Assignment - 2

Choose :-

1. d) stack

2. c) compile 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. B) False.

Short answer type question ?

1. a) New operator (keyword) :

The new operation request for the memory allocation in heap. If the sufficient memory is availble, it initialize the memory to the pointer variable and returns its address.

Syntax of new operator :-

pointer_variable = new data type ;

Syntax to initialize the memory :-

pointer_variable = new data type (value) ;

Syntax to allocate a block of memory :-

pointer_variable = new datatype [size];

Code :-

```
#include <iostream>
using namespace std;
int main() {
{
  int *ptr1 = NULL;
  ptr1 = new int;
  float *ptr2 = new float (223.324);
  int *ptr3 = new int [28];
  *ptr = 28;
  cout << "value of pointer variable 1 :" << *ptr1 << endl;
  cout << "value of pointer variable 2 :" << *ptr2 << endl;
  if (!ptr3)
  cout << "Allocation of memory failed \n";
  else
  {
    for (int i = 10; i < 15, i++)
    ptr3[i] = i+1;
    cout << "Value of store in block of memory :";
    for (int i = 10; i < 15; i++)
    cout << ptr3[i] << " ";
  }
  return 0;
}
```

Output:

Value of pointer variable 1 : 28

Value of pointer variable 2 : 223.324

Value of store in block of memory :

11 12 13 14 15

The Delete operator :-

The delete operator is used to dellocate the memory. user has privelege to deallocate the created pointer variable by this delete operator.

Syntax of delete operator :

```
delete pointer_variable;
```

Syntax of to delete the block of allocated memory:

```
delete [ ] pointer_variable;
```

Code:

```
#include <iostream>
using namespace std;
int main ()
{
   int *ptr1 = NULL;
```

```
ptr1 = new int ;
float *ptr 2 = new float (299.121);
int *ptr 3 = new int [28];
*ptr 1 = 28;
Cout << "Value of pointer variable 1 :" << *ptr1 << endl;
Cout << "value of pointer variable 2 :" << *ptr 2 << endl;
if (!ptr 3)
Cout << "Allocation of memory failed \n";
else {
    for (int i = 10 ; i < 15, i++)
    ptr 3 [i] = i+1;
    Cout << "value of store in block of memory :";
   for (int i = 10; i < 15; i++)
    cout << ptr 3 [i] << " ";
   }
   delete ptr 1;
   delete ptr 2;
   delete [] ptr3;
   return 0;
}
```

Output :

Value of pointer variable 1 : 28

Value of pointer variable 2 : 299.121

Value of store in block of memory : 11 12 13 14 15

2) All Constructor :

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

Constructors are required to construct an object of the class : i.e : It is used to initialize all class data members.

Types of Constructors :

1. Default Constructors :

This constructor doesn't take any argument. It has no problem.

Example :-

```
#include <iostream>
using namespace std;
class Construct
{
  Public :
     int a, b;
     Construct ( )
     {
        a = 10;
        b = 20;
     }
```

```
};
int main()
{
    construct c;
    cout << "a:" << c.a << endl;
    Cout << "b:" << c.b << endl;
    return 1;
}
```

Output:

a : 10

b : 20

2. Parameterized Constructors:

It is possible to pass arguments to Constructors Typically, these arguments help initialize object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructors body, use the parameters to initialize the object.

Example:

```
#include <iostream>
using namespace std;
class point
{ private:
    int x, y;
  public:
    Point (int x1, int y1)
```

```
{
    x = x1;
    y = y1;
}
int getx()
{
    return x;
}
int gety()
{
    return y;
}
};
int main()
{
    point p1(10,15);
    cout << "p1.x =" << p1.getx() << ", p1.y
                                  = " << p1.getx();
    return 0;
}
```

Output :

P1. X = 10, P1. y = 15.

3. Copy Constructor :

A copy constructor is a member function which initialize an object using another object of the same class.

Example:

```
#include "iostream"
using namespace std;
class point
{ private:
   double x, y;
  Public:
   point (double px, double py)
   {
     x = px, y = py;
   }
};
int main (void).
  {
    point a[10];
    point b = point (5, 6);
  }
```

(3) Procedural Oriented Programming

Object Oriented programming.

① In procedural programming program is divided into small parts called functions

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

② Procedural programming follow top down approach

Object oriented programming follows bottom up approach.

③	There is no access specifier in procedural programming	object oriented programming have access specifiers like private, public, protected etc.
④	Adding new data and functions is not easy	Adding new data and function is easy.
⑤	Procedural programming does not have any proper way for hiding data so it is less secure	Object oriented programming provides data hiding so it is more secure.
⑥	Overloading is not possible	Overloading is possible.
⑦	function is more important than data	data is more important than function.
⑧	It is based on unreal world	It is based on real world.
⑨	Eg: C, FORTAN Pascal, Basic etc	Eg: C++, Java, python, C# etc.

Long Answers:-

A) Types of polymorphism:-

1. Compile Time
   * Function overloading
   * Operator overloading.
2. Run Time.
   * Virtual function.

Compile time polymorphism:

This type of polymorphism is achieved by function overloading or operator overloading.

Function Overloading:

When there are multiple functions with same name but different parameters then these function are said to be overloaded. Functions can be overloaded by change in number of arguments or change in type of arguments.

Code:

```
#include <bits/stdc++.h>
using namespace std;
class obj
{ public :
  Void func (int x)
```

```
{
  cout << "value of x is " << x << endl;
}
void func (double x).
  {
    cout << "value of x is " << x << endl;
  }
void func (int x, int y)
  {
    cout << "value of x and y is
          " << x << ", " << y << endl;
  }
};
int main ()
{
  obj obj1;
  obj1.func(7);
  obj1.func(9.132);
  obj1.func(85, 64);
  return 0;
}
```

Output:

Value of x is 7

Value of x is 9.132

Value of x and y is 85, 64.

Operator Overloading :

C++ also provide option to overload operators. For eg: we can make the operator ('+') for string class to concanterate two strings.

Code :-

```
#include <bits/stdc++.h>
using namespace std;
class complex
{
  private:
    int real, imag;
  public:
    Complex (int r=0, int i=0)
        {real = r; imag = i;}
    complex operator + (complex const &obj)
    {
        complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print() { cout << real << " + i"
                   << imag << endl; }
};
int main()
{ complex C1(10,5), C2(2,4);
  complex C3 = C1+C2;
  C3.print();
}
```

Output :

12+19

2. Runtime polymorphism:-

It is achieved by function overriding.

Function overriding :-

It occurs when a derived class has a definition for one of the member functions of the base class

```
#include <bits/stdc++.h>
using namespace std;
class base
 {
   public:
     virtual void print( )
      { cout << "print base class" << endl; }
     void show ( )
      { cout << " show base class" << endl; }.
     };
     class derived : public base
     {
      public:
       void print ( )
        { cout << "print derived class" << endl; }
       void show ( )
       { cout << "show derived class" << endl; }
     };
```

```
int main ()
{
    base *bptr;
    derived d;
    bptr = &d;
    bptr -> print ();
    bptr -> show ();
    return 0;
}
```

Output:

point derived class
show base class.

B)
```
# include <bits/stdc++.h>
using namespace std;
void sort012 (int* arr, int n)
{
    int count=0, count1 = 0, count2 = 0;
    for (int i=0; i<n; i++)
    {
        if (arr[i] == 0)
            count0++;
        if (arr[i] == 1)
            count1++;
        if (arr[i] == 2)
            count2++; }
```

```
for (int i = 0; i < count 0; i++)
    arr[i] = 0;
for (int i = count 0; i < (count 0 + count 1); i++)
    arr[i] = 1;
for (int i = (count 0 + count 1); i < n; i++).
    arr[i] = 2;
return;
}.
Void printArray(int* arr, int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
        cout << endl;
    }
int main()
{
    int arr[] = {1, 1, 2, 2, 0, 0, 2, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    Sort 012 (arr, n);
    printArray (arr, n);
    return 0;
}
```

Output:

0 0 1 1 1 2 2 2

C.
```
#include <bits/stdc++.h>
#include <string>
using namespace std;

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:" << endl;
       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.gethine (specialization, 20);
   cout << " Department :" << endl;
   cin.gethine (department 20);
  }
  void display ()
  {
  cout << " \n\t Displaying employee details \t\n
```

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

```
cout << "Specialization : " << Specialization << endl;
cout << "Department : " << Department << endl;
}
void printSalary()
{
cout << "\n Salary of the member is :"
}                                << Salary << endl;
};
int main()
{
employee e;
manager m;
e.input();
m.input();
e.display();
e.printSalary();
m.display;
m.printSalary();
}
```

Output:

Enter Employee Details.

Name:

Sally

Age:

22

Phone number:

9876543210

Address:

ABCD

Salary:

5000

Specialization:

Department:

HR

Enter Manager Details

Name:

ABC

Age:

30

Phone number:

24897685

Address:
xxx
Salary:
8000.
Specilation:
Department:
Manager

Displaying Employee details

Name: Sally
Age : 22
Phone number: 9876543210
Address : ABCD
Salary : 5000
Specialization:
Department : HR.
Salary of the member is 5000
Enter Manage Details:
Name : ABC
Age : 30
Phone numbe: 24897685

Address : xxx

Salary : $5000

Specialization :

Departmet : Manager.

Salary of the member is : $5000