Assignment - 2

Choose the correct option

1) Local variables are stored in an area called ____.

Sol-> (d) Stack

2) Choose the correct option?

```
#include using namespace std;
class Base { };
class Derived : public Base { };
int main()
{ Base *bp = new Derived;
    Derived *dp = new Base;
}
```

Sol-> (c) Compiler Error in line "Derived *dp = new Base"

3) When the inheritance is private, the private methods in base class are _____ in the derived class (in C++).

Sol-> (a) Inaccessible

4) Which of the following is true?

Sol-> (a) The number of times destructor is called depends on Number of objects created.

5) State true or false

Type conversion is automatic whereas type casting is explicit.

Sol-> True

Page No.
Date /

Short answer type question?

1. Explain about new and delete Keywords with code.

Sol:- (a) The new operator denotes a request for memory allocation on the free store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Syntax :-

```
pointer-variable = new data-type;
```

Example :-

```
int *p = new int(25);
float *q = new float(23.45);
```

(b) The delete operator is used to deallocate the memory. User has privilege to deallocate the created pointer variable by this delete operator.

Syntax :-

```
delete pointer_variable;
```

Syntax to delete the block of allocated memory :-

```
delete[] pointer_variable;
```

Example :-

```
delete p;
delete[] q;
```

Date /

2. What are constructors? Why they are required? Explain different types of constructors with suitable example.

Sol-> **Constructor :-**

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.

There are three types of constructors :-

(1) **Default constructors :-** Default constructor which doesn't take any argument. It has no parameters.

Example :-

```
class construct
{public:
    int a, b;
     construct()
    { a = 10;
      b = 20;
    }
};
    int main()
{
  construct c;
  cout << "a : " << c.a << endl;
  cout << "b : " << c.b;
  return 1;
}
```

Date /

② Parameterized constuctors:- It is possible to pass arguments to constructors. Typically, these arguments help initialize an 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 constructor's body use the parameters to initialize the object.

Example:-

```
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()
{ cout << "P1.x = " << p1.getX() << "P1.y = " << p1.getY();
  return 0;
}
```

③ Copy constructor:- A copy constructor is a member function which initializes an object using another object of the same class.

Date /

Example:-

```
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. Explain the difference b/w object oriented and procedural programming language in detail

Sol->

	Object Oriented Programming	Procedural Oriented Programming
(i)	In object oriented programming program is divided into small parts called objects.	(i) In procedural programming, the program is divided into small parts called functions.
(ii)	Object oriented programming follows bottom up approach.	(ii) Procedural programming follows top down approach
(iii)	Object oriented programming have acess specifiers like private, public, protected etc.	(iii) There is no access specifier in procedural programming.

Date

(iv) Object oriented programming provides data hiding so it is more secure.	(iv) Procedural programming does not have any proper way for hiding data so it is less secure.
(v) Overloading is possible in object oriented programming.	(v) In procedural programming overloading is not possible.
(vi) Object oriented programming is based on real world.	(vi) Procedural programming is based on unreal world.
(vii) Examples:- C++, Java, Python etc.	(vii) Examples:- C, FORTRAN, Pascal etc.

Long answer Type question.

a) Explain the type of polymorphism with code.

Sol→ In C++ polymorphism is ~~main~~ mainly divided into two types:-

(i) Compile time polymorphism
(ii) Runtime polymorphism

- Compile time polymorphism:- This type of polymorphism is achieved by function overloading or operator overloading.

→ Function Overloading:- When there are multiple with same name but different parameters the these

Page No.
Date /

functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type arguments.

Example:-

```
#include <iostream>
  using namespace std;
 void print(int i){
 cout << "Here is int" << i << endl;
}
void print(double f){
cout<<"Here is float" << f << endl;
}
void print(char const *c){
  cout << "Here is char*" << c << endl;
}
int main(){
    print(10);
    print(10.10);
    print("ten");
    return 0;
    }
```

→ Operator Overloading :- C++ also provide option to overload operators - for example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.

For example :-

```
#include <iostream>
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();
}
```

2. Runtime polymorphism :- This type of polymorphism is achieved by Function Overriding.

→ Function overriding :- This on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden

Example :-

```
#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;
}
```

Date /

Sol(c)

```
#include <iostream>
#include <string>
using namespace std;
class member {
char name[20], address[40];
double numbe;
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 : " << numbe << 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;
 cout cin.getline(specialization,20);
 cout<<"Department:"<<endl;
 cin.getline(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()
{ coult<<"\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;
cout<<endl;
}
};
int main()
{
 employee e;
 manager m;
 e.input();
 m.input();
 e.display();
 e.printSalary();
 m.display();
 m.printSalary();
}
```