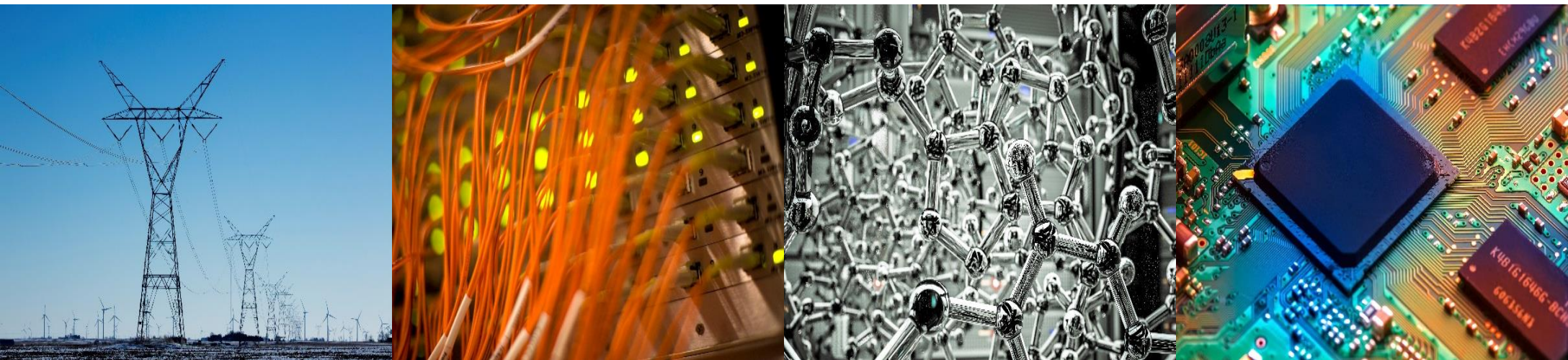# ECE 220 Computer Systems & Programming

Lecture 22 – Introduction to C++: Rule of Three & Template

November 14, 2024

- **Final exam is Wednesday, 12/18, at 7pm; conflict request form is live**

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

What we've learned so far in C++

1. Abstraction

2. Encapsulation

3. Inheritance

4. Polymorphism

- Class vs. Struct

- Dynamic Memory Allocation

- Basic I/O

- Pass by Value vs. Pass by Address (pointer) vs. Pass by Reference

- Function / Operator Overloading

- Base Class & Derived Class

- Virtual Function

- Abstract Class

Rule of Three (Big Three)

```
class demo{
public:
    demo(int x);                //constructor
    demo();                     //default constructor
    demo(const demo &x);     //copy constructor
    ~demo();                     //destructor
    demo &operator=(const demo &x); //copy assignment operator
}
```

- These are called Rule of Three or Big Three (sometimes Big Five)
- When you are implementing one, you should be thinking about the others too

Shallow Copy vs. Deep Copy

```cpp
class Person{
    char *name;
    int id;
  public:
    Person(){};
    Person(const char
*_name, int _id);
    Person(const Person &p);
    void ShowData();
    ~Person();
};
//default copy constructor will be
//inserted if it's not implemented
Person::Person(const Person &p){
    this->name = p.name;
    this->id = p.id;}
```
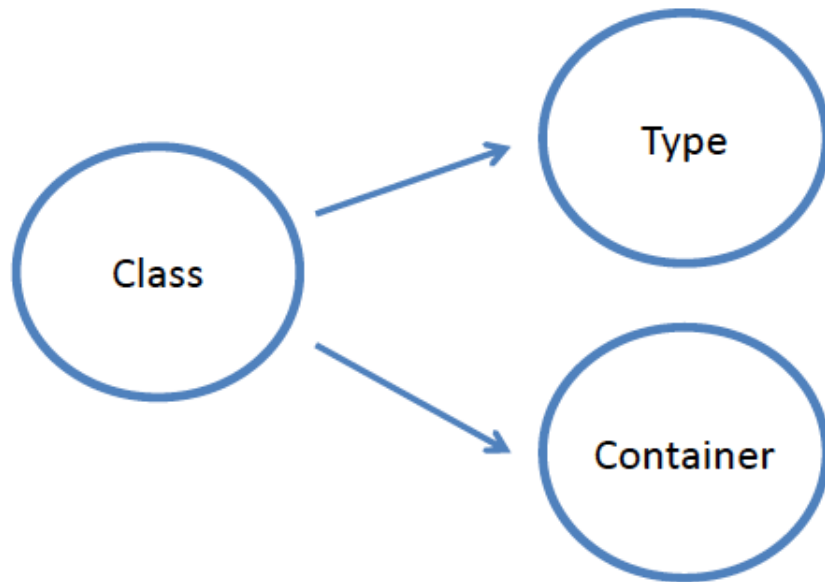
```cpp
Person::Person(const char
*_name, int _id){
    name = new char[48];
    strcpy(this->name, _name);
    this->id = _id;
}

Person::~Person(){
    delete []name;
}
```

```cpp
Person::Person(const Person &p){
    this->name = new char[48];
    strcpy(this->name, p.name);
    this->id = p.id;
}
```

ECE ILLINOIS

Templates

- Separate type from the structure

- Type focuses on data values, basic operators

- Container focuses on data structure (stack, list, array, tree, etc...)

Function Template

```
//functions can have the same names (overload)
int getmin(int a, int b){
    return (a<b)?a:b;
}


double getmin(double a, double b){
    return (a<b)?a:b;
}


//define function with generic type instead!
template <class T>
T getmin(T a, T b){
    return (T)(a<b)?a:b;
}
```

```
int main(){
    cout << getmin(5,7) << endl;
    cout << getmin(1.5,2.7) << endl;
    return 0;
}
```

ECE ILLINOIS

Class Template

```cpp
template <class T>
class mypair {
   T a, b;
   public:
   mypair (T first, T second)
      {a=first; b=second;}
   T getmin ();
};


template <class T>
T mypair<T>::getmin () {
   T retval = a<b?a:b;
   return retval;
}
```

```cpp
int main () {
   mypair <int> myobject (100, 75);
   cout << myobject.getmax() << endl;
   return 0;
}
```

ECE ILLINOIS

Friend Function and Class

- Allows outside function/class to access a class' private and protected members
- "Friendship" is one-way

C++ Standard Template Library (STL) Containers - Vector

- Sequence container that encapsulates **dynamic size arrays** (container - an object that holds the same type of data)
- Can expand or shrink during run-time
- Similar to array, elements are stored in contiguous memory locations

Table 20.1 Some Useful Methods of the C++ STL Vector Class		
	Description	**Example**
`size()`	return size of vector	`intVector.size();`
`resize()`	change size of vector	`intVector.resize(newSize);`
`insert()`	add elements to vector	`intVector.insert(where, value);`
`erase()`	remove elements from vector	`intVector.erase(where);`
`push_back()`	add new element at end of vector	`intVector.pushback(value);`
`front()`	access first element	`y = intVector.front() - 1;`
`back()`	access last element	`intVector.back() = 3;`

Vector Example

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    // Create a vector containing integers
    vector<int> v {1,3,5,7};
    // Add two more integers to vector
    v.push_back(9);
    v.push_back(11);
    // Iterate and print values of vector
    for(int i = 0; i < v.size(); i++) {
        cout << v[i] << endl;
    }
    return 0;
}
```

ECE ILLINOIS

Other STL Containers

List

- Sequence container that encapsulates **a doubly linked list**
- Use when you need to insert and delete elements without moving existing elements
- Elements are individually allocated

Iterators

- Abstract way to represent traversing a collection
- Can be used for reading or writing (depending on iterator type)
- To use them, collection has begin() and end() class methods

List and Iterators Example

```cpp
#include <iostream>
#include <list>
#include <iterator>
using namespace std;

int main(){
    //Create a list containing ints
    list<float> l {2.1, 3.2, 4.5, 3.2, 5.6, 5.6};
    //Remove all elements of list with the value '4.5'
    l.remove(4.5);
    //Remove adjacent duplicate elements of list
    l.unique();
    for(auto it=l.begin(); it!=l.end(); it++){
        cout<<*it<<endl;}
    return 0;
}
```