#2: C++ Review

Our First Class – Library:

Assume we want to build a class with all the data and functionality of a library. What area some member variables and functions a library class should have?

Which of the above are public? Which are private?

Default Constructors and the Rule of Three:

Every class in C++ has a constructor – even if you didn't define one!

Default constructors will call the default constructor of any member variable. Default copy constructors and assignment operators use shallow copying. As a consequence a good rule of thumb is if you are defining one of the following, you should define all of the following:

1.

2.

3.

What is the rule of zero?

Stack Frames

All variables (including parameters to the function) that are part of a function are part of that function's **stack frame**. It is managed by the computer and limited in size. It is 'local' memory.

Heap Memory:

As programmers, we can use heap memory in cases where the lifecycle of the variable exceeds the lifecycle of the function.

1. The only way to create heap memory is with the use of the **new** keyword.

2. The only way to free heap memory is with the use of the **delete** keyword.

3. Memory is never automatically reclaimed, even if it goes out of scope. Any memory lost, but not freed, is considered to be "leaked memory".

Pointers and References – Introduction

A major component of C++ that will be used throughout all of CS 225 is the use of references and pointers.

Often, we will have direct access to our object:

```
Cube c1;      // A variable of type Cube
```

Occasionally, we have a reference or pointer to our data:

```
Cube & s1;  // A reference variable of type Cube
Cube * s1;  // A pointer that points to a Cube
```

Reference Variable

A reference variable is an alias to an existing variable. Modifying the reference variable modifies the variable being aliased. Internally, a reference variable maps to the same memory as the variable being aliased:

main-ref.cpp

```
int main() {
  int i = 7;
  int & j = i;   // j is an alias of i

  j = 4;                            // j and i are both 4.
  std::cout << i << " " << j << std::endl;

  i = 2;                            // j and i are both 2.
  std::cout << i << " " << j << std::endl;
  return 0;
}
```

...run this yourself: run `make` and `./main-ref` in the lecture source code.

Pointers

Unlike reference variables, which alias another variable's memory, pointers are variables with their own memory. Pointers store the memory address of the contents they're "pointing to".

main.cpp

```
int main() {
  cs225::Cube c;
  std::cout << "Address storing `c`:" << &c << std::endl;

  cs225::Cube *ptr = &c;
  std::cout << "Addr. storing ptr: "<< &ptr << std::endl;
  std::cout << "Addr stored in ptr: "<< ptr << std::endl;
  std::cout << "Value at address: "<< *ptr << std::endl;

  return 0;
}
```

In the above example, what is the reference operator? What does it do?

In the above example, what is the dereference operator? What does it do?

Templates in C++

Templates are a way to write generic code whose actual type is determined during compilation, potentially for multiple data types.

If it helps you conceptually, think of it as a function which takes a data type as an input parameter. This is not strictly true but compilers are outside the scope of 225!

Templated Functions:

functionTemplate1.cpp

```

template <typename T>
T maximum(T a, T b) {
  T result;
  result = (a > b) ? a : b;
  return result;
}
```