C++ Templates

Function Swap Two Items

- Swapping two **integers**

```
void swap(int& a, int& b) {
  int temp = a;
  a = b; b = temp;
}
```

- Swapping two **strings**

```
void swap(string& s1, string& s2)
 string temp = a;
  a = b; b = temp;
}
```

- Swapping two instances of a class

```
void swap(TypeT& s1, TypeT& s2)
```

Same Same but Different

- Same for a lot of algorithms like sorting
- Do we have to write the same function for EVERY type?
  - Time Consuming
  - Difficult to modify ALL together
    - E.g. new features, found a bug
- In C++, we can use templates to save our time

C++ Function Templates

```
template<class TypeT>
void bubble(TypeT a[], int n) {
  int i, j;
  for (i = 0; i<n - 1; i++)
    for (j = i + 1; j<n; j++)
      if (a[i]>a[j])
        swap(a[i],a[j]);
}
```

The Same for Class

```
class ListNode
{
private:
        int _item;
        ListNode *_next;

public:
        ListNode(int);
        int content() { return _item; };
        friend class List;
};

class List
{
private:
        int _size;
        ListNode *_head;

public:
        List()
```

- So far, we have constructed a Linked List for integers
- To construct a LL for strings, or other type, we just need to change these

C++ Template with Type : T

Add “template < class T>

int becomes T

ListNode becomes ListNode<T>

```
class ListNode
{
private:
        int _item;
        ListNode *_next;

public:
        ListNode(int);
        int content() { return _item; };
        friend class List;
};

class List
{
private:
        int _size;
        ListNode *_head;

public:
        List()
        {
```

```
template <class T>
class ListNode
{
private:
        T _item;
        ListNode<T> *_next;

public:
        ListNode(T);
        T content() { return _i
        friend class List<T>;
};

template <class T>
class List
{
private:
        int _size;
        ListNode<T> *_head;

public:
```

Same for the Function Bodies

```
void List::insertHead(int n)
{
        ListNode *aNewNode = new ListNode(n);
        aNewNode->_next = _head;
        _head = aNewNode;
        _size++;
};
```

Linked List for Integers

```
template <class T>
void List<T>::insertHead(T n)
{
        ListNode<T> *aNewNode = new ListNode<T>(n);
        aNewNode->_next = _head;
        _head = aNewNode;
        _size++;
};
```

Linked List Template for any type/class

Template File Organization

- However, compiling template files is a big headache
- There are a few methods programmers are using but we will just introduce one that is
  - Easiest to use
  - Reduce errors
  - More compatible with all the platforms

Naming the File

- Let's say you want to implement a template for Linked List
- You can still do the same thing for the header file
- However, there are two changes:
  - Rename the Linked List .cpp template to .hpp (optional)
  - Include the .hpp file in your .h file

MS Visual Studio

- For MSVC users
  - When you add the files to your project, include the .hpp under Headers instead of Sources