Object Oriented Methodology

Week – *1*, Lecture – *2*

Introduction to OOM

SAURABH SRIVASTAVA

VISITING FACULTY

IIIT LUCKNOW

What is Object-Orientated Modelling?

Object-Oriented Modelling refers to an approach towards designing software

The idea is to represent the software in terms of collaborating *objects*

- We will discuss what objects are shortly…

The design documents produced in the process follow some terminology and conventions

Usually, these documents contain one or more diagrams, e.g. Class Diagrams

These documents do not necessarily follow any standards…

- … but they usually communicate enough information to describe the software in context

There are two major reasons for the popularity of Object Oriented Modelling approach

- First, it aligns well with the real world (we will see how)
- Second, Object Orientation support is fairly common with many general-purpose programming langauges

Structured Programming

An alternative to writing Object Oriented code, is structured code

- This is what you essentially did in the previous semester

In this approach, you decouple the data from the operations over the data

The data is stored in multiple variables, often collected together inside a wrapper like a structure

The operations are implemented as functions…

- … which take the required data as parameters, and return the results, if any

There is no concept of “privacy” of data

- If a structure variable is passed to a function, all its member variables are accessible for reading and writing

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

You should be able to understand the behaviour of this program…

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

In particular, observe the `printLastName()` function

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

While it may be doing what is expected from it, i.e., printing the first name from the structure, it is also manipulating the last name…

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

This is a problem with structure variables – we cannot have a tight control over what member variables are accessible where !!

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

Now check some similar C++ code on the right…

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

While the code on the right may not make much sense to you right now, check out the terms "`private`" and "`public`"

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

We will see what these terms mean later, but the takeaway here is that a class – which may look like an extension of a structure to you as of now – has some mechanisms to control the access of its member variables

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

Classes and Objects

A class is a collection of related data items, together with some operations over them

- In contrast, a structure only groups data items, it does not have any operations

A class consists of two types of *members*

- The data items or variables, called *fields*
- The operations or functions that operate over the fields, called *methods*

A class is a “template”, declaring a class doesn’t occupy any memory

An object is an “instantiation” of a class

- If class is a type, an object of the class is a variable of that type
- An object can be seen as a counterpart of the structure variables

The operations are essentially functions, which “belong” to the class

- They can access the fields of the class seamlessly, as if, they are some global variables

Some Examples

Class – *Automobile*

- Fields – Number of Wheels, Fuel Type, Fuel Capacity etc.
- Operations – drive, add fuel, inflate tyres etc.
- Objects – `Volvo B8R`, `Rolls-Royce Phantom`, `Harley-Davidson Iron 883` etc.

Class – *Movie*

- Fields – Name, Release Year, Viewer Certificate etc.
- Operations – stream, download to device, delete from device etc.
- Objects - `The Shawshank Redemption`, `Interstellar`, `Baahubali: The Beginning`

Remember that the object of one class, could be a field for another class

- For example, think about a class called *Streaming Platform*…
- … which could have a field called list of all movies…
- … which in turn, could be an array of *Movie* objects

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

Now, you can see that *Person* is a class, and `player` is an object of the class *Person*.

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct PersonStruct
{
        char fname[20];
        char lname[20];
}Person;

void printLastName(Person p)
{
        strcpy(p.fname, "Ishant");
        printf("%s\n", p.lname);
}

int main()
{
        Person player;
        strcpy(player.fname, "Rohit");
        strcpy(player.lname, "Sharma");
        printLastName(player);
}
```

PersonStruct.c 1,1 All

```cpp
#include<stdio.h>
#include<string.h>

class Person
{
        private:
        char fname[20];
        char lname[20];

        public:
        void setFname(char const* fNameToSet)
        {
                strcpy(fname, fNameToSet);
        }
        void setLname(char const* lNameToSet)
        {
                strcpy(lname, lNameToSet);
        }
        void printLastName()
        {
                printf("%s\n", lname);
        }
};

int main()
{
        Person player;
        player.setFname("Rohit");
        player.setLname("Sharma");
        player.printLastName();
}
```

PersonClass.cpp 1,1 All

Also, fname and lname are fields, whereas setFname(), setLname() and printLastName() are methods that represent operations

Homework !!

Think about more examples around you of classes, objects, fields and operations

- If you get confused, discuss it with your peers…
- … if you are still confused, send me an email !!

Rewrite the program called `StringUtilsWithGlobalString.c` from Week 7 of ITP

- Use a class called `String`
- What will be the field(s)?
- What will be the operation(s)?
- How will the `main()` change?