CS 225

Data Structures

January 29 – Memory
G Carl Evans

Cube.h

```
1   #pragma once
2
3   namespace cs225 {
4     class Cube {
5       public:
6         Cube(double length);
7         double getVolume();
8         double getSurfaceArea();
9
10
11
12
13
14
15
16
17  }
18
19
20
```

Cube.cpp

```
1   #include "Cube.h"
2   namespace cs225 {
3     Cube::Cube(double length) {
4       length_ = length;
5
6     }
7
8     double Cube::getVolume() {
```

puzzle.cpp

```
1   #include "Cube.h"
2   using cs225::Cube;
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   int main() {
8     Cube c;
9     cout << "Volume: " << c.getVolume() << endl;
10    return 0;
11  }
```

Hate Typing cout:: and cs225::?

Useful Shortcut:

```
using std::cout;    // Imports cout into global scope
using std::endl;    // Imports endl into global scope
using cs225::Cube;  // Imports Cube into global scope
```

Strongly Discouraged Shortcut

```
using namespace std;  // Imports all of std:: into
                      // global scope!
                      // ...THOUSANDS of things!
```

Cube.h

```
1   #pragma once
2
3   namespace cs225 {
4     class Cube {
5       public:
6
7         Cube(double length);
8         double getVolume();
9         double getSurfaceArea();
10
11
12
13
14
15
16
17
18  }
19
20
```

Cube.cpp

```
1   #include "Cube.h"
2   namespace cs225 {
3
4
5
6
7     Cube::Cube(double length) {
8       length_ = length;
9     }
10
11    double Cube::getVolume() {
```

puzzle.cpp

```
7   int main() {
8     Cube c;
9     cout << "Volume: " << c.getVolume() << endl;
10    return 0;
11  }
```

Cube.h

```
1   #pragma once
2
3   namespace cs225 {
4     class Cube {
5       public:
6
7         Cube(double length);
8         double getVolume();
9         double getSurfaceArea();
10
11
12
13
14
15
16
17
18  }
19
20
```

Cube.cpp

```
1   #include "Cube.h"
2   namespace cs225 {
3
4
5
6
7     Cube::Cube(double length) {
8       length_ = length;
9     }
10
11    double Cube::getVolume() {
```

puzzle.cpp

```
7   int main() {
8     Cube c;
9     cout << "Volume: " << c.getVolume() << endl;
10    return 0;
11  }
```

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.

Indirection Operators

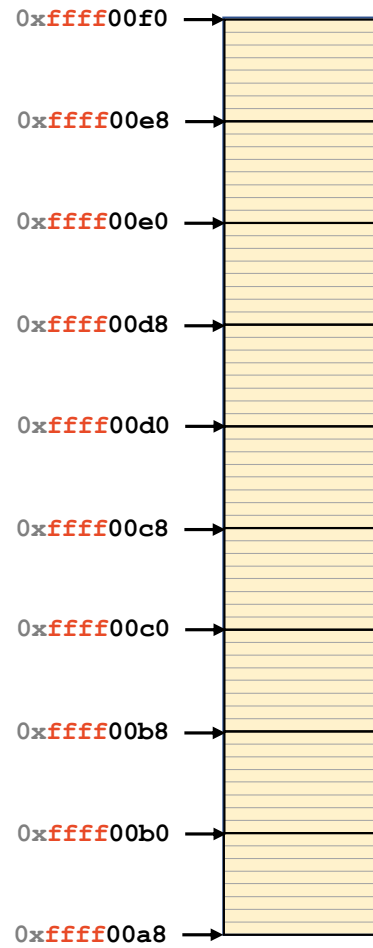**Given any variable v:**

&v

*v

v->

main.cpp

```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 }
```

Stack Memory

0xffff00f0	
0xffff00e8	
0xffff00e0	
0xffff00d8	
0xffff00d0	
0xffff00c8	
0xffff00c0	
0xffff00b8	
0xffff00b0	
0xffff00a8	

Location	Value	Type	Name

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

```cpp
#include <iostream>

int main() {
  std::cout << sizeof(int *) << std::endl;
  return 0;
}
```

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	`}`

Location	Value	Type	Name
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
```

example2.cpp

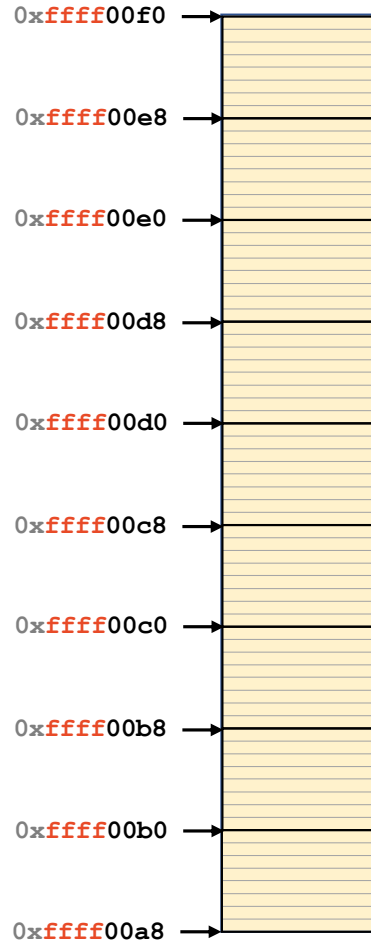|	Location	Value	Type	Name
	0x**ffff**00f0			
	0x**ffff**00e8			
	0x**ffff**00e0			
	0x**ffff**00d8			
	0x**ffff**00d0			
	0x**ffff**00c8			
	0x**ffff**00c0			
	0x**ffff**00b8			
	0x**ffff**00b0			
	0x**ffff**00a8			

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

```cpp
#include <iostream>
#include "Cube.h"

int main() {
  std::cout << sizeof(cs225::Cube) << std::endl;
  std::cout << sizeof(cs225::Cube *) << std::endl;
  return 0;
}
```

sizeof-cube.cpp

Stack Frames

stackframe.cpp

```cpp
int hello() {
    int a = 100;
    return a;
}

int main() {
    int a;
    int b = -3;
    int c = hello();
    int d = 42;

    return 0;
}
```

0xffff00f0 →

0xffff00e8 →

0xffff00e0 →

0xffff00d8 →

0xffff00d0 →

0xffff00c8 →

0xffff00c0 →

0xffff00b8 →

0xffff00b0 →

0xffff00a8 →

Location	Value	Type	Name
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

puzzle.cpp

```cpp
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 }
```

Location	Value	Type	Name
0xffff00f0			
0xffff00e8			
0xffff00e0			
0xffff00d8			
0xffff00d0			
0xffff00c8			
0xffff00c0			
0xffff00b8			
0xffff00b0			
0xffff00a8			

puzzle.cpp

```cpp
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 }
```

Location	Value	Type	Name

0x**ffff**00f0 →

0x**ffff**00e8 →

0x**ffff**00e0 →

0x**ffff**00d8 →

0x**ffff**00d0 →

0x**ffee**00f0 →

0x**ffee**00e8 →

0x**ffee**00e0 →

0x**ffee**00d8 →

0x**ffee**00d0 →

puzzle.cpp

```cpp
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 }
```

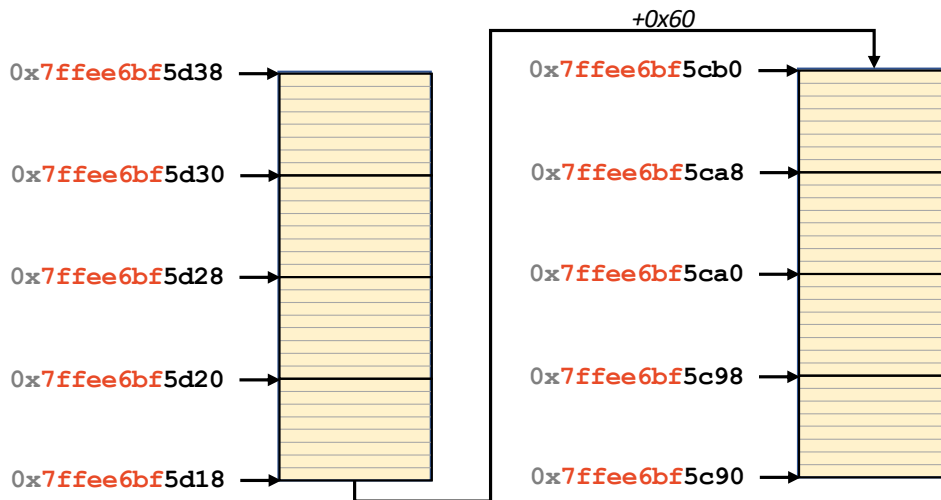# What happens on a real system?

```
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;
```

+0x60

0x7ffee6bf5d38
0x7ffee6bf5d30
0x7ffee6bf5d28
0x7ffee6bf5d20
0x7ffee6bf5d18

0x7ffee6bf5cb0
0x7ffee6bf5ca8
0x7ffee6bf5ca0
0x7ffee6bf5c98
0x7ffee6bf5c90

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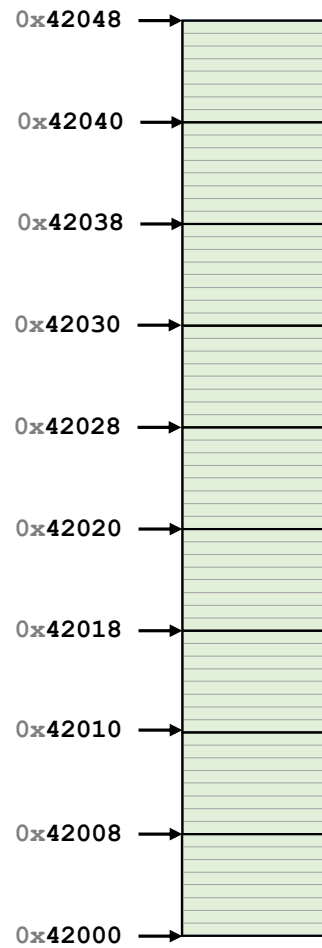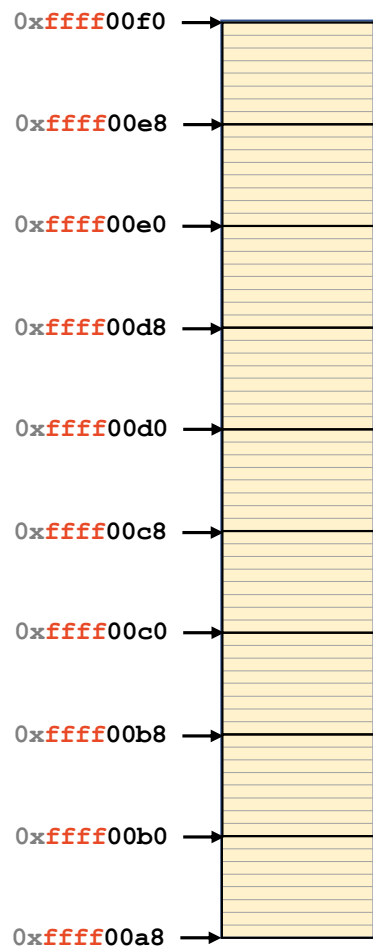
Real results when running on **linus.ews.illinois.edu**
```
&c (CreateCube): 0x7ffee6bf5ca8
8000
c->getVolume(): 2.07941e-317
```

Stack Memory   vs.   Heap Memory

Stack memory column (yellow) with addresses:
0xffff00f0
0xffff00e8
0xffff00e0
0xffff00d8
0xffff00d0
0xffff00c8
0xffff00c0
0xffff00b8
0xffff00b0
0xffff00a8

Heap memory column (green) with addresses:
0x42048
0x42040
0x42038
0x42030
0x42028
0x42020
0x42018
0x42010
0x42008
0x42000
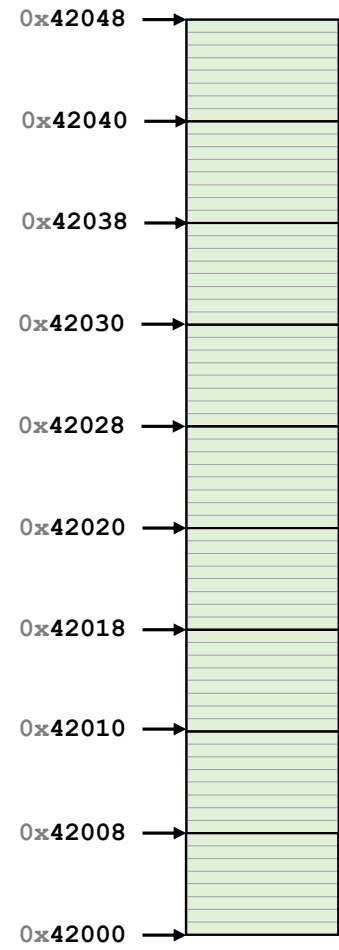
Heap Memory

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 <u>only</u> 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".
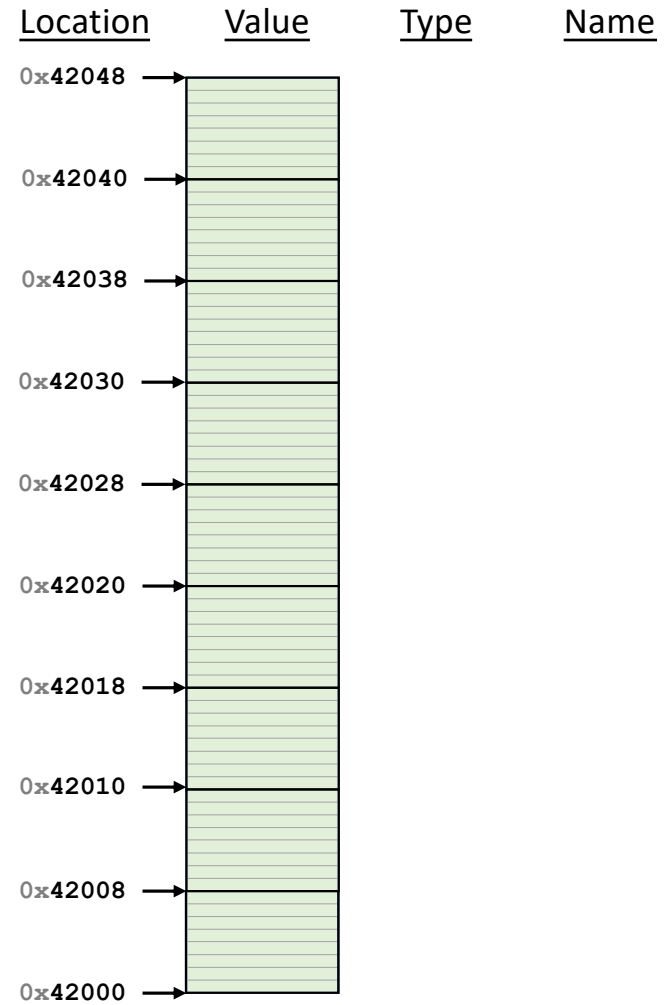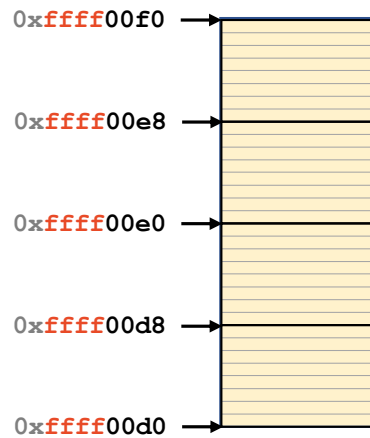
heap1.cpp

```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  }
```

Location	Value	Type	Name
0x42048			
0x42040			
0x42038			
0x42030			
0x42028			
0x42020			
0x42018			
0x42010			
0x42008			
0x42000			

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

heap2.cpp

```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 }
```

Location	Value	Type	Name
0x42048			
0x42040			
0x42038			
0x42030			
0x42028			
0x42020			
0x42018			
0x42010			
0x42008			
0x42000			

0xffff00f0

0xffff00e8

0xffff00e0

0xffff00d8

0xffff00d0

extra-puzzle1.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int *p;
    int x;

    p = &x;
    x = 6;

    cout << x << endl;
    cout << p << endl;

    return 0;
}
```

extra-puzzle2.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
  int *p, *q;
  p = new int;
  q = p;
  *q = 8;
  cout << *p << endl;

  q = new int;
  *q = 9;
  cout << *p << endl;
  cout << *q << endl;

  return 0;
}
```