## ju

## Thanks to all of you who pointed out typos in the notes, and gave formatting suggestions:)

## But the google doc seems auto-reject comments from time to time - please lmk if you ran into this!

### **Variable / reference variables / pointer**

|  |  |
| --- | --- |
| 1  23 | Cube s1; // a variable containing a Cube object  Cube & s2; // a reference to a variable of type Cube  Cube \* s3; // a pointer to a variable of type Cube |

#### **Pointers**

* + Stores a memory address of the instance instead of storing data
  + Must resolve the memory address to access the data
  + Pointers are extremely powerful and extremely dangerous

|  |  |
| --- | --- |
| main.cpp | |
| 1  2  3  4  5  6  7  8 | 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 << "Contents of ptr: "<< ptr << std::endl;  } |

#### 

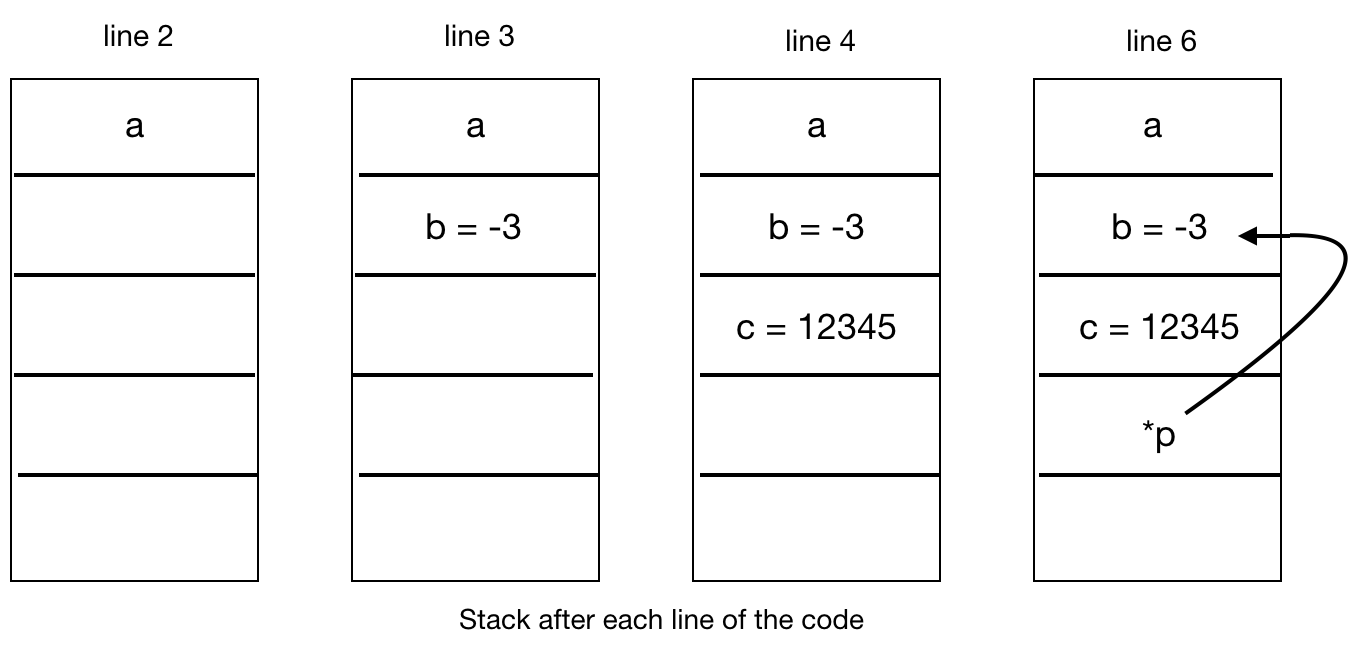
#### **Indirection operators**

* + - **&c** - returns the memory address of c’s data. We can say that & operator takes us one step away from the data.
    - **\*ptr** - returns the data at the memory address contained at ptr, aka dereferencing a pointer. We say that \* operator takes us one step closer to the data.
  + **ptr->**
    - (\*ptr).getVolume() = ptr->getVolume()

#### **Stack Memory**

* + The default type memory
  + Starts near the top of memory (but it does not necessarily start with the most top piece memory)
  + Starts at a high address and grows towards 0
  + The data is readhorizontal line from low to high (the data is read up).
  + All variables are by default on stack (automatic variables)
  + Function sizeof() : returns the size of a type in bytes
    - Int variable takes 4 bytes
    - Pointer takes 8 bytes

|  |  |  |
| --- | --- | --- |
| example1.cpp | | example2.cpp |
| 1  2  3  4  5  6  7  8 | int main() {  int a;  int b = -3;  int c = 12345;  int \*p = &b;  return 0;  } | int main() {  cs225::Cube c;  cs225::Cube \*p =&c;  std::cout << "&c: " << &c << std::endl;  std::cout << "&p: " << &p << std::endl;  return 0;  } |





* (&p) = 0xffff00e0

### **Stack Frames**

Each function invocation gets a stack frame.

* + A stack frame is created whenever a function is called.
  + A stack frame is reclaimed when a function returns, and automatically marked free (not actually freed) . When memory is marked free, it can be overwritten. (We **never** want to return a pointer to a stack variable)

|  |  |  |  |
| --- | --- | --- | --- |
| stackframe.cpp | | | |
| 1  2  3  4  5 | int hello() {  int a = 100; //automatic variables  return a;  } | 6  7  8  9  10  11  12 | int main() {  int a;  int b = -3;  int c = hello();  int d = 42;  return 0;  } |

