Polymorphism

Virtual Functions:

* Virtual functions provide polymorphism in C++
  + Regular function that has “virtual” before it
* Class with virtual functions in it are polymorphic class
* Virtual functions are declared in base class, NOT derived classes
  + Defined in base class and REDEFINED in derived class (overwriting functions)
* Keyword virtual only needed in base class
  + Don’t need to re-declare as virtual in derived, if not redefined, it will still work the same as the base class

Pointers

* Don’t hold data, point to data
* **An object that contains a memory address**
* Anything with an address in the memory can have a pointer: int, double, string, char, float, classes, objects of classes.
* Declaring a pointer:
  + Type \*var-name;
  + Setting it = to something will access the memory allocation instead of value
  + Type: double, int, string etc… it states what it is pointing to, the first byte of data and then follows to get all of the remaining bytes for the data type
    - Type is for the variable it points to
* var-name is the name of the pointer variable
* ex: int \*b 🡪 b is a pointer which can hold the memory address of an integer variable
* Getting the memory address of a variable
  + Use & operator
  + & and \* are unary operators
* Act of using pointers is indirection
* 3 uses of \*
  + Declare a pointer
  + Get the value from a data address
  + Multiply
* 32- bit environment:
  + Char = 1 byte
  + Int = 3 byte
  + Float = 4 bye
  + Double = 8 byte
* Only four arithmetic operations that can be used on a pointer: ++, --, +, or –
  + Cannot multiple or divide addresses, but can add/subtract/increment
  + Increments based on the type and system
    - If \*b = 200 and increment by 1, an int in 32-bit would go to 204, to 208 in a 64-bit
    - After incrementing up, it is no longer pointing to what was at 200.
* Character arrays become strings when incremented (or something)

Null Pointers

* Float \*p = 0
  + Setting = to 0 makes it point to nothing, it’s a safety measure

Multiple Indirection

* Using a normal pointer = single indirection, holding the value with an address
* Using a pointer to hold a pointer that holds a value = multiple indirection
* Ex) floe p, \*r. \*\*q;
  + q is the double pointer that can point to r and then whatever r is pointing to
  + p = 3.14; r = &p; q = &r;
  + to get the value of p from q: cout << \*\*q;

Passing Arrays and Pointers to Functions

* Same as normal, just indicate the parameters of the function as pointers
  + Could also just pass the addresses to the function to run the function

Pointers with Function

* Example int \*Function(){ }

Pointers with classes

* Use an arrow to assign pointer values
  + Temp \*t;
  + T -> tem = 100;
  + (Look at point\_obj file)