Parameterized constructor :-

a constructor with parameter is known as Parameterized constructor. This is the preffered method to initialize membordata

```
eg:- class wall {
        Public;
        wall ( double len, double hgt) {
          length = len;
          height = hgt;
        }
     };
```

3- Copy Constructor :-

the copy constructor in C++ is used to copy data of one object to another

```
eg:- class wall {
        Public;
        wall (wall &obj) {
            length = obj. length;
            hieght = obj. hight;
        }
```

4- Static Constructor :- it is used to initialize any static data or to perform a pardicular action that needs to be performed only ouce. it is not in C++

5- Private Constructor :- this special instince constructor is generally used in class that contain static members only: it is not in C++

Procedural oriented :-

* in Procedural Progmming Programs is divided into small parts called Function
* Procedural Programms follows top down approach
* there is no access specifier in Procedural Programming
* it does not have any Proper way hiding data so is less secure
* adding new data function is not easy
* in it overloading is not possible

object oriented :-

* in it Programs is divided into small parts called object
* it Follows bottom up approach
* it have access specifiers like Private, Public Protected etc
* adding new data and Function is easy
* it Provide data hiding so it is more secure
* overload is Possible in object oriented.

1. new Keyword :- this keyword is an operator which denotes a request of memory allocation on the heap. when you create an object of class using new keyword

Delete Keyword :- it is an operator that is used to destroy array and non-array objects which created by new expression

ex :-

```
#include <bits/stdc++.h>
int main()
{
  int* p = NULL;
  p = new (nothrow) int;
  if(!p)
     cout<<"allocation of memory Failed\n";
  else
    {
      *p = 29
       cout<<"value of p: "<<*p<<endl;
    }
  float *r = new float(75.25);
  cout<<"value of r: "<<*r<<endl;
  int n=5;
  int* q = new (nothrow) int[n];
  if(!q)
      cout<<"allocation of memory Failed\n";
  else
  {  for(int i=0;i<n;i++)
     q[i] = i++;
     cout<<"value store in block memory";
```

```
For (int i=0;i<n;i++)
   Cout<<q[i]<<" ";
}
 delet p;
 delet r;
 delet[]q;
 veturn 0;
}
```

2- A constructor is a member Fuction of a class which intialize object of a class. in C++ constructor is automatically called when object create. it is a special member of class

Types of Constructors:-

1- Default constructor:-

A constructor with no Parameter is known as a default constructor

```
eg:- class wall {
        Public;
        wall() {
        }
};
```

1- Local Variable are stored in area called stack

2- No Compiler Error

3- when the inheritance is private, the private method in methods in base class are inaccassible in the derived class

4- The number of times destructor is called depends on Numbers of objects created

5- Type Conversion is automatic whereas type casting is explict is False