OOP Theory Assignment 1

Jaynam Modi. PG-43. G3. Aug 14, 2020

1. Explain various features of Object Oriented Programming.

> There are three major features in object-oriented programming that makes them different than non-OOP languages: encapsulation, inheritance and polymorphism.

- Encapsulation : Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called "classes," and one instance of a class is an "object".

- Inheritance : Classes are created in hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, only the processing and data associated with that unique step needs to be added.

- Polymorphism : Object-oriented programming allows procedures about

objects to be created whose exact type is not known until runtime.

2. Write a C++ program to find the Factorial of a Number using Default Constructor & Parameterized Constructor.

```
class Factorial{

    public:

    int fact = 1;

    int n = 1;


    Factorial(){

        cout << " > Enter your Number : ";

        cin >> n;

        for(int i = 1; i <= n; i++){

            fact = fact * i;

        }

        cout << " > Factorial : " << fact << endl;
```

```
}

Factorial(int a){

    n = a;

    for(int i = 1; i <= n; i++){

        fact = fact * i;

    }

    cout << " > Factorial : " << fact << endl;

}

Factorial(){

    cout << " > Deleting Object" << endl;

}

};

int main(){
```

```
    Factorial f(5);

    Factorial f2;

}
```

3. Explain the difference between Compile-time Polymorphism & Run-time Polymorphism.

> Compile-time Polymorphism : Function overloading and Operator overloading are perfect example of Compile time polymorphism. For example, we have two functions with same name but different number of arguments. Based on how many parameters we pass during function call determines which function is to be called, this is why it is considered as an example of polymorphism because in different conditions the output is different. Since, the call is determined during compile time thats why it is called compile time polymorphism.

> Run-time Polymorphism : Function overriding is an example of Runtime polymorphism. When child class declares a method, which is already present in the parent class then this is called function overriding, here child class overrides the parent class. In case of function overriding we have two definitions of the same function, one is parent class and

one in child class. The call to the function is determined at runtime to decide which definition of the function is to be called, thats the reason it is called runtime polymorphism.

4. Write a C++ Program for Matrix addition using Operator Overloading.

```
class Matrix{

    int a[3][3];

    public:

    void accept(){

        cout << " > Enter Matrix Elements
        (3 X 3) : " << endl;

        for(int i = 0; i < 3; i++){

            for(int j = 0; j < 3; j++){

                cout << " ";

                cin >> a[i][j];

            }

        }
```

```
}

void display(){
    for(int i = 0; i < 3; i++){
        cout << " ";
        for(int j = 0; j < 3; j++){
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
}

void operator +(Matrix x){
    cout << " > Sum : " << endl;
    int sum[3][3];
```

```
for(int i = 0; i < 3; i++){
    cout << " ";
    for(int j = 0; j < 3; j++){
        sum[i][j] = a[i][j] +
        x.a[i][j];
        cout << sum[i][j] << " ";
    }
    cout << endl;
}
};
int main(){
    Matrix m,n;
    m.accept();
    n.accept();
    cout<<" > First Matrix : " << endl;
```

```
m.display();
cout<<" > Second Matrix : " << endl;
n.display();
m + n;
}
```