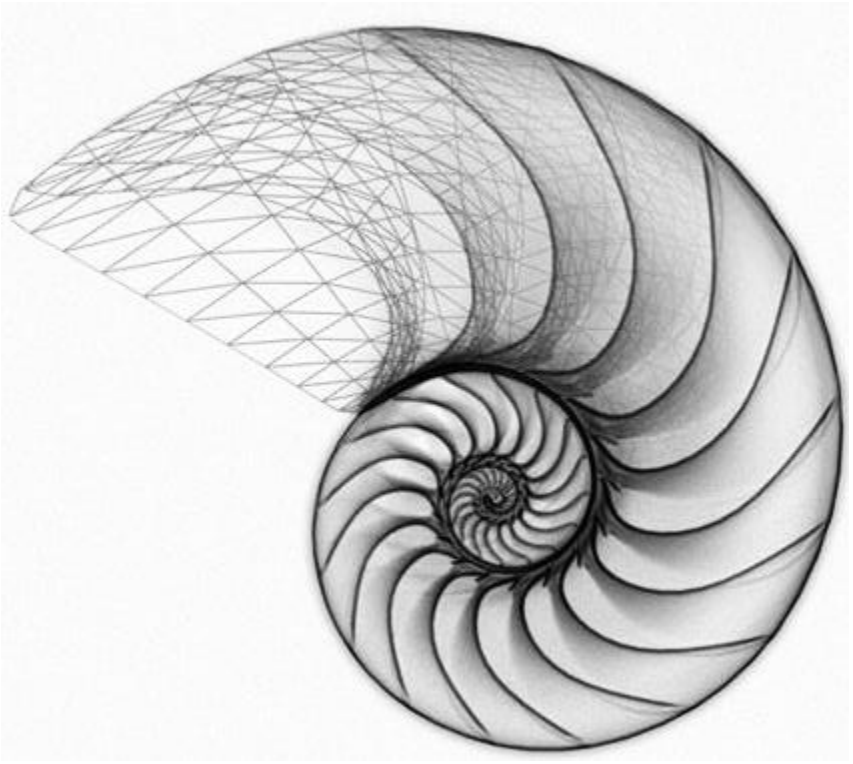

Lecture 04

C++ Primer

CSE225: Data Structures and Algorithms

Basics

- For every data structure, we will create 3 files in our Codeblocks project
  - The declaration file (with .h extension)
  - The definition file (with .cpp extension)
  - The driver file (with .cpp extension)

Basics

dynarr.h (header file)

```
#ifndef DYNARR_H_INCLUDED
#define DYNARR_H_INCLUDED

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

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

#endif // DYNARR_H_INCLUDED
```

Basics

dynarr.cpp (definition file)

```
#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;
}
```

```
void dynArr::allocate(int s)
{
    temp = new int[s];
    for(int i=0; i<s; i++)
       temp[i] = data[i];
    delete []data;
    data = temp;
    temp = NULL;
    size = s;
}

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

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

Basics

main.cpp (driver file)

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

int main()
{
    dynArr d(10);
    int i;

    for(i=0;i<10;i++)
        d.setValue(i,3*i+1);
    for(i=0;i<10;i++)
        cout << d.getValue(i) << endl;
    return 0;
}
```

Template Class

- Now we have a neat class that gives us a 1D dynamic array (you are free to make your own improvisations at home by adding more functions to the class)
  - But it only works for integer type
  - What if we are to make it versatile, so that it works for any type, e.g. float, double and char
  - Should we have separate classes for each type?
    - Write the same code for each type with just minor changes?
    - Instead, we can use template classes

Template Class

```
#ifndef DYNARR_H_INCLUDED
#define DYNARR_H_INCLUDED

template <class T>
class dynArr
{
    private:
    T *data;
    int size;

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

#endif // DYNARR_H_INCLUDED
```

dynarr.h (declaration file)

Template Class

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

template <class T>
dynArr<T>::dynArr()
{
    data = NULL;
    size = 0;
}

template <class T>
dynArr<T>::dynArr(int s)
{
    data = new T[s];
    size = s;
}

template <class T>
dynArr<T>::~dynArr()
{
    delete [] data;
}
```

```
template <class T>
void dynArr<T>::allocate(int s)
{
    data = new T[s];
    size = s;
}

template <class T>
T dynArr<T>::getValue(int index)
{
    return data[index];
}

template <class T>
void dynArr<T>::setValue(int index, T
value)
{
    data[index] = value;
}
```

dynarr.cpp (definition file)

Template Class

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

int main()
{
    dynArr<int> di(10);
    dynArr<double> dd(10);
    int i;

    for(i=0;i<10;i++)
    {
        di.setValue(i,3*i+1);
        dd.setValue(i,7.29*i/1.45);
    }
    for(i=0;i<10;i++)
        cout << di.getValue(i) << " " << dd.getValue(i) <<
endl;

    return 0;
}
```

main.cpp (driver file)