ECE 220 Computer Systems & Programming

Templates, Iterators, The Big Three, I/O with Objects

ECE ILLINOIS

ILLINOIS

Function Template

```
//functions can have the same names (overload)
int sum(int a, int b){
	return a+b;
}

double sum(double a, double b){
	return a+b;
}


//define function with generic type instead!
template <class Type>
Type sum (Type a, Type b){
	return a+b;
}

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

Class Template

```
template <class T>
class mypair {
    T a, b;
    public:
    mypair (T first, T second)
        {a=first; b=second;}
    T getmax ();
};
template <class T>
T mypair<T>::getmax () {
    T retval;
    retval = a>b? a : b;
    return retval;
}
int main () {
    mypair <int> myobject (100, 75);
    cout << myobject.getmax();
    return 0;
}
```

Linked List Example

```
template <class T>
class List;

//Node Template
template <class T>
class Node{
    public:
    T data;
    //constructor
    Node(T new_data){



    }
    private:
    Node<T> *next;
    friend class List<T>;
};
```

```
int main(){
    List<int> list1;
    list1.add(5);
    list1.add(2);
    list1.add(4);

    List<double> list2;
    list2.add(3.4);
    list2.add(2.8);
    list2.add(1.5);

    return 0;
}
```

```
//List Template
template <class T>
class List{
    public:
    //constructor - init head pointer to NULL
    List(){                                                    }
    //destructor - delete nodes on this list if list isn't empty
    ~List(){




    }
    //member function to add new node to the head of the list
    void add(T new_data){



    }
    private:
    Node<T> *head;
};
```

Iterators

- Abstract way to represent traversing a collection
- Can be used for reading or writing (depending on iterator type)
- Derives from `std::iterator<std::forward_iterator_tag, T>`
- Override default construction, copy-construction, copy-assignment, ==, !=, both forms of ++, and unary * as both lvalue and rvalue
  - operator++() is pre-increment, operator++(int) is post-increment
- To use them, collection has begin() and end() class methods

Deep Class Structure

```
class demo {
public:
    demo(int x);            // constructor
    demo();                 // default constructor
    demo(const demo &x);    // copy constructor
    demo(demo &&);          // move constructor (C++11)
    ~demo();                // destructor
    demo &operator=(const demo &x); // copy assignment
    demo &operator=(demo &&x);
        // move assignment (C++11)
}
```

Deep Class Structure

```
class demo {
public:
    demo(int x);            // constructor
    demo();                 // default constructor
    demo(const demo &x);    // copy constructor
    demo(demo &&);          // move constructor (C++11)
    ~demo();                // destructor
    demo &operator=(const demo &x); // copy assignment
    demo &operator=(demo &&x);
        // move assignment (C++11)
}
```

Overloading operator<< and operator>>

ostream& operator<< (ostream & stream, const List<T> & list);

istream & operator>>(istream &stream, List<T> &list);