CS 225

#3: Memory

January 18, 2019 · *Zilles, 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

```
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 }
```

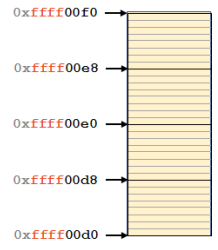
Indirection Operators:

&v

***v**

v->

Stack Memory:



example1.cpp

```
1  int main() {
2    int a;
3    int b = -3;
4    int c = 12345;
5
6    int *p = &b;
7
8    return 0;
9  }
```

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

example2.cpp

```
3  int main() {
4    cs225::Cube c;
5    cs225::Cube *p = &c;
6
7    return 0;
8  }
```

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

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  int hello() {
2    int a = 100;
3    return a;
4  }
5
```

```
6  int main() {
7    int a;
8    int b = -3;
9    int c = hello();
10   int d = 42;
11
12   return 0;
13 }
```

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

Puzzle: What happens here?

puzzle.cpp

```
4  Cube *CreateCube() {
5    Cube c(20);
6    return &c;
7  }
8
9  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
0xffff00f0 →	
0xffff00e8 →	
0xffff00e0 →	
0xffff00d8 →	
0xffff00d0 →	

Heap	Value
0x42020 →	
0x42018 →	
0x42010 →	
0x42008 →	
0x42000 →	

CS 225: TTBD → lab_intro due Sunday; MP1 released and due Jan. 28th