# LAB 4

# CSE225L

## Template Class & Operator Overloading

In this lab, we will:

- Modify the **dynArr** class to work as a template class so the array elements can be of any type defined by the user.
- Update the header (**.h**) and source (**.cpp**) files to support dynamic memory allocation for any data type.
- Build a complex number class.
- Overload the ***** (multiplication) and **!=** (not equal) operators for the **Complex** class to support complex number arithmetic and comparison.
- Implement necessary methods to handle these operations.

## TEMPLATE CLASS

**Task 1:** Modify the given header (**.h**) and source (**.cpp**) files to implement a **template** class that allows dynamic memory allocation for array elements of any user-defined type.

**dynarr.h**

```
#ifndef DYNARR_H
#define DYNARR_H

class dynArr
{
private:
    int *data;
    int size;

public:
    dynArr();
    dynArr(int);
    ~dynArr();
    void setValue(int, int);
    int getValue(int);
};

#endif // DYNARR_H
```

**dynarr.cpp**

```
#include "dynarr.h"
#include <iostream>
using namespace std;

dynArr::dynArr()
{
    data = NULL;
    size = 0;
}

dynArr::dynArr(int s)
{
    data = new int[s];
    size = s;
}

dynArr::~dynArr()
{
    delete[] data;
}

int dynArr::getValue(int index)
{
    return data[index];
}

void dynArr::setValue(int index, int value)
{
    data[index] = value;
}
```

## OPERATOR OVERLOADING

**Task 2:** Modify the given **Complex** number class to overload the ***** (multiplication) and **!=** (not equal) operators. In the driver file (main.cpp), include the **Complex** class, create two objects, and demonstrate the usage of the ***** and **!=** operators between them.

**complex.h**

```
#ifndef COMPLEX_H
#define COMPLEX_H

class Complex
{
public:
    Complex();
    Complex(double, double);
    Complex operator+(Complex);
    void Print();

private:
    double Real, Imaginary;
};

#endif // COMPLEX_H
```

**complex.cpp**

```
#include "complex.h"
#include <iostream>

using namespace std;

Complex::Complex()
{
    Real = 0;
    Imaginary = 0;
}

Complex::Complex(double r, double i)
{
    Real = r;
    Imaginary = i;
}

Complex Complex::operator+(Complex a)
{
    Complex t;
    t.Real = Real + a.Real;
    t.Imaginary = Imaginary + a.Imaginary;
    return t;
}

void Complex::Print()
{
    cout << Real << endl;
    cout << Imaginary << endl;
}
```