CS 225

#3: C++ Review

January 24, 2022 · *G Carl Evans*

Pointers and References

Often, we will have direct access to our object:

```
Cube s1;  // A variable of type Cube
```

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

```
Cube & r1 = s1; // A reference variable of type Cube
Cube * p1;  // A pointer that points to a Cube
```

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".

Three things to remember on pointers:

1.

2.

3.

Indirection Operators:

```
&v

*v

v->
```

main.cpp

```
 4  int main() {
 5    cs225::Cube c;
 6    std::cout << "Address storing `c`:" << &c << std::endl;
 7
 8    cs225::Cube *ptr = &c;
 9    std::cout << "Addr. storing ptr: "<< &ptr << std::endl;
10    std::cout << "Contents of ptr: "<< ptr << std::endl;
11
12    return 0;
13  }
```

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. Using **new** will:
   -
   -
   -

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

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".

Heap Memory – Allocating Arrays

heap-puzzle3.cpp

```
 5  int *x;
 6  int size = 3;
 7
 8  x = new int[size];
 9
10  for (int i = 0; i < size; i++) {
11    x[i] = i + 3;
12  }
13
14  delete[] x;
```

*: **new[]** and **delete[]** are identical to **new** and **delete**, except the constructor/destructor are called on each object in the array.

Memory Lifecycle

-Stack

-Heap

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. Three key ideas:

1.

2.

3.

reference.cpp

```
 3  int main() {
 4    int i = 7;
 5    int & j = i;   // j is an alias of i
 6
 7    j = 4;                          // j and i are both 4.
 8    std::cout << i << " " << j << std::endl;
 9
10    i = 2;                          // j and i are both 2.
11    std::cout << i << " " << j << std::endl;
12    return 0;
13  }
```

---

heap-puzzle1.cpp

```
 6  int *x = new int;
 7  int &y = *x;
 8
 9  y = 4;
10
11  cout << &x << endl;
12  cout << x << endl;
13  cout << *x << endl;
14
15  cout << &y << endl;
16  cout << y << endl;
17  cout << *y << endl;
```

heap-puzzle2.cpp

```
 6  int *p, *q;
 7  p = new int;
 8  q = p;
 9  *q = 8;
10  cout << *p << endl;
11
12  q = new int;
13  *q = 9;
14  cout << *p << endl;
15  cout << *q << endl;
```

Memory and Function Calls

Suppose we want to join two Cubes together:

joinCubes-byValue.cpp

```
11  /*
12   * Creates a new Cube that contains the exact volume
13   * of the volume of the two input Cubes.
14   */
15  Cube joinCubes(Cube c1, Cube c2) {
16    double totalVolume = c1.getVolume() + c2.getVolume();
17
18    double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20    Cube result(newLength);
21    return result;
22  }
```

By default, arguments are "passed by value" to a function. This means that:

-
-

CS 225 – Things To Be Doing:

1. Finish Setting up VM
2. Join CampusWire