

#### #3: Memory

August 31, 2018 · Wade Fagen-Ulmschneider

### **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 & s1; // A reference variable of type Cube
Cube * s1; // 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.

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

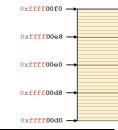
## **Indirection Operators:**

&v

\*v

v->

# **Stack Memory:**



```
example1.cpp

int main() {
   int a;
   int b = -3;
   int c = 12345;

   int *p = &b;
   return 0;
   }
}
```

Location	Value	Type	Name
0xffff00f0 →	Value	Турс	Tunic
0xffff00e8 →			
0xffff00e0 →			
0xffff00d8 →			
0xffff00d0 →			

```
example2.cpp

3 int main() {
    cs225::Cube c;
    cs225::Cube *p = &c;
    return 0;
    }
}
```

Location	Value	Type	Name
0xffff00f0 →		1340	Tunic
0xffff00e8 →			
0xffff00e0 →			
0xfffff00d8 →			

#### **Stack Frames**

All variables (including parameters to the function) that are part of a function are part of that function's **stack frame**. A stack frame:

1.

2.

	stackframe.cpp					
1	1 int hello() {			<pre>int main() {</pre>		
2	2 int a = 100;			int a;		
3	return a;	;	8	int $b = -3;$		
4	4 }		9	<pre>int c = hello();</pre>		
5	5		10	int d = 42;		
			11			
			12	return 0;		
			13	}		
I	Location	Value	Т	ype	Name	
0x	ffff00f0 →					
0х	fffff00e8 →					
	affff00e0 →					
	fffff00d8 →					
0×	ffff00d0 →					

# Puzzle: What happens here?

```
puzzle.cpp
    Cube *CreateCube() {
5
     Cube c(20);
6
     return &c;
7
   }
8
   int main() {
10
     Cube *c = CreateCube();
11
     double r = c->getVolume();
12
     double v = c->getSurfaceArea();
13
     return 0;
14
```

## **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".

```
heap1.cpp

4 int main() {
5 int *p = new int;
6 Cube *c = new Cube(10);
7
8 return 0;
9 }
```

Stack	Value	Hea	ap	Value
0xffff00f0 →	Varue	0x42	2020 →	varue
0xfffff00e8 →			2018 <del>&gt;</del>	
0xfffff00e0 →		0x4	2010 →	
0xfffff00d8 →			2008 →	
0xfffff00d0 →		0 <b>x</b> 42	2000 →	

CS 225: TTBD → lab\_intro due Sunday; MP1 released and due Sept. 10