# CS180 - End of C++, + Simple linked lists

Note Title 9/11/2013

## Announcements

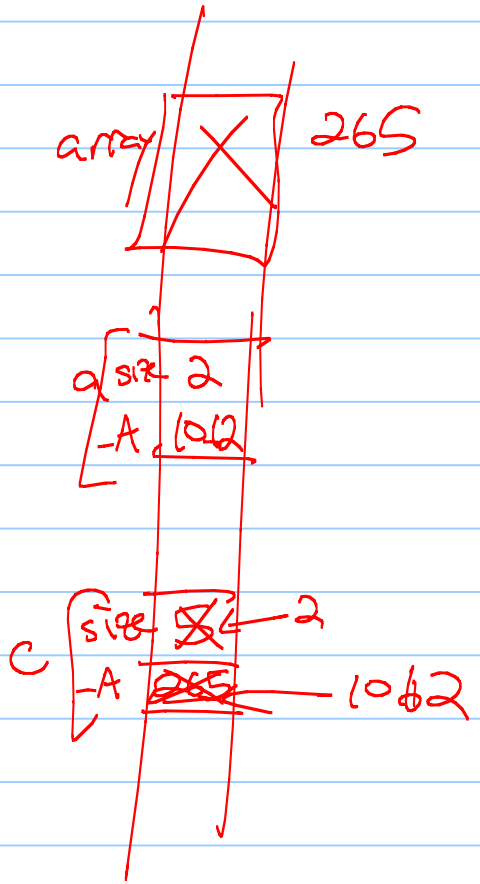
- HW due Monday

Another issue: operator =

```
MyFloatVec c;
c = a;
```

What does this do?

- shallow copy
- memory leak!



c = a = a;  a = b = c;

Solution: rewrite the "=" operation

```
MyFloatVec Operator=(const MyFloatVec& other) {
    if (this != &other) {
        _size = other._size;
        delete[] _A;
        _A = new double[_size];
        for (int i = 0; i < _size; i++)
            _A[i] = other._A[i];
    }
    return *this;
}
```

# Recap: Housekeeping Functions

1. Copy Constructor
2. Operator =
3. Destructor

why? - memory leaks
- deep copies

Enum: user defined types

```
enum Color {RED, BLUE, GREEN};
```

(RED = 0, BLUE = 1, GREEN = 2)

```
Color sky = BLUE;
Color grass = GREEN;

if (sky == BLUE)
    cout << "It's nice out today!" << endl;
```

(sky == 1)

# Structs - simple classes

useful for simple collections of objects

Ex: enum MealType {NO_PREF, VEG, REGULAR, KOSHER};

```
struct Passenger {
    string name;
    MealType mealPref;
    bool isFreqFlyer;
    string freqFlyerNo;
}
```

## Using structs

We can then create instances of a struct in the program:

```
Passenger pass = { "John Smith", VEG, true,
                   "1234" };

pass.mealPref = KOSHER;
```

# More Complex

```
Passenger* p;
p = new Passenger;
p->name = "Barbara Wright";
p->mealPref = REGULAR;
(*p).isFreqFlyer = false;
(*p).freqFlyerNo = "None";
```

# Templates

If we want a function to work for multiple classes – eg int and floats – we can template the variable type.

Ex:

```
template <typename T>
T min(T a, T b) {
    if (a<b)
        return a;
    else
        return b;
}
```

generic type

Important:
Will work for any class with appropriate operators!

Ex:
```
int x = 53;
int y(96);
int z = min(x,y);
```

Line 1;
Line 2;

min(1,2); ← error

```
string a = "Hello";
string b = "Goodbye";
cout << min(a,b) << endl;
```

min(x,a); ← error

# Templates in classes

These work in classes, also.

Important in data structures, so our code will make a list of ints or strings or lists!

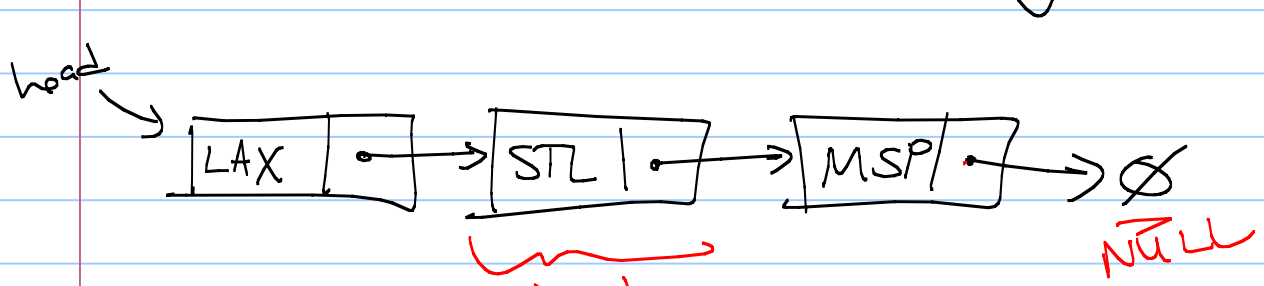
# Using a template:

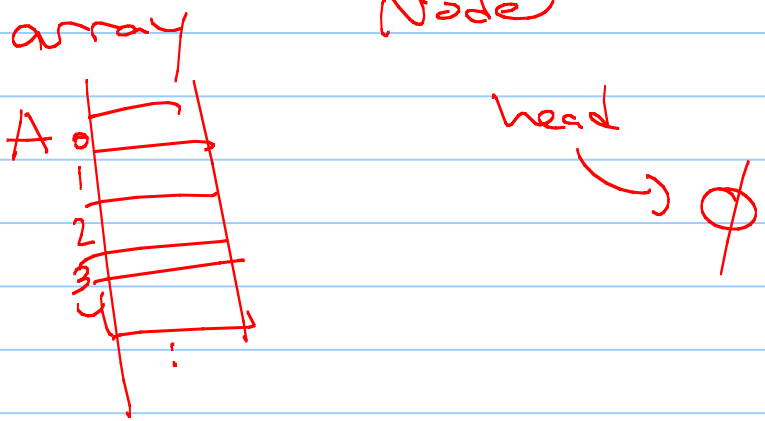
```
MyList <int>    list1;

MyList <string> list2;
```

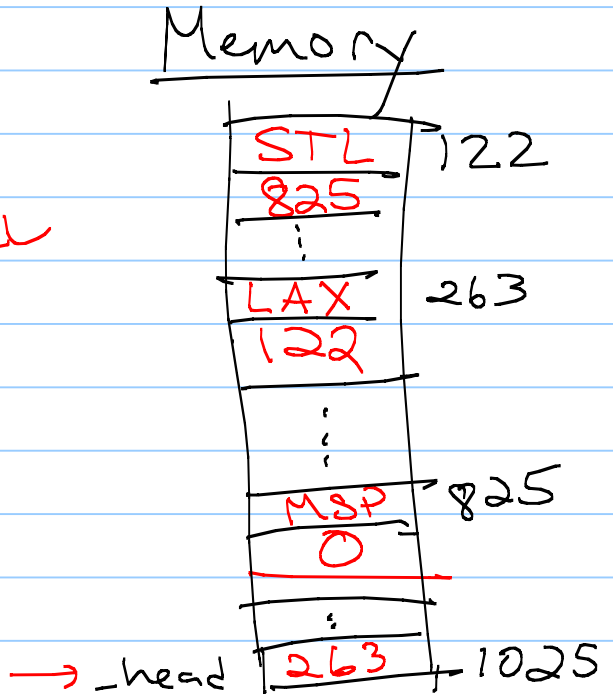
says what T is for this variable instance of class

# Singly Linked Lists

A collection of nodes that together form a linear ordering.



## Why this structure?

Note: This is not the same as the list class which we'll write later.
(nor is it like Python lists)

This linked structure will show up in a lot of our data structures
- similar to arrays as a building block.

So why?
- flexibility - no maximum size

# Implementation

What is a node & how do we code it?
- a separate struct or class

Private data?
- head - pointer to a node
- size (?)

Functions? add to beginning & remove from beginning

Code

```
template <typename Object>
class SLinkedList {

    private:

        class SNode {
            private:
                Object _elem;
                SNode<Object>* _next;
        };

        SNode<Object>* _head;
```

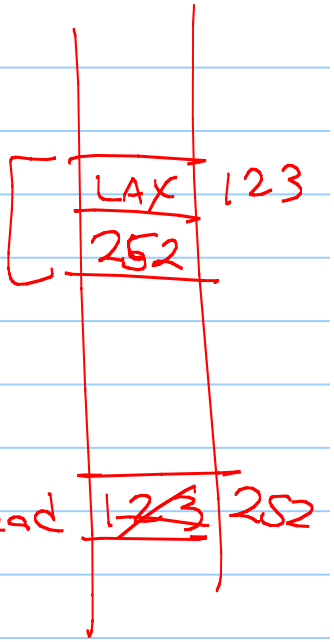
Functions (listed in .h file)

```
public:

    SLinkedList();
   ~SLinkedList();
    bool empty() const;
    const Object& front() const;
    void addFront(const Object& e);
    void removeFront();

};
```

Next:

Let's code it!

(Will post .h + test file on schedule page.)




remove