Generic Programming

Why Generic function?

```
int main()
{
   const int size = 10;
   int numbers[size];
   for (int i = 0; i < size; ++i) {
      numbers[i] = i+1;
   }
   displayInt(numbers, size);
   string names[] = {"Jim", "Fred", "Jane", "Bob", "Mary",
      "Mike", "Terri", "Allison", "Mason",
      "Meredith"};
   displayStr(names, size);
   return 0;
}
```

Non-generic function

```
void displayInt(int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

void displayStr(string arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

Generic functions

```
template <typename T>
void display(T arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

Non-generic function

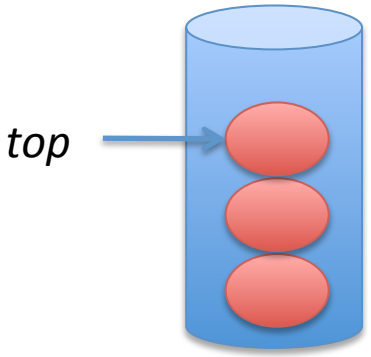
```
void maxInt(int a, int b) {
   return (a>b) ? a : b;
}

void maxDouble(double a, double b) {
     return (a>b) ? a : b;
}

void maxStr(string a, string b) {
     return (a>b) ? a : b;
}
```

Generic functions

```
template <typename T>
T &max(T &a, T &b) {
    return (a>b) ? a : b;
}
```

Non-generic class

```
class Stack {
private:
   int datastore[100];
   int top;
public:
   Stack() { top = -1; }
   void push(int num) {  datastore[++top] = num; }
   int pop() { return datastore[top--]; }
   int peek() {return datastore[top]; }
};
```



Generic class

```
template <typename T>
class Stack {
private:
   T datastore[100];
   int top;
public:
   Stack() { top = -1; }
   void push(T num) {  datastore[++top] = num; }
   T pop() { return datastore[top--];  datastore[top+1] = 0; }
   T peek() {return datastore[top]; }
};
```

- Stack<int>, Stack<double> works
- Stack<string> doesn’t work!

Template specialization

```
template <>
class Stack<string> {
private:
   string datastore[100];
   int top;
public:
   Stack() { top = -1; }
   void push(string num) {  datastore[++top] = num; }
   string pop() { return datastore[top--];  datastore[top+1] = “”; }
   string peek() {return datastore[top]; }
};
```

- Now Stack<string> works

Map

```
int main()
{
    MyMap<string, int> grades;
    grades.insert("Jones", 88);
    grades.insert("Smith", 90);
    cout << grades.find("Smith") << endl;

    MyMap<int, string> numbers;
    numbers.insert(1, "one");
    numbers.insert(2, "two");
    cout << numbers.find(1) << endl;
}
```

"Jones" → 88
"Smith" → 90

1 → "one"
2 → "two"

Generic Class

```
template<class T, class U>
class MyMap {
private:
   vector<T> keys;
   vector<U> values;
public:
   void insert(T key, U value) {
      keys.push_back(key);
      values.push_back(value);
   }
   U find( T key ) { for(int i=0; i<keys.size(); i++)
         if(key == keys[i]) return values[i]; return 0; }
};
```