C++ REVIEW – POINTERS AND TEST DRIVEN DEVELOPMENT
TOPIC FOR TODAY – C++ POINTERS
POINTERS & MEMORY

• int x = 5;
### POINTERS & MEMORY

- `int x = 5;`
- `int* y = &x;`

The `operator '&'` takes the memory address of an object/variable.
• `int x = 5;`
• `int* y = &x`
• `int* z = y;`
• int x = 5;
• int* y = &x
• int* z = y;
• *z = 0;

operator ** dereferences a pointer in memory to access the underlying object/data
ALLOCATING MEMORY USING NEW

• Point *p = new Point(5, 5);

• new allocates space to hold the object.
• new calls the object’s constructor.
• new returns a pointer to that object.
DEALLOCATING MEMORY USING DELETE

// allocate memory
Point *p = new Point(5, 5);

...

// free the memory
delete p;

• For every call to new, there must be exactly one call to delete.
USING NEW WITH ARRAYS

- `int* nums = new int[10]; // ok`

- Dynamically allocates an array of 10 integers
USING DELETE ON ARRAYS

// allocate memory
int* nums = new int[10];

...)

// free the memory
delete[] nums;

• Have to use delete[].

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```
SYNTACTIC SUGAR “->”

Point *p = new Point(5, 5);

// Access a member function:
(*p).move(10, 10);

// Or more simply:
p->move(10, 10);

Dereference the object then call one of its functions
## STACK VS. HEAP

<table>
<thead>
<tr>
<th>On the Heap / Dynamic allocation</th>
<th>On the Stack / Automatic allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point *p = new Point();</td>
<td>Point p;</td>
</tr>
<tr>
<td>Point *ps = new Point[n];</td>
<td>Point ps[10];</td>
</tr>
</tbody>
</table>

What happens when `p` goes out of scope?
DEVELOPMENT (ONE OUT OF MANY PERSPECTIVES)

1. Solve

2. Implement
   1. Write test
   2. Write code
   3. Repeat

3. Integrate

4. Release
TEST DRIVEN DEVELOPMENT (TDD)
I'm trying to understand test-driven development.

That's easy. First you make a test that fails, then you do the least amount of work possible to make it pass.

So, if I'm going to build a bridge...

...step one would be to drive your car over a cliff.

I don't want to be on your project anymore.

We can discuss that later. Take these keys.
PRACTICAL EXAMPLE

• Lets practice some TDD on the following example

Your project manager at BusyBody Inc says he needs a feature implemented which determines the average amount of time a worker at the company spends at their desk. He says the number of hours each day is already being measured and is stored in an internal array in the code base.
PRACTICAL EXAMPLE

• How do we solve this?

Compute an average of an array!
PRACTICAL EXAMPLE

• First we write a test
  • in other words, set up the scaffolding of the code instead of a function which you don’t know if it works or not – and continue to struggle finding bugs

```cpp
template<typename It>
double sum(It first, It last) {
    return inf; //note this clearly does not work and is thus failing
}

int main() {
    vector<double> arr{0, 1, 1, 2, 3, 5, 8};
    if(sum(arr.begin(), arr.end()) != 20)
        cout << “Test failed?!?!?! I suck!” << endl; //you don’t really suck, its supposed to fail!
}
```
PRACTICAL EXAMPLE

• Before we continue, let's review
  • Positives
    • Scaffolding, function interface, and test all implemented
    • We know it is good design
    • Tests to tell if the code is correct, before we struggle with debugging many lines of code
  • Negatives
    • Code isn’t written until later…..but is that really that bad? NO

• In fact, with TDD you code FASTER and more EFFECTIVELY than without it
PRACTICAL EXAMPLE

• Now the code — and then run the test!

template<typename It>
double sum(It first, It last) {
    double s = 0;
    while(first != last)
        s += *first++;
    return s;
}
THINGS TO REMEMBER

• Always have code that compiles

• Test writing is an art that takes practice (and more learning!)

• Compile and test often!
TESTING FRAMEWORKS

• Many frameworks exist CppUnit, JUnit, etc.

• We will be using a much more simple unit testing framework developed by me
  • If you have downloaded the lab zip for today open it and look there
  • Follows SeTT (Setup, Test, Teardown) – unit testing paradigm
PAIR PROGRAMMING

• Driver – One at the keyboard
• Navigator – Helps direct driver
• Teamwork and communication are key here
• Switch roles frequently!
TDD - EXERCISE

• Download the zip file for this exercise

• Write a c++ function to move an array of integers from one place in memory to another place. Essentially is copy, but delete the original array after copying.

• Pair program!