Week 6


the “i’m going to do later”
project 3
starter pack
are you free to go out tonight?
NETFLIX
ADD TO CART
what are we supposed to be doing again?
YouTube
“I think I’m going to reorganise my room tonight”
TikTok

Announcements

- Midterm 2
- Project 3 is up (start early!)
- HW4
  - due 5/24, good to start early (new parts will appear periodically)

Questions?

- anything?

Generic Programming

Generic functions

```
// without generic programming, I had to define a different swap function
// for each of my 3 use cases below

// int
void swap(int& a, int& b);

int a = 5;
int b = 10;
swap(a, b);

// string
void swap(string& s1, string& s2);

string c = "Yay";
string d = "Long weekend";
swap(c, d);

// custom Class
void swap(MyFakeClass& m1, MyFakeClass& m2);

MyFakeClass o1;
MyFakeClass o2;
swap(o1, o2);

// todo: make this a generic function
// remember, if you template a function, at least one of the arguments
// must be of the template type
```

```
template<typename T>
void swap(T& a, T& b) {
  int temp;

  // the assignment operator needs to be defined for classes that you
  // pass to this function
  temp = a;
  a = b;
  b = temp;
}

// careful about having operators declared when using templates with Classes
```

Syntax Summary

```
// template functions
template<typename T>
T func(T a, T b) {
  // ...
}

// is there a way to have a template with two different types?
template<typename T1, typename T2>
T func(T1 a, T2 b) {
  // ...
}
```

Generic Classes

```
// not covered yet, but just shows you how else you can use templates

// similar to functions, use the prefix at the top
template<typename ItemType>
class HiWhatsUp {
  public:
    void printSomething() {
      cout << m_value;           // remember, the operator<< needs to be defined
                                 // for the class you end up using
    }
    void test123();

  private:
    ItemType m_value:
}
```

```
// ItemType isn't being used in the function,
// but its part of the class so we still need it
template<typename ItemType>
void HiWhatsUp::test123() {
  // ...
}
```

Worksheet

```
/*
Will this code compile? If so, what is the output?
If not, what is preventing it from compiling?

Note: We did not use namespace std because
std has its own implementation of max and namespace std will
thus confuse the compiler.
*/

#include <iostream>

template <typename T>
T max(T x, T y)
{
  // ternary statement
  return (x > y) ? x : y;

  // equivalent to ternary statement
  if (x > y)
    return x;
  else
    return y;
}

// WON'T compile
int main() {
  std::cout << max(3, 7) << std::endl;
  std::cout << max(3.0, 7.0) << std::endl;
  std::cout << max(3, 7.0) << std::endl;     // int and a double, NOT same type
}
```

STL

```
// we've seen these two already
#include <stack>
```

```
#include <queue>
```

Vector

```
// new!
#include <vector>

// vector vs array
// the main difference is that vectors can dynamically grow in size
// with arrays, you're stuck with the size you choose, whether dynamic or
// static allocated

vector<int> v1(3);     // [0 0 0]
vector<int> v2;        // [ ]

v1[0] = 30             // [30 0 0]
v1.push_back(123)      // [30 0 0 123]
cout << v1[3];         // 0

// STL classes have lots of functions and ways to use them, so
// do some Googling to find the docs
v1.front();
v1.back();
v1.pop_back();

v1.size();    // arrays don't have this!

// can do it += n for any integer, not limited to ++ or --
// careful about insert(pos, val)

int* q = v1[0];
v1.insert(4, 456);
*q    // undefined behavior b/c v1 could have moved in memory
```

List

```
// should look familiar from HW
#include <list>    // linked list (doubly linked)

list<float> l;
l.push_front(3.5);
l.push_back(2.2);

// can I do this? no
l[3]
```

```
// limited to ++ or -- to move iterator
// insert(pos, val) shouldn't move things around in memory
```

Iterators

```
vector<int> vec;        // [ ]

vec.push_back(3);       // [ 3 ]
vec.push_back(1);       // [ 3 1 _ ]

// what do these return?
vec.begin(); // points to the 3
vec.end();   // points to spot after the 1


// todo: re-write the loop using an iterator
for (int i = 0; i < vec.size(); i++) {
  cout << vec[i] << " ";
}

for (vector<int>::iterator it = vec.begin(); it != vec.end(); it++) {
  cout << (*it) << " ";
}

// ************************************************** //

// erase
vector<int>:: iterator it1 = vec.begin();

// it1 is invalidated so we create it2
// we'll see how to re-use it so we don't have to keep making new iterators
vector<int>:: iterator it2 = vec.erase(it1)

// ************************************************** //

// what if your stl object is holding a custom class?
class TA {
  public:
    void foo();
  // ...
};

list<TA> l;

list<TA>::iterator it;
it = l.begin();

// todo: call foo function using the iterator
for(list<TA>::iterator it = l.begin(); it != l.end(); it++) {
  it -> foo();
```

```
  (*it).foo();        // . has higher precedence than *, so you need parenthesis
}

// const iterators
// if your STL class is const, use a const_iterator
void takeBreak(const list<TA> &tas) {
  list<TA>::const_iterator it;
}
```

Delete Stuff (Iterator Invalidation)

```
// if you add or delete stuff, your iterators can be invalidated
// b/c under the hood C++ might decide to move your object to a diff spot
// in memory

// todo: are we erasing correctly?
// (hint) does the return type of erase help you in any way?

vector<int> v(3);
v.push_back(1);
v.push_back(2);
v.push_back(3);

for (vector<int>::iterator it = v.begin(); it != v.end(); ) {
  if (*it == 1)
    v.erase(it);             // it = v.erase(it) or else it is invalidated but we
                             // continue to use it
  else
    it++;
}
```

Worksheet

```
// todo: find and fix 3 errors that cause runtime or compile time errors

class Potato {
  public:
    Potato(int in_size) : size(in_size) { }
    int getSize() const {
      return size;
    }
  private:
    int size;
};

int main() {
```

```
  vector<Potato> potatoes;
  Potato p1(3);
  potatoes.push_back(p1);
  potatoes.push_back(Potato(4));
  potatoes.push_back(Potato(5));

  vector<int>::iterator it = potatoes.begin();  // vector<Potato>::iterator it
  while (it != potatoes.end()) {
    potatoes.erase(it);           // it = potatoes.erase(it)
    it++;                         // remove this line so you don't skip Potatoes
  }

  for (it = potatoes.begin(); it != potatoes.end(); it++) {
    cout << it.getSize() << endl;    // (*it).getSize(); or it->getSize();
  }
}
```

Worksheet Problems (from breakout room)

```
// just my implementations for 2 problems, not 100% sure if they're completely
// correct
// but they both showcase "linear" recursive problem solving

int sumOverThreshold(int x[], int length, int threshold) {

  // base case 1 - array of length 0
  if (length < 1)
    return -1;

  // base case 2 - we've reached or exceeded threshold with the first n <= length elements
  if (threshold <= 0)
    return 0

  // recursive call - shrink x and reduce the threshold
  int sum = sumOverThreshold(x+1, length-1, threshold-x[0]);

  // propagate the -1 up if we reach base case 1
  if (sum == -1)
    return -1;

  // otherwise just sum the first n elements on our way back up
  return x[0] + sum;

}

string endX(string str) {
```

```
  // base case - return empty string
  if (str.length() < 1)
    return "";

  // recursion - decrease length of str by 1
  string result = endX(str + 1);

  // if current char is 'x', append to the end
  if (str[0] == 'x')
    return result + 'x';

  // else, prepend the current char
  return str[0] + result;
}
```