



CS 225

Data Structures

August 30 – Memory

G Carl Evans

Pointers and References

A variable containing an instance of an object:

```
1 Cube s1;
```

A reference variable of a Cube object:

```
1 Cube & r1 = s1;
```

A variable containing a pointer to a Cube object:

```
1 Cube * p1;
```



Pointers

Three key ideas:

1.

2.

3.

main.cpp

```
1 #include <iostream>
2 #include "Cube.h"
3
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

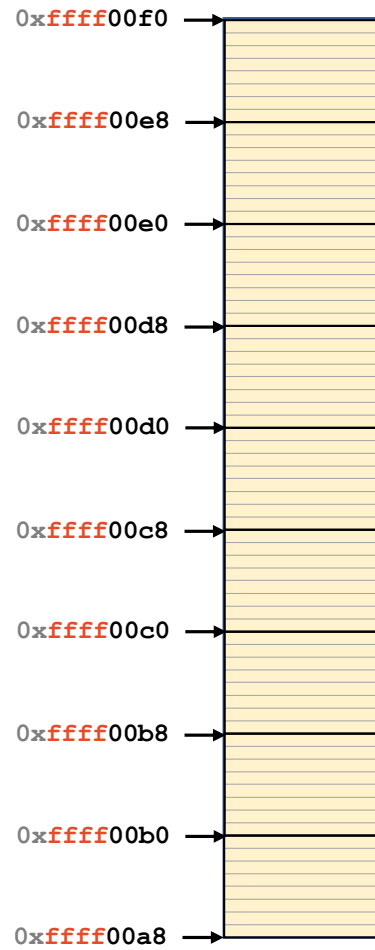
Given any variable **v**:

&v

***v**

v->

Stack Memory





<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

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 }
```



```
1 #include <iostream>
```

```
2
```

```
3 int main() {
```

```
4     std::cout << sizeof(int) << std::endl;
```

```
5     return 0;
```

```
6 }
```

sizeof-int.cpp



```
1 #include <iostream>
```

```
2
```

```
3 int main() {
```

```
4     std::cout << sizeof(int *) << std::endl;
```

```
5     return 0;
```

```
6 }
```

sizeof-intptr.cpp

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 }
```

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0x7ffe2ee87228			
0x7ffe2ee87220			
0x7ffe2ee87218			
0x7ffe2ee87210			
0x7ffe2ee87208			
0x7ffe2ee87200			
0x7ffe2ee871f8			
0x7ffe2ee871f0			
0x7ffe2ee871e8			
0x7ffe2ee871e0			

Real results when running on linus.ews.illinois.edu

```
&a: 0x7ffe2ee87218  
&b: 0x7ffe2ee87214  
&c: 0x7ffe2ee87210  
&p: 0x7ffe2ee87208
```



<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

example2.cpp

```
1 #include "Cube.h"
2
3 int main() {
4     cs225::Cube c;
5     cs225::Cube *p = &c;
6
7     return 0;
8 }
9
```



```
1 #include <iostream>
```

sizeof-cube.cpp

```
2 #include "Cube.h"
```

```
3
```

```
4 int main() {
```

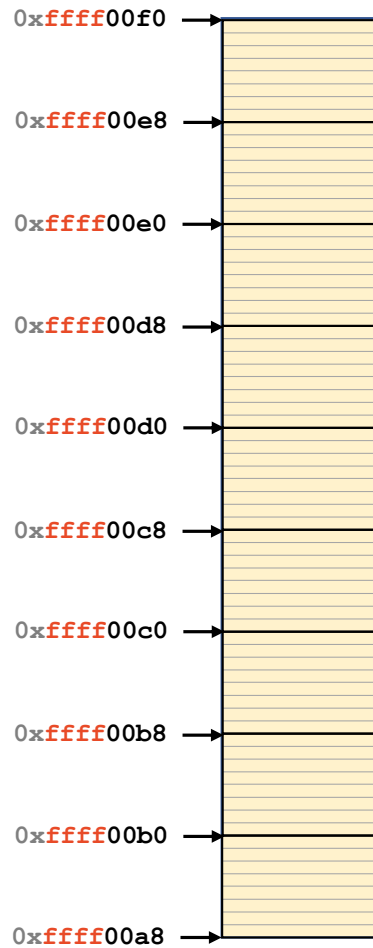
```
5     std::cout << sizeof(cs225::Cube) << std::endl;
```

```
6     std::cout << sizeof(cs225::Cube *) << std::endl;
```

```
7     return 0;
```

```
8 }
```

Stack Frames



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 }
```



Problems of the Day (POTD)

POTDs are small, daily problems for you to practice programming in an environment similar to the CBTF exam environment.

Each POTD is worth **+1** extra credit point, capped at **+40**.
(Course-wide, all extra credit is capped at +100.)

POTD#1 is available on Tuesday, until 8:00am Wednesday morning when POTD#2 becomes available!

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

```

1 #include "Cube.h"
2 using cs225::Cube;
3
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 }

```

puzzle.cpp

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

```

1 #include "Cube.h"
2 using cs225::Cube;
3
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 }

```

puzzle.cpp

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffee00f0			
0xffee00e8			
0xffee00e0			
0xffee00d8			
0xffee00d0			

```

1 #include "Cube.h"
2 using cs225::Cube;
3
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 }

```

puzzle.cpp

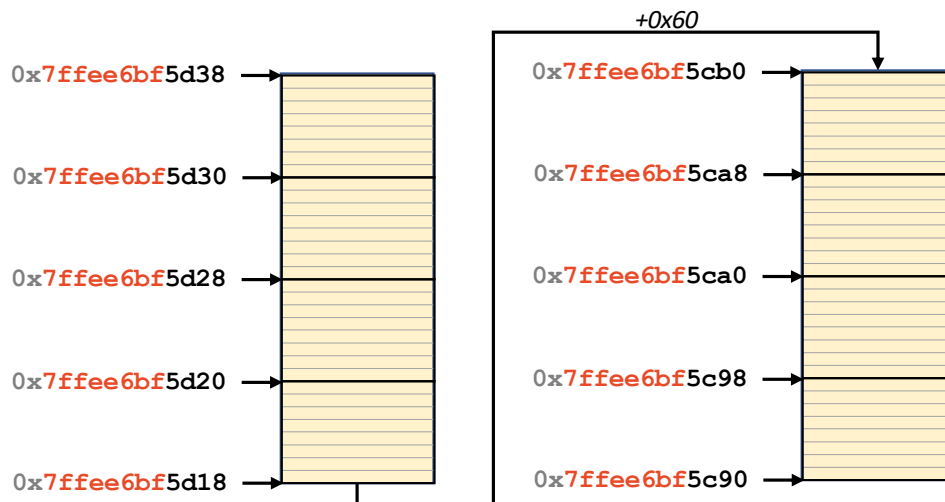


What happens on a real system?

```

13 int main() {
14     Cube *c = CreateCube();
15     cout << c->getVolume() << endl;
16     cout << "c->getVolume(): " << c->getVolume() << endl;
17     cout << "&c (main): " << &c << endl;
18     cout << " c (main): " << c << endl;
19     double r = c->getVolume();
20     cout << "&r (main): " << &c << endl;
21     cout << " r (main): " << c << endl;
22     double v = c->getSurfaceArea();
23     cout << "&v (main): " << &c << endl;
24     cout << " v (main): " << c << endl;
25     return 0;

```



Real results when running on linus.ews.illinois.edu

```

&c (CreateCube): 0x7ffee6bf5ca8
8000
c->getVolume(): 2.07941e-317


&c (main): 0x7ffee6bf5d30
 c (main): 0x7ffee6bf5ca8

&r (main): 0x7ffee6bf5d28
 r (main): 6.95312e-310

&v (main): 0x7ffee6bf5d20
 v (main): 0

```

```
13 int main() {
14     Cube *c = CreateCube();
15     cout << c->getVolume() << endl;
16     cout << "c->getVolume(): " << c->getVolume() << endl;
17
18
19
20
21
22
23
24
25
```

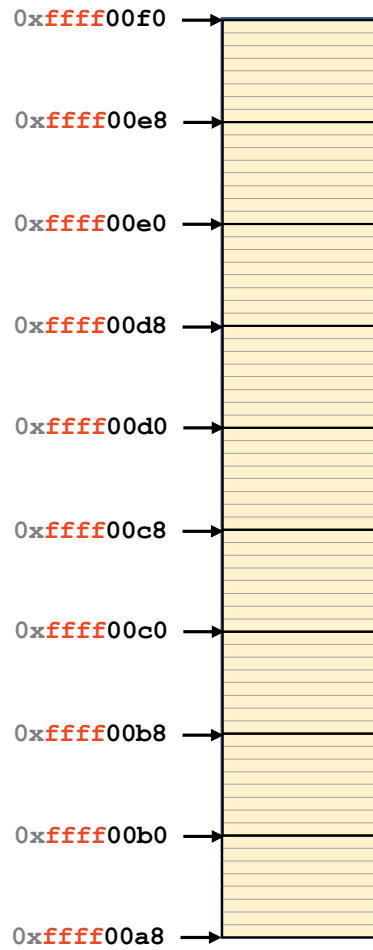


Real results when running on linus.ews.illinois.edu
&c (CreateCube): 0x7ffee6bf5ca8
8000
c->getVolume(): 2.07941e-317

Stack Memory

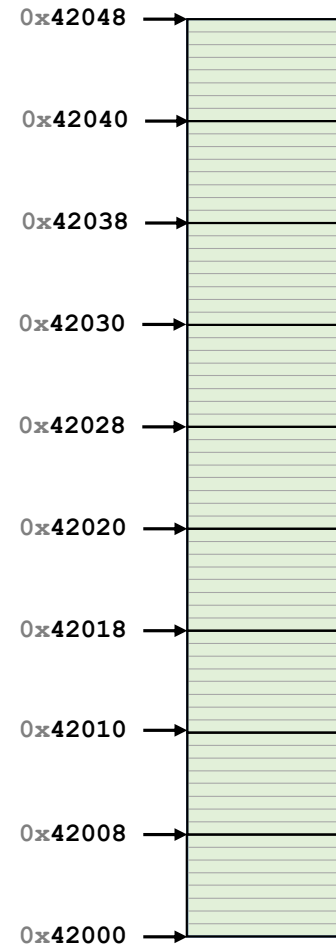
vs.

Heap Memory





Heap Memory





Heap Memory - new

As programmers, we can use heap memory in cases where the lifecycle of the variable exceeds the lifecycle of the function.

The only way to create heap memory is with the use of the **new** keyword. Using **new** will:

- 1.

- 2.

- 3.



Heap Memory - delete

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

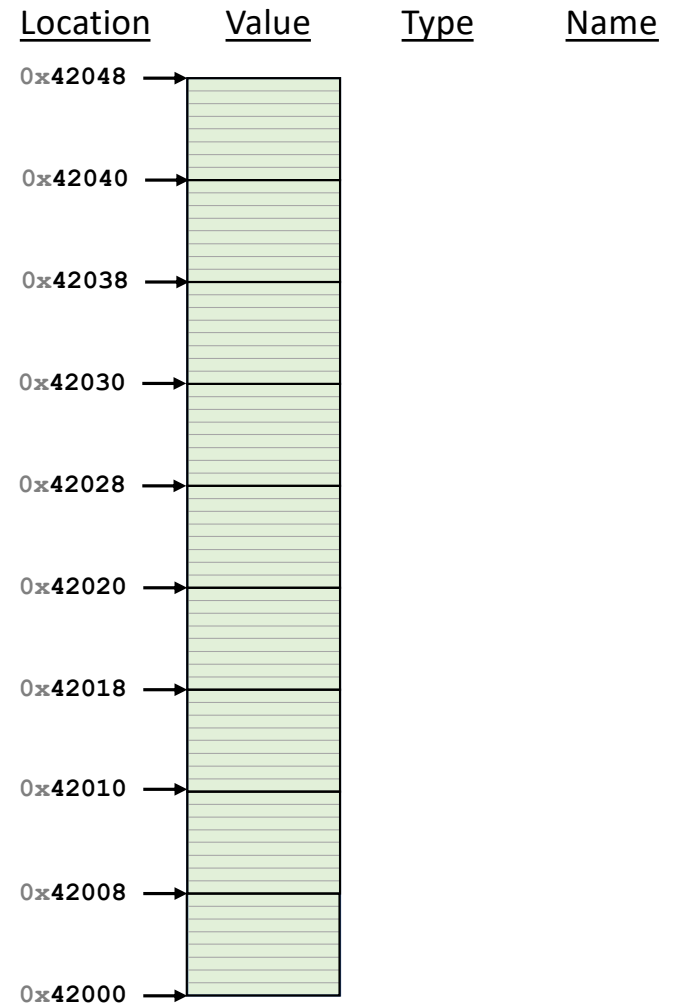
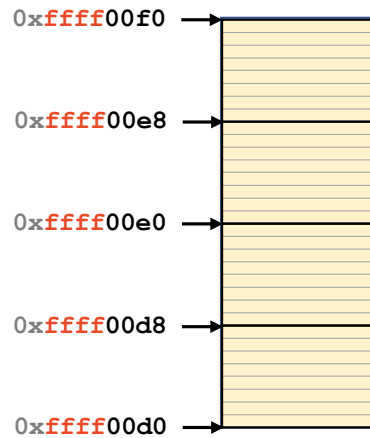
```
1 #include "Cube.h"
2 using cs225::Cube;
3
4 int main() {
5     int *p = new int;
6     Cube *c = new Cube(10);
7
8     return 0;
9 }
```

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0x42048			
0x42040			
0x42038			
0x42030			
0x42028			
0x42020			
0x42018			
0x42010			
0x42008			
0x42000			

heap2.cpp

```
1 #include "Cube.h"
2 using cs225::Cube;
3
4 int main() {
5     Cube *c1 = new Cube();
6     Cube *c2 = c1;
7
8     c2->setLength( 10 );
9
10    return 0;
11 }
```



extra-puzzle1.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int *p;
6     int x;
7
8     p = &x;
9     x = 6;
10
11     cout << x << endl;
12     cout << p << endl;
13
14     return 0;
15 }
```

extra-puzzle2.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int *p, *q;
6     p = new int;
7     q = p;
8     *q = 8;
9     cout << *p << endl;
10
11     q = new int;
12     *q = 9;
13     cout << *p << endl;
14     cout << *q << endl;
15
16     return 0;
17 }
```