Assignment -2

1) D

Stack

2) C

Compiler Error in line "Derived *dp = new Base;"

3) A

Inaccessible

4) A

The number of times destructor is called depends on Number of objects created

5) A

True

SHORT ANSWERS

1) The heap memory allocation is requested by the new operator.
It initialises the memory to the pointer variable and returns its address if enough memory is available.
The delete operator is used to deallocate the memory. User has privilege to deallocate the created pointer variable by this delete operator.

```
#include
using namespace std;

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+1;

cout << "Value store in block of memory: ";
for (int i = 0; i < n; i++)
cout << q[i] << " ";
   }
delete p;
delete r;
delete[] q;

return 0;
}
```

2) A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

Three tpyes of constructor are :

Default

parameterized

Copy

Procedural Oriented Programming :

-> In procedural programming, program is divided into small parts called functions.

-> Procedural programming follows top down approach.

-> There is no access specifier in procedural programming.

-> Adding new data and function is not easy.

-> Procedural programming does not have any proper way for hiding data so it is less secure.

-> In procedural programming, overloading is not possible.

-> In procedural programming, function is more important than data.

-> Procedural programming is based on unreal world.

Examples: C, FORTRAN, Pascal, Basic etc.

Object Oriented Programming :

-> In object oriented programming, program is divided into small parts called objects.

-> Object oriented programming follows bottom up approach.

-> OOP have access specifiers like private, public, protected etc.

-> Adding new data and function is easy.

-> Object oriented programming provides data hiding so it is more secure.

-> Overloading is possible in object oriented programming.

-> In OOP data is more important than function.

-> Object oriented programming is based on real world.

Examples: C++, Java, Python, C# etc.

LONG ANSWERS

A) Polymorphism in C++ means, the same entity (function or object) behaves differently in different scenarios.

For example :

The "+" operator in c++ can perform two specific functions at two different scenarios i.e when the "+" operator is used in numbers, it performs addition and the same "+" operator is used in the string, it performs concatenation.

Categorised into :

1. Compile time polymorphism
2. Run time polymorphism

Compile Time Polymorphism :

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding.

Function overloading and operator overloading is the type of Compile time polymorphism.

I. Function Overloading

Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism.

```
#include
using namespace std;
class Addition {
public:
int ADD(int X,int Y)
{
return X+Y;
    }
int ADD() {
string a= "HELLO";
string b="World";
string c= a+b;
cout<
    }
};
int main(void) {
Addition obj;
cout< obj.ADD();
return 0;
}
```

II.Operator Overloading

Operator overloading means defining additional tasks to operators without changing its actual meaning. We do this by using operator function.
The purpose of operator overloading is to provide a special meaning to the user-defined data types.

```
#include
using namespace std;
class A
{

string x;
public:
A(){}
A(string i)
{
x=i;
   }
void operator+(A);
void display();
};

void A:: operator+(A a)
{
```

```
string m = x+a.x;
cout<<"The result of the addition of two objects is : "<
}
int main()
{
A a1("Welcome");
A a2("back");
a1+a2;
return 0;
}
```

2.Runtime Polymorphism

In a Runtime polymorphism, functions are called at the time the program execution. Hence, it is known as late binding or dynamic binding.
Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.
It is achieved by using virtual functions and pointers. It provides slow execution as it is known at the run time. Thus, It is more flexible as all the things executed at the run time.

I. Function overriding

In function overriding, we give the new definition to base class function in the derived class. At that time, we can say the base function has been overridden. It can be only possible in the 'derived class'.

```
#include
using namespace std;
class Animal {
public:
void function(){
cout<<"Eating..."<    }
};
class Man: public Animal
{
public:
void function()
{
cout<<"Walking ..."<    }
};
int main(void) {
Animal A =Animal();
A.function();  //parent class object
Man m = Man();
m.function();  //child class object
return 0;
}
```

II. Virtual Function

A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void.

A virtual function is a member function in the base class. We can redefine it in a derived class.

```
#include
using namespace std;
class Add
{
int x=5, y=20;
public:
void display()
{
cout << "Value of x is : " << x+y<    }
};
class Substract: public Add
{
int y = 10,z=30;
public:
void display()
{
cout << "Value of y is : " <    }
};
```

```
int main()
{
Add *m;
Substract s;
m = &s;
m->display();
return 0;
}
```

Pure virtual Function

When the function has no definition, we call such functions as "Do-nothing function or Pure virtual function". The declaration of this function happens in the base class with no definition.

```
#include
using namespace std;
class Animal
{
public:
virtual void show() = 0; /Pure virtual function declaration.
};
class Man: public Animal
{
```

```
public:
void show()
{
cout << "Man is the part of animal husbandry " << endl;
}
};
int main()
{
Animal *aptr; //Base class pointer
//Animal a;
Man m; //derived class object creation.
aptr = &m;
aptr->show();
return 0;
}
```

B)

```
#include
using namespace std;

void sort012(int a[], int arr_size)
{
int lo = 0;
int hi = arr_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 printArray(int arr[], int arr_size)
{
for (int i = 0; i < arr_size; i++)
cout << arr[i] << " ";
}

int main()
{
int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
int n = sizeof(arr) / sizeof(arr[0]);

sort012(arr, n);

cout << "array after segregation ";

printArray(arr, n);

return 0;
}
```

C)

```
#include
#include
using namespace std;

class member{

char name[20], address[40];
double number;
int age;

public:
int salary;
void input()
{  cout<cout<<"Name : "< cin.getline(name, 20);
cout<<"Age : "< cin>>age;
cout<<"Phone Number : "< cin>>number;
cout<<"Address : "< cin.getline(address, 40);
cout<<"Salary : "< cin>>salary;

 }
void display()
{  cout<cout<<"Name : "< cout<<"Age : "< cout<<"Phone Number : "<
cout<<"Address : "< cout<<"Salary : "<
 }
};
```

```
class employee : public member{
char specialization[20], department[20];
public:
void input()
{
cout<<"\n \t Enter Employee Details \t \n";
member::input();
cout<<"Specialization : "< cin.getline(specialization, 20);
cout<<"Department : "< cin.getline(department, 20);
  }
void display()
{
cout<<"\n \t Displaying Employee Details \t \n";
member::display();
cout<<"Specialization : "< cout<<"Department : "<  }
void printSalary()
{
cout<<"\n Salary of the member is : "<  }


};
class manager : public member{
char specialization[20], department[20];
public:
void input()
{
```

```
cout<<"\n \t Enter Manager Details \t \n";
member::input();
cout<<"Specialization : "< cin.getline(specialization, 20);
cout<<"Department : "< cin.getline(department, 20);
}
void display()
{
cout<<"\n \t Displaying Manager Details \t \n";
member::display();
cout<<"Specialization : "< cout<<"Department : "< }
void printSalary()
{
cout<<"\n Salary of the member is : "< }
};
int main()
{
employee e;
manager m;
e.input();
m.input();
e.display();
e.printSalary();
m.display();
m.printSalary();
}
```