CS 225

Data Structures

August 31 – Memory

Wade Fagen-Ulmschneider

Pointers and References

A variable containing an instance of an object:

```
Cube s1;
```

A reference variable of a Cube object:

```
Cube & s1;
```

A variable containing a pointer to a Cube object:

```
Cube * s1;
```

Pointers

Three key ideas:

1.

2.

3.

main.cpp

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

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;

  return 0;
}
```

Indirection Operators

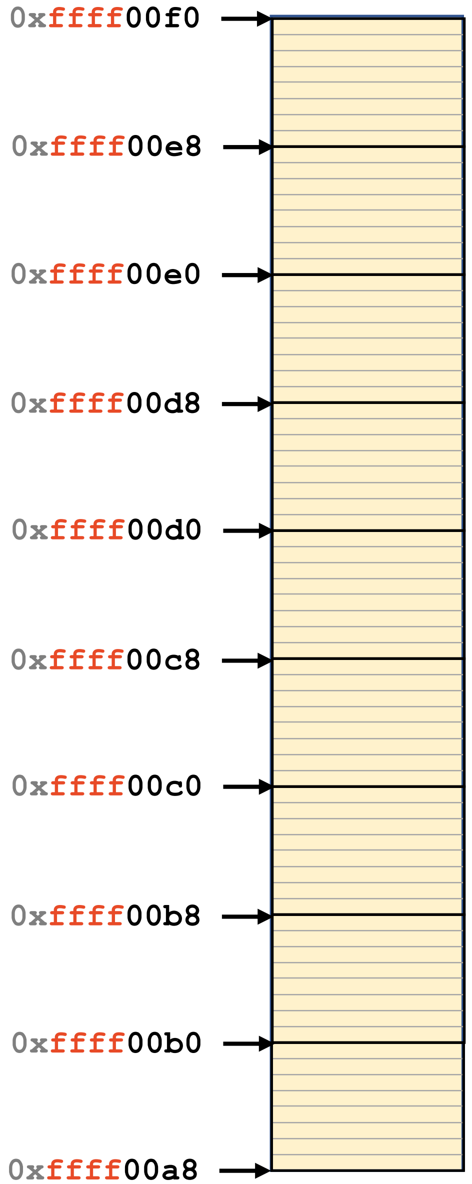
Given any variable v:

&v

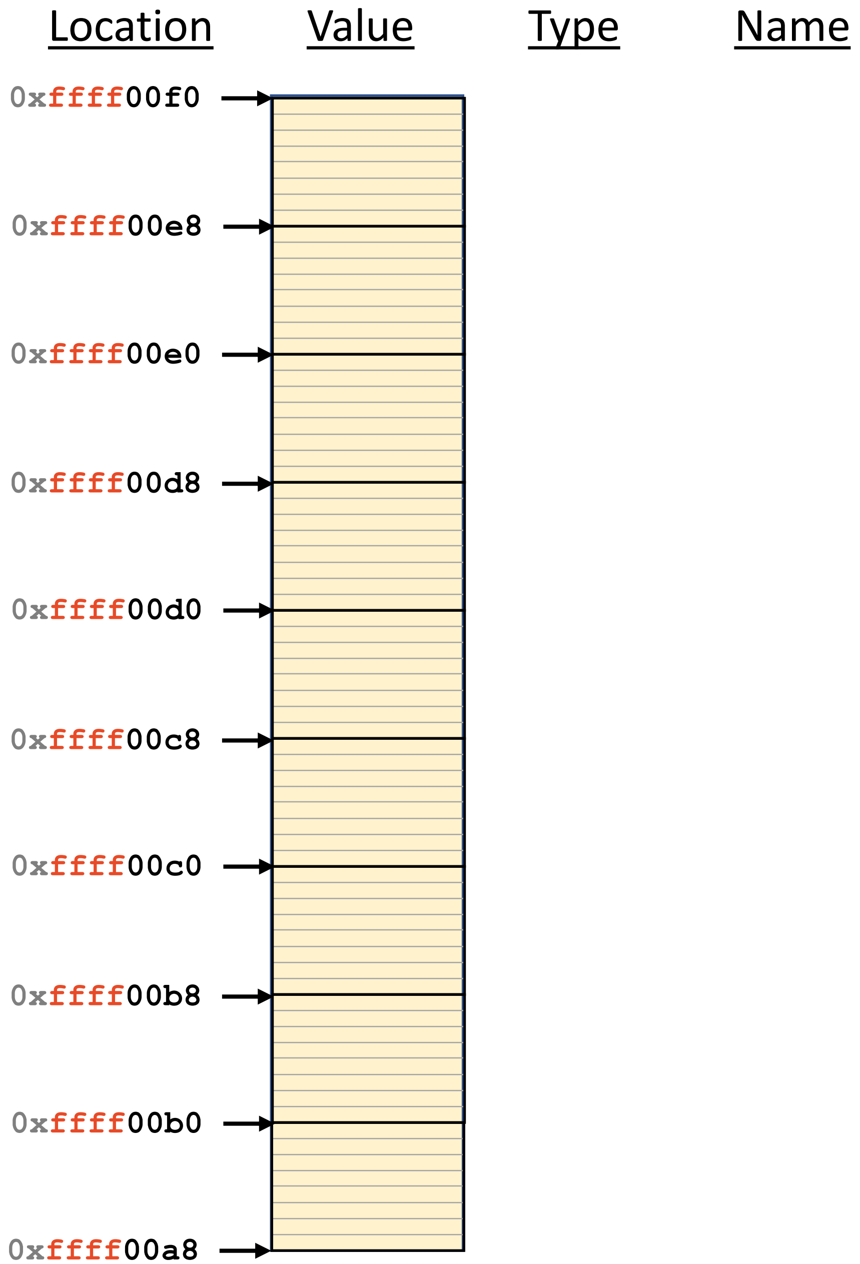
***v**

v->

Stack Memory



example1.cpp



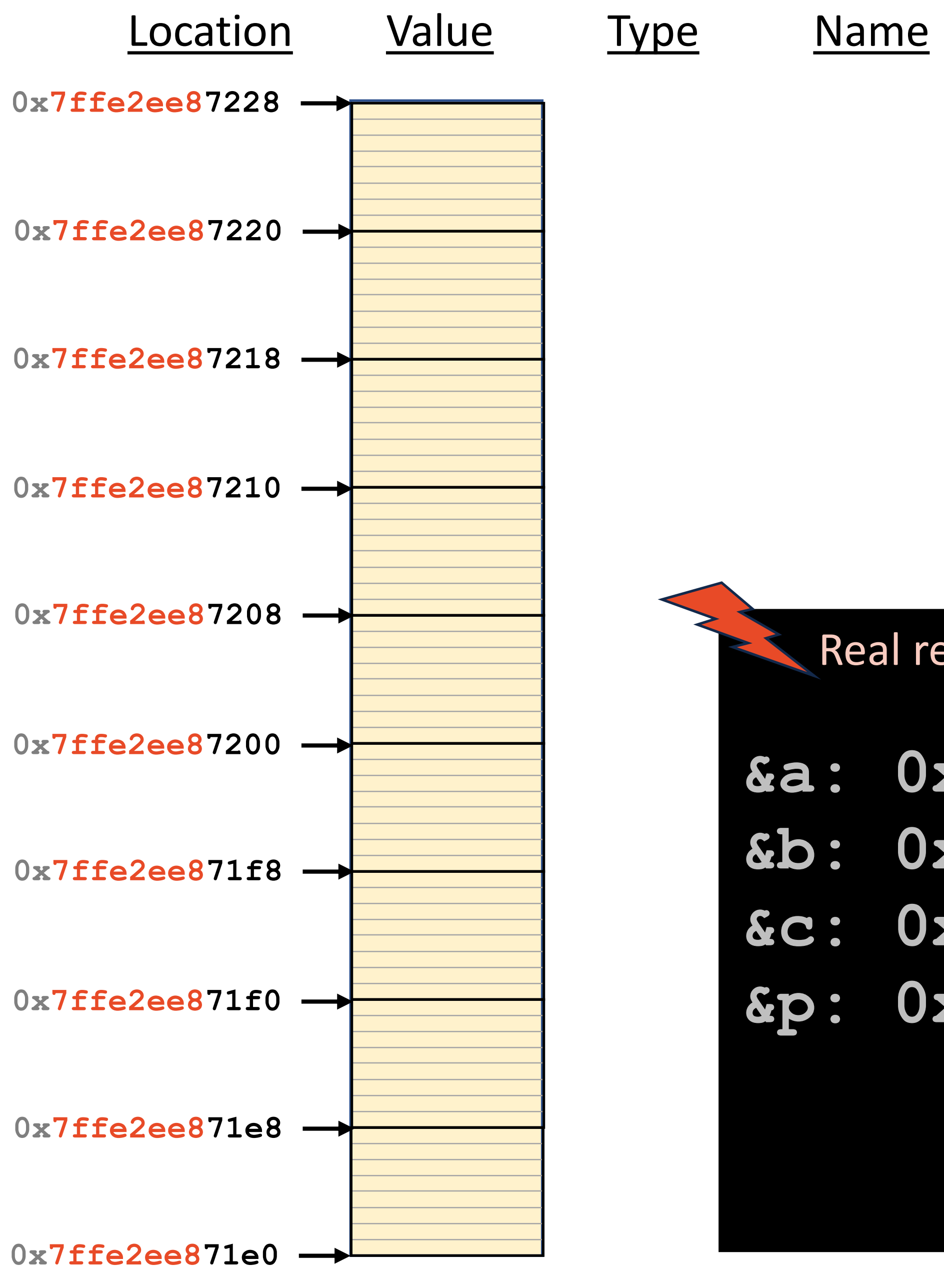
```cpp
int main() {
  int a;
  int b = -3;
  int c = 12345;

  int *p = &b;

  return 0;
}
```

sizeof-int.cpp

```cpp
#include <iostream>

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

sizeof-intptr.cpp

```cpp
#include <iostream>

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



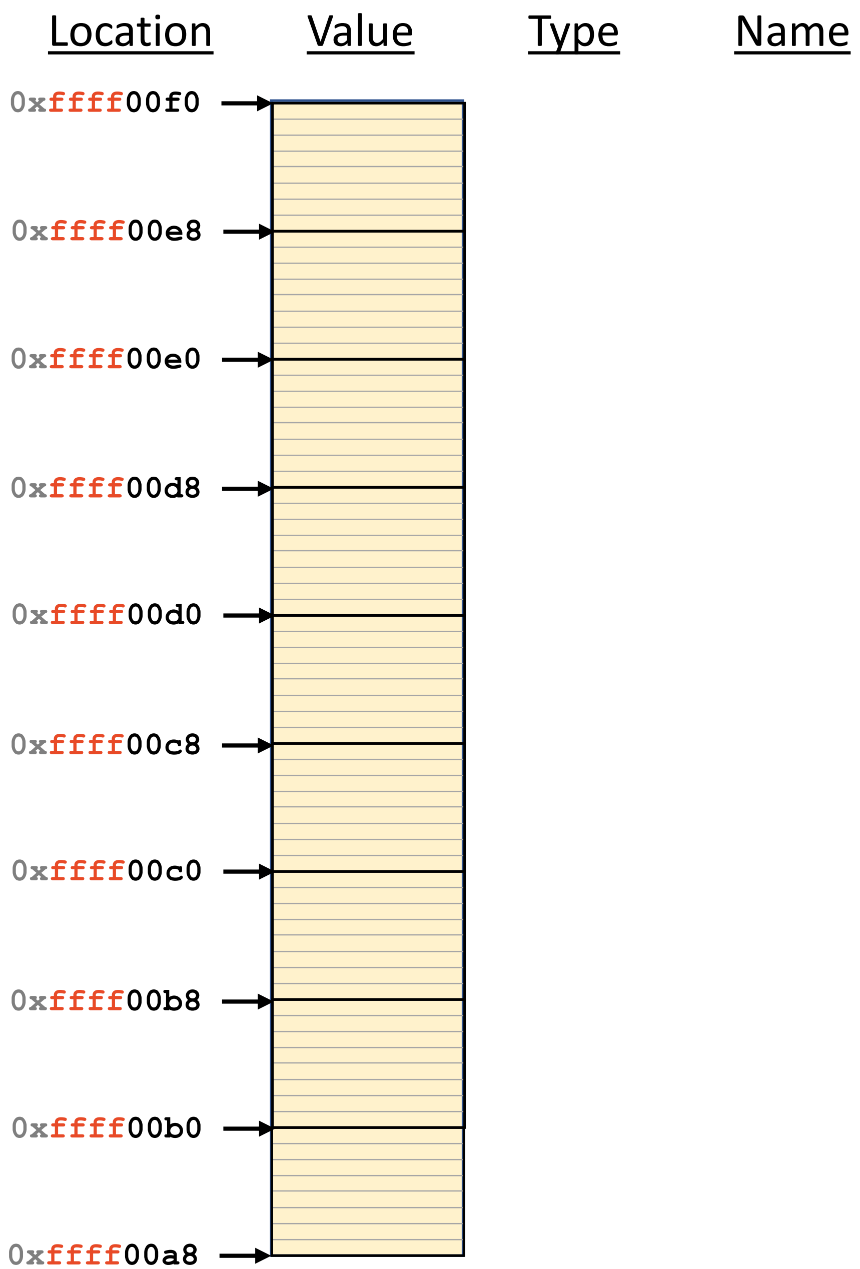
example1.cpp

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

  int *p = &b;

  return 0;
}
```

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

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

example2.cpp

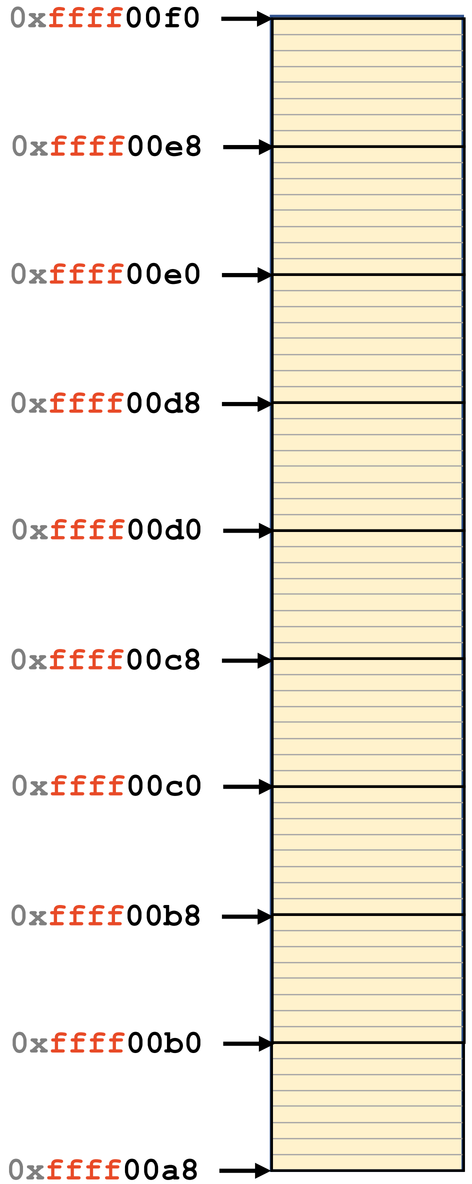


```cpp
#include "Cube.h"

int main() {
  cs225::Cube c;
  cs225::Cube *p = &c;

  return 0;
}
```

sizeof-cube.cpp

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

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

Stack Frames



stackframe.cpp

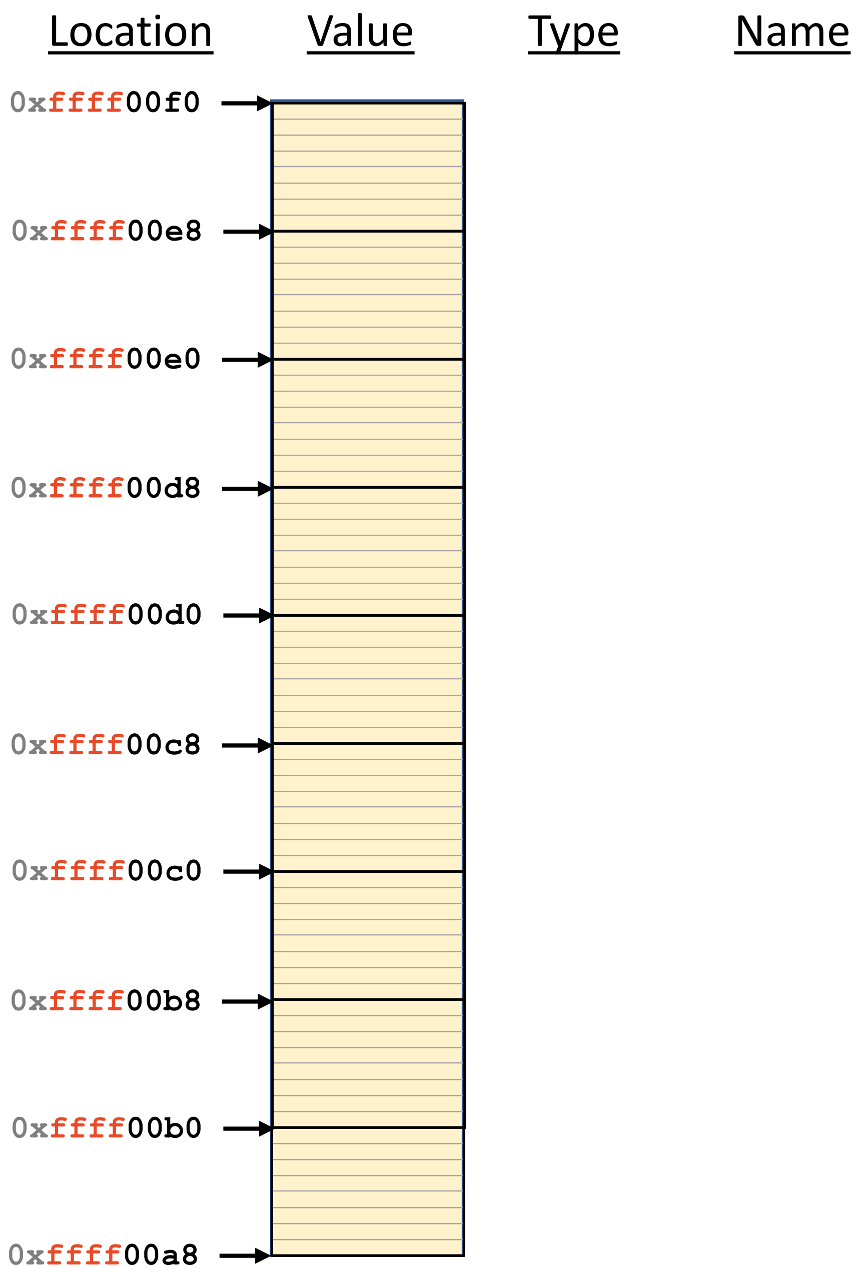
```
int hello() {
  int a = 100;
  return a;
}

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

  return 0;
}
```

Problems of the Day (POTD)

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

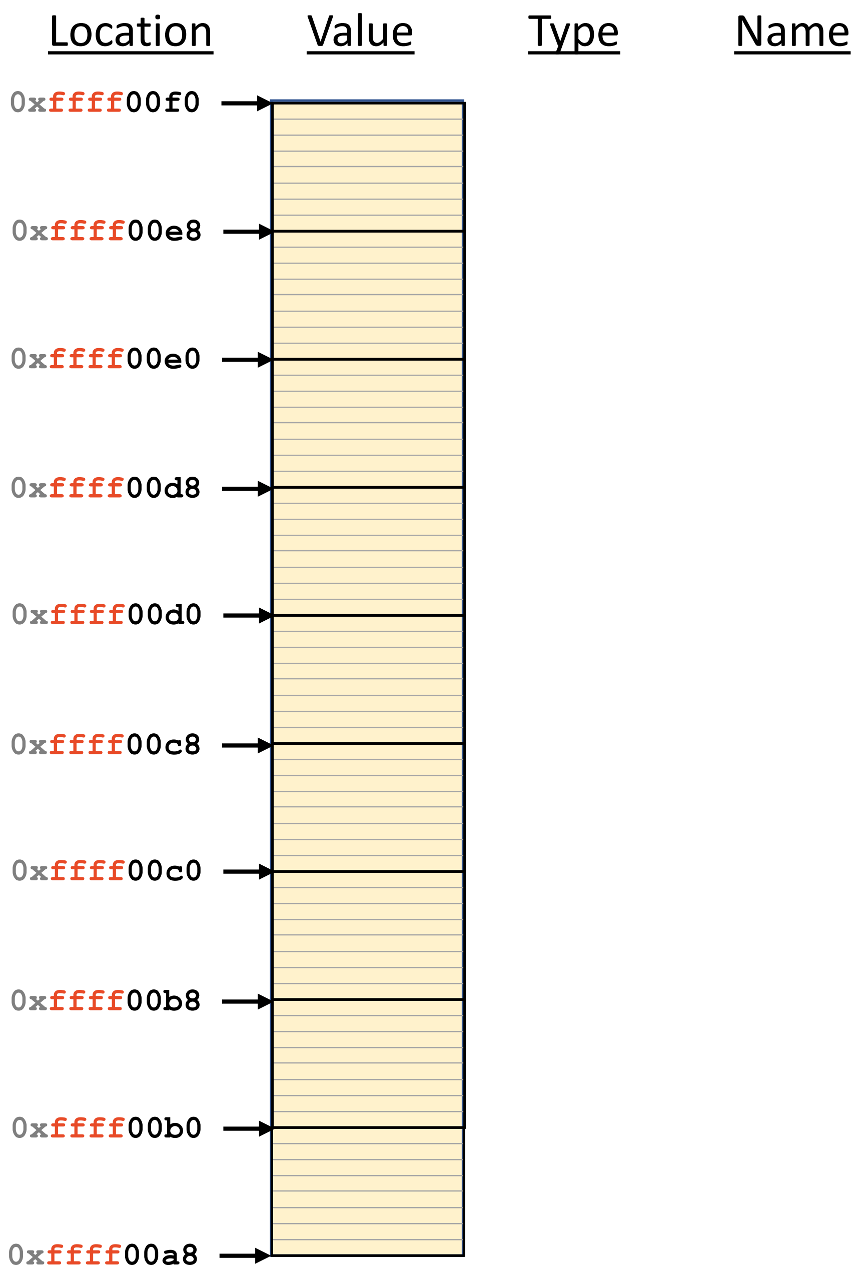
Each POTD is worth **+1** extra credit point, capped at **+40**.
(Course-wide, all extra credit is capped at +100.)

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



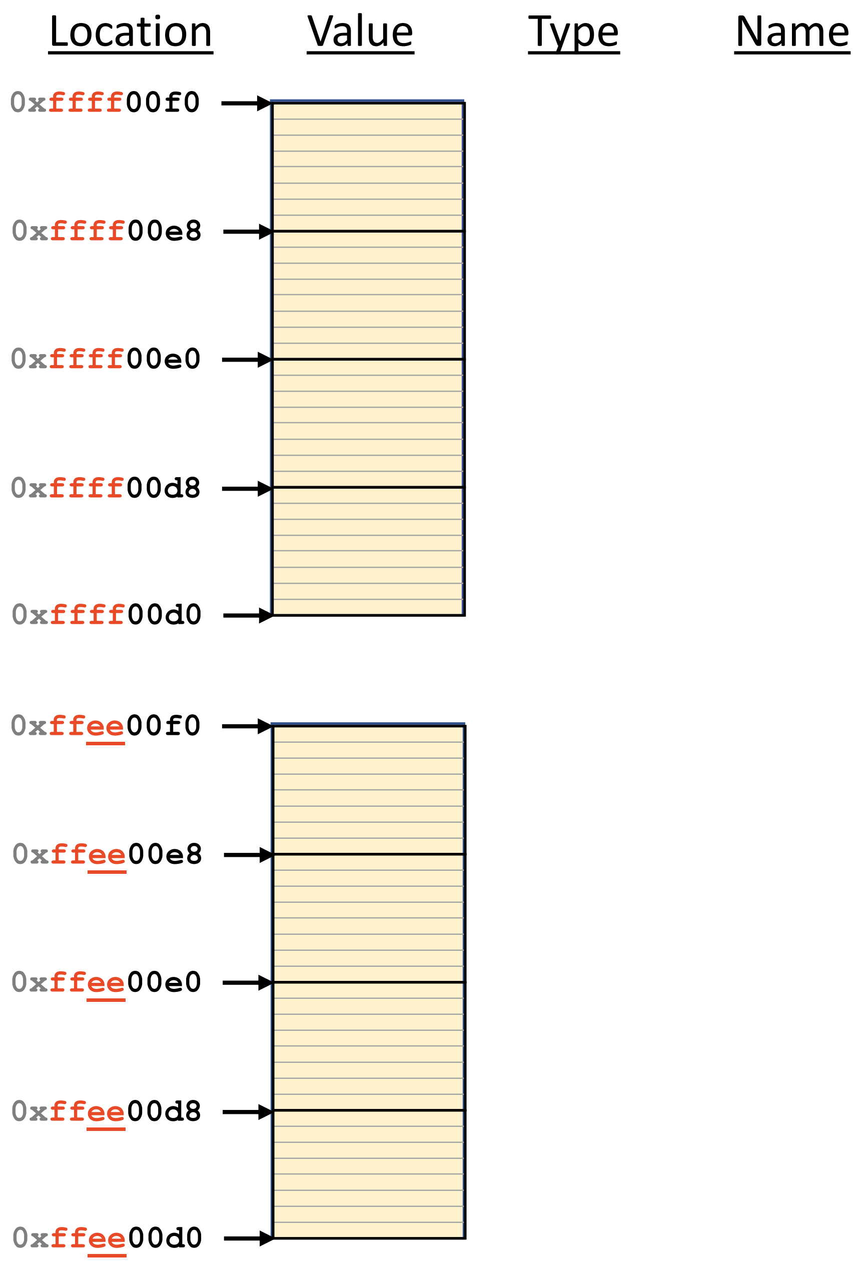
puzzle.cpp

```cpp
#include "Cube.h"
using cs225::Cube;

Cube *CreateCube() {
  Cube c(20);
  return &c;
}

int main() {
  Cube *c = CreateCube();
  double r = c->getVolume();
  double v = c->getSurfaceArea();
  return 0;
}
```



puzzle.cpp

```
#include "Cube.h"
using cs225::Cube;

Cube *CreateCube() {
  Cube c(20);
  return &c;
}

int main() {
  Cube *c = CreateCube();
  double r = c->getVolume();
  double v = c->getSurfaceArea();
  return 0;
}
```



puzzle.cpp

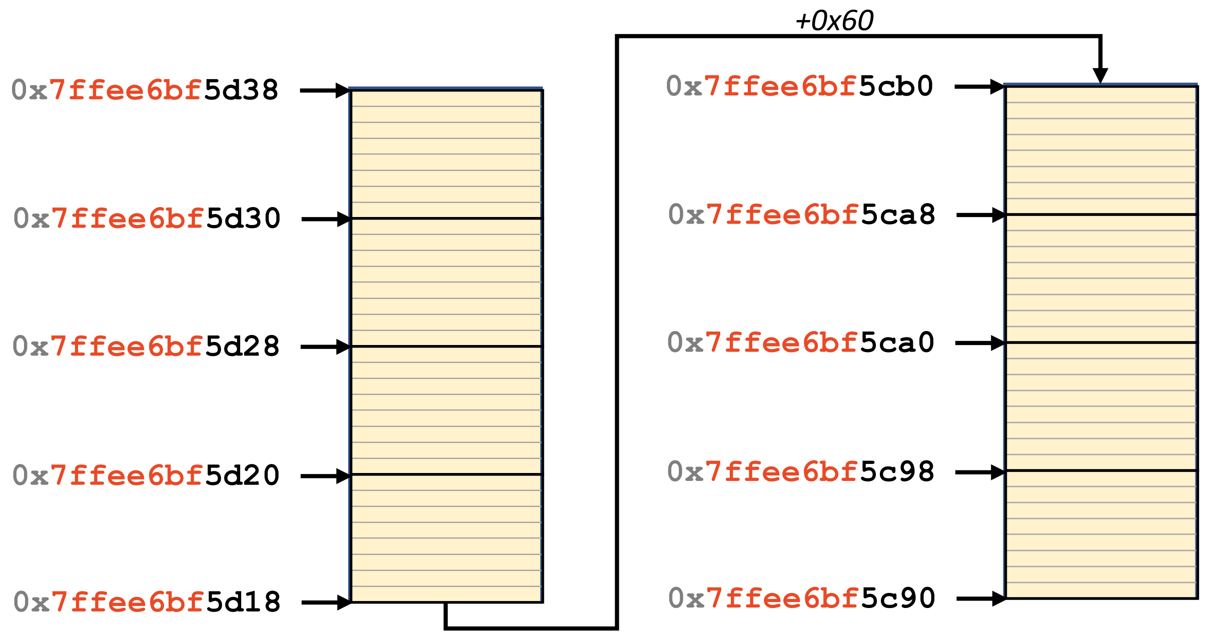
```
#include "Cube.h"
using cs225::Cube;

Cube *CreateCube() {
  Cube c(20);
  return &c;
}

int main() {
  Cube *c = CreateCube();
  double r = c->getVolume();
  double v = c->getSurfaceArea();
  return 0;
}
```

What happens on a real system?

```cpp
int main() {
  Cube *c = CreateCube();
  cout << c-> getVolume() << endl;
  cout << "c->getVolume(): ‟ << c->getVolume() << endl;
  cout << "&c (main): " << &c << endl;
  cout << " c (main): " <<  c << endl;
  double r = c->getVolume();
  cout << "&r (main): " << &c << endl;
  cout << " r (main): " <<  c << endl;
  double v = c->getSurfaceArea();
  cout << "&v (main): " << &c << endl;
  cout << " v (main): " <<  c << endl;
  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
```

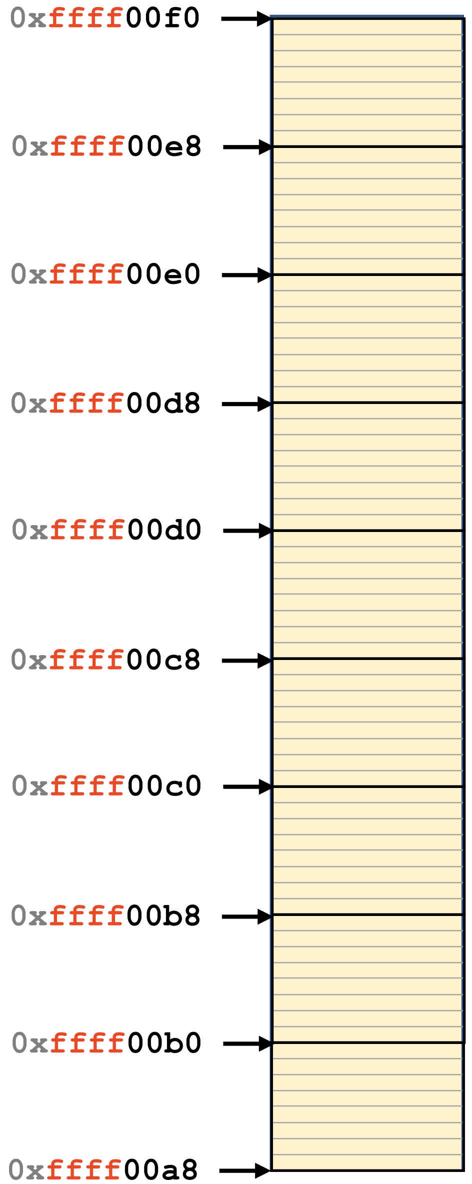


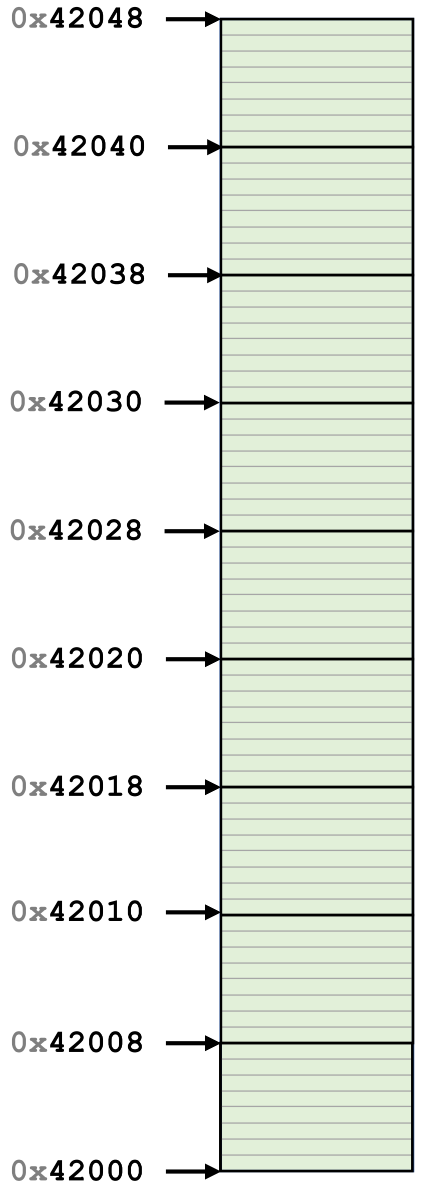
```
int main() {
  Cube *c = CreateCube();
  cout << c-> getVolume() << endl;
  cout << "c->getVolume(): ‟ << c->getVolume() << endl;
```

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

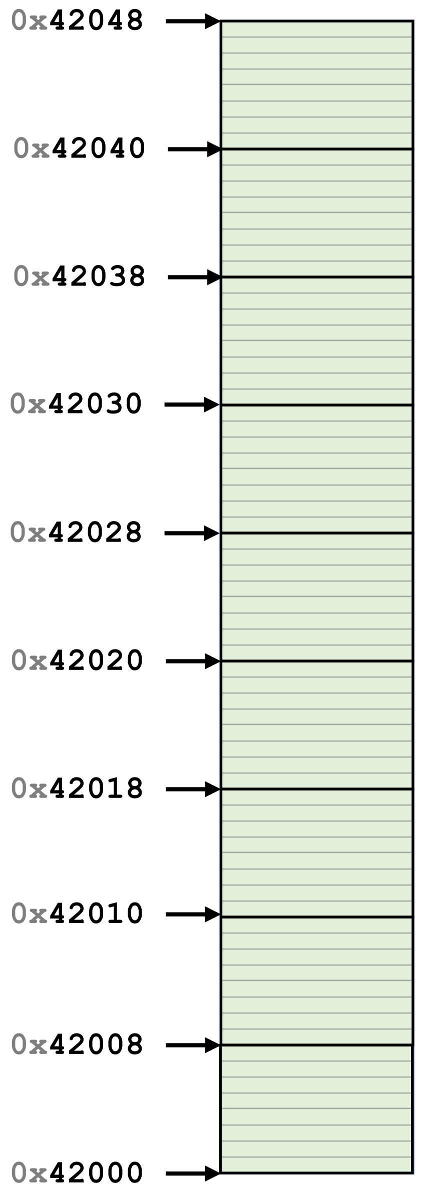
```
&c (CreateCube): 0x7ffee6bf5ca8
8000
c->getVolume(): 2.07941e-317
```

Stack Memory vs. Heap Memory





Heap Memory


0x42048
0x42040
0x42038
0x42030
0x42028
0x42020
0x42018
0x42010
0x42008
0x42000

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

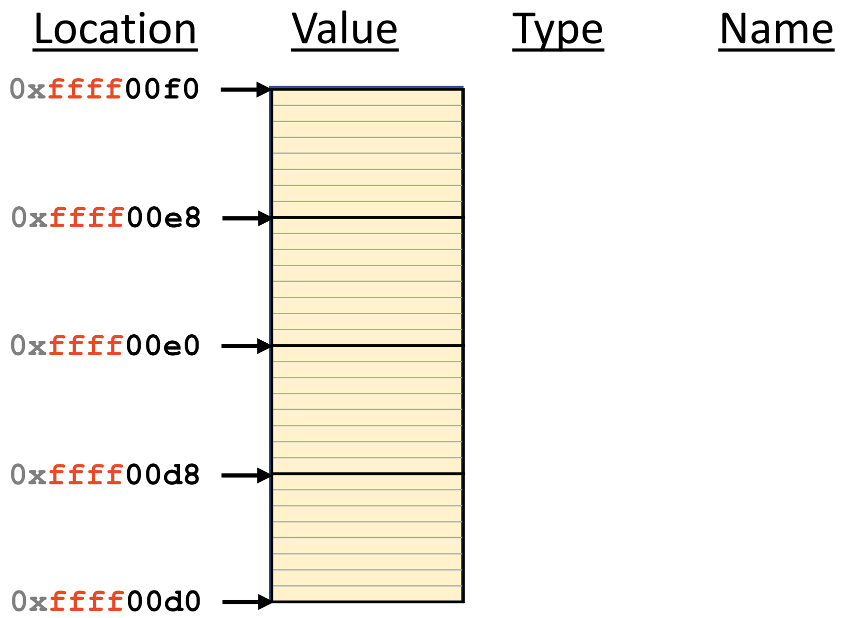
2. The <u>only</u> way to free heap memory is with the use of the **delete** keyword. Using **delete** will:

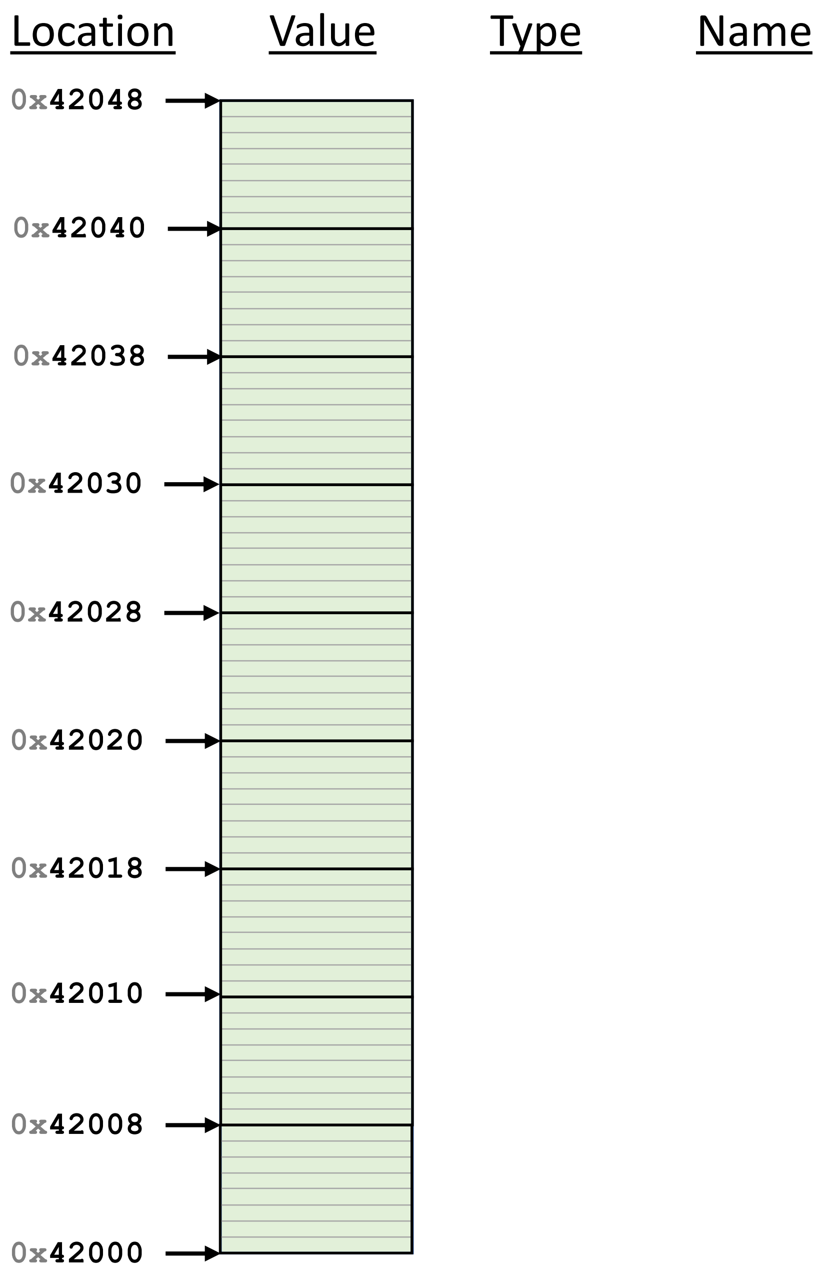
-
-

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

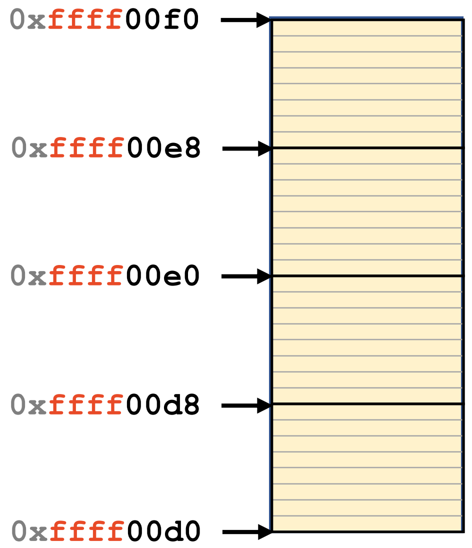
```cpp
#include "Cube.h"
using cs225::Cube;

int main() {
  int *p = new int;
  Cube *c = new Cube(10);

  return 0;
}
```

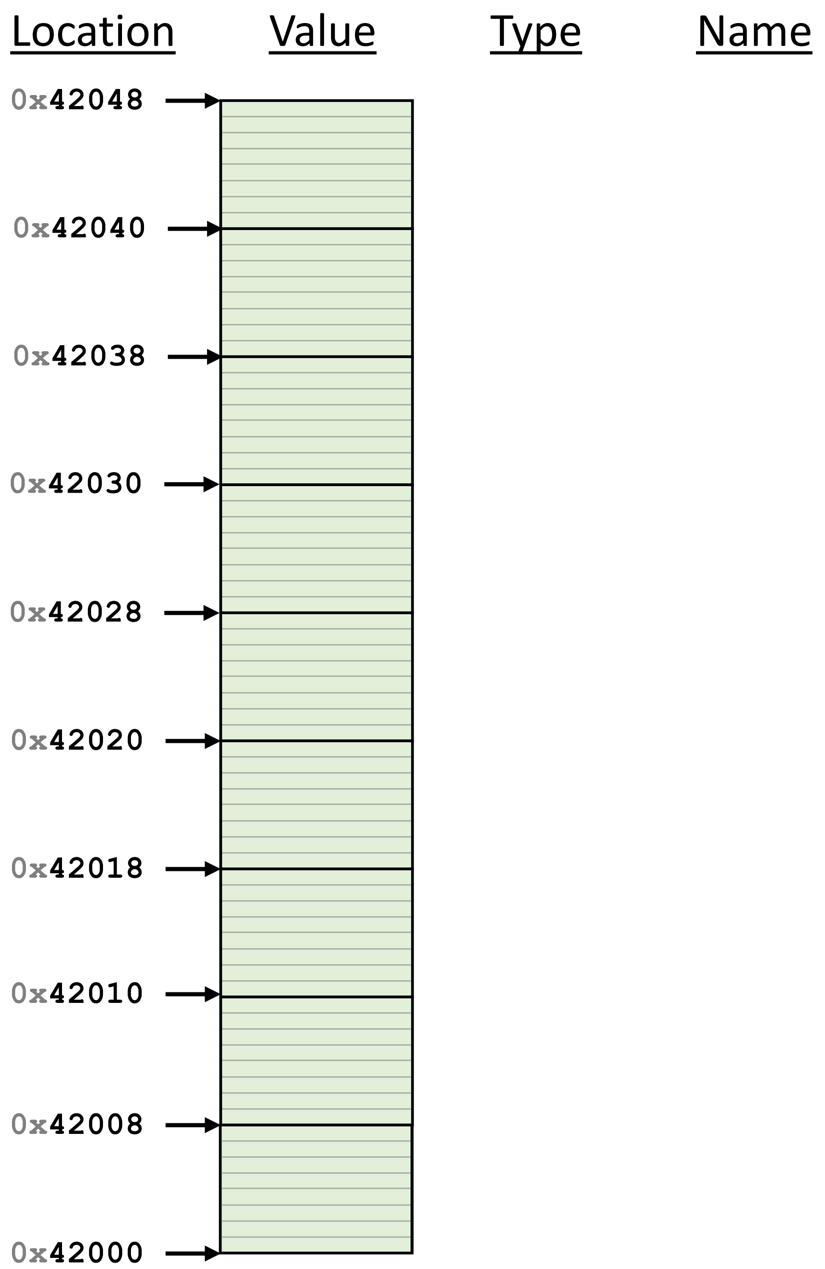




heap2.cpp

```cpp
#include "Cube.h"
using cs225::Cube;

int main() {
  Cube *c1 = new Cube();
  Cube *c2 = c1;

  c2->setLength( 10 );

  return 0;
}
```





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;
}
```