



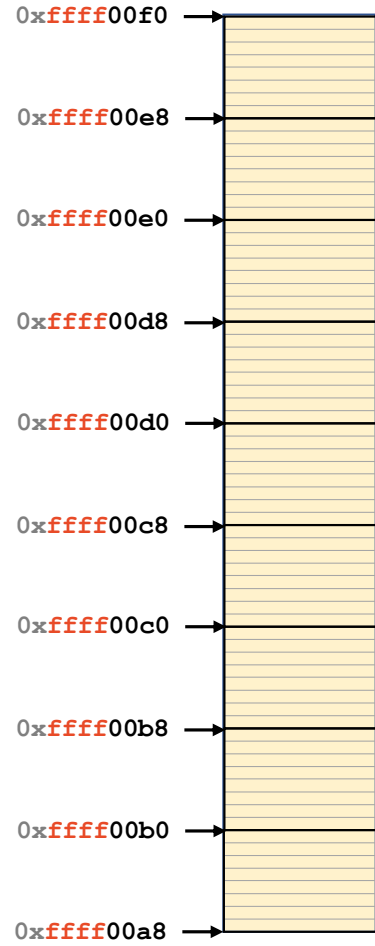
CS 225

Data Structures

August 31 – Heap Memory

G Carl Evans

Stack Memory



<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     SomeOtherFunction();
12     double v = c->getVolume();
13     double a = c->getSurfaceArea();
14     return 0;
15 }

```

puzzle.cpp

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

main's stack frame

```

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    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }

```

puzzle.cpp

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

main's stack frame

Cube * c

CreateCube frame

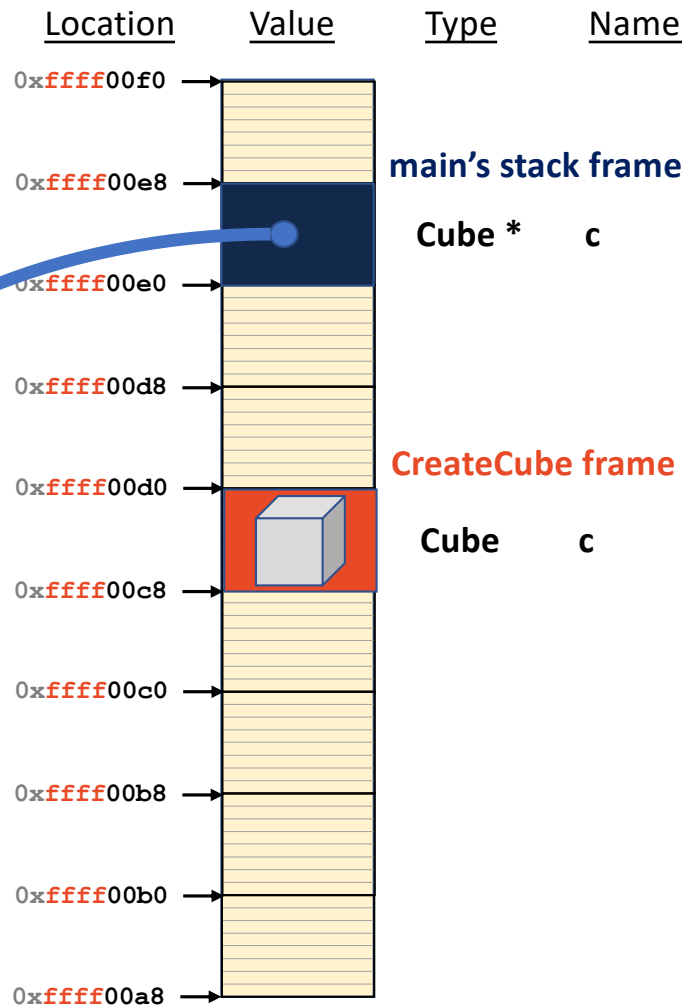
Cube c

```

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    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }

```

puzzle.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    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }

```

puzzle.cpp

<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0			
0xffff00e8		main's stack frame	
0xffff00e0		Cube *	c
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    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }

```

puzzle.cpp

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

main's stack frame

SomeOtherFunction

```

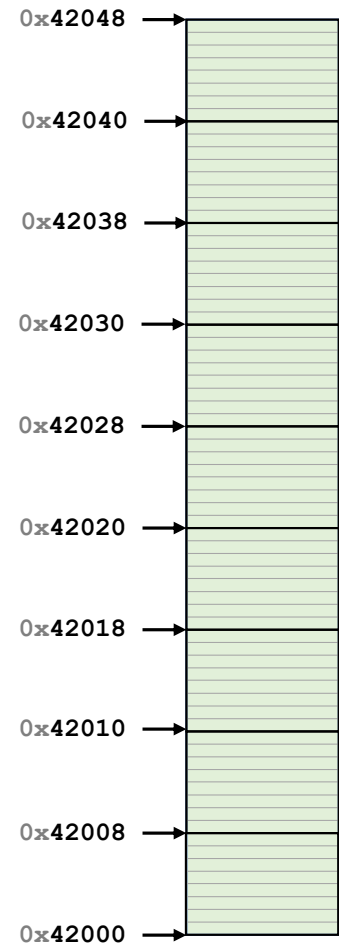
1 #include "Cube.h"
2 using cs225::Cube;
3
4 Cube *CreateCube() {
5     Cube c(20);
6     return &c;
7 }
8
9 int main() {
10    Cube *c = CreateCube();
11    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }

```

puzzle.cpp



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

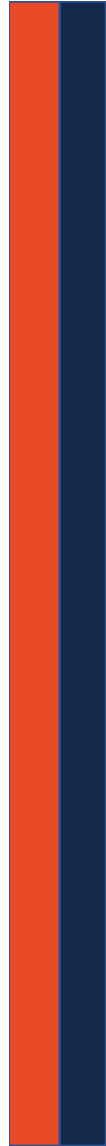
2. The only way to free heap memory is with the use of the **delete** keyword. Using **delete** will:

-
-

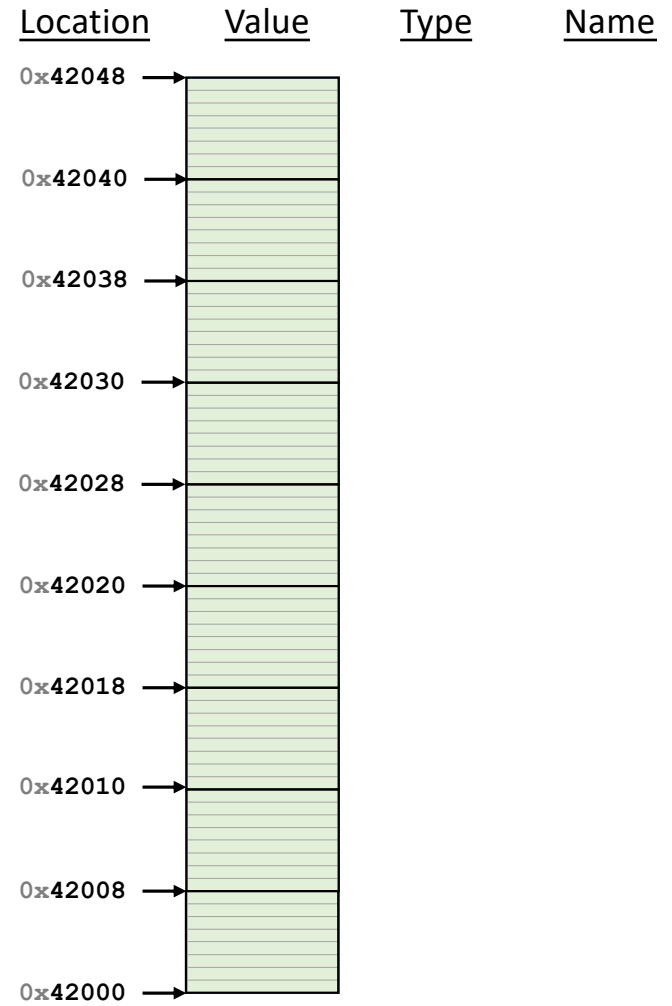
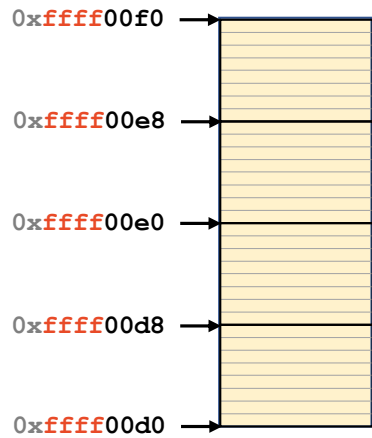
3. Memory is never automatically reclaimed, even if it goes out of scope. Any memory lost, but not freed, is considered to be “leaked memory”.



Heap Memory vs. Stack Memory Lifecycle



```
1 int main() { heap2.cpp
2   Cube *c1 = new Cube();
3   Cube *c2 = c1;
4   c2->setRadius( 10 );
5   delete c2;
6   delete c1;
7   return 0;
8 }
9
10
11
```

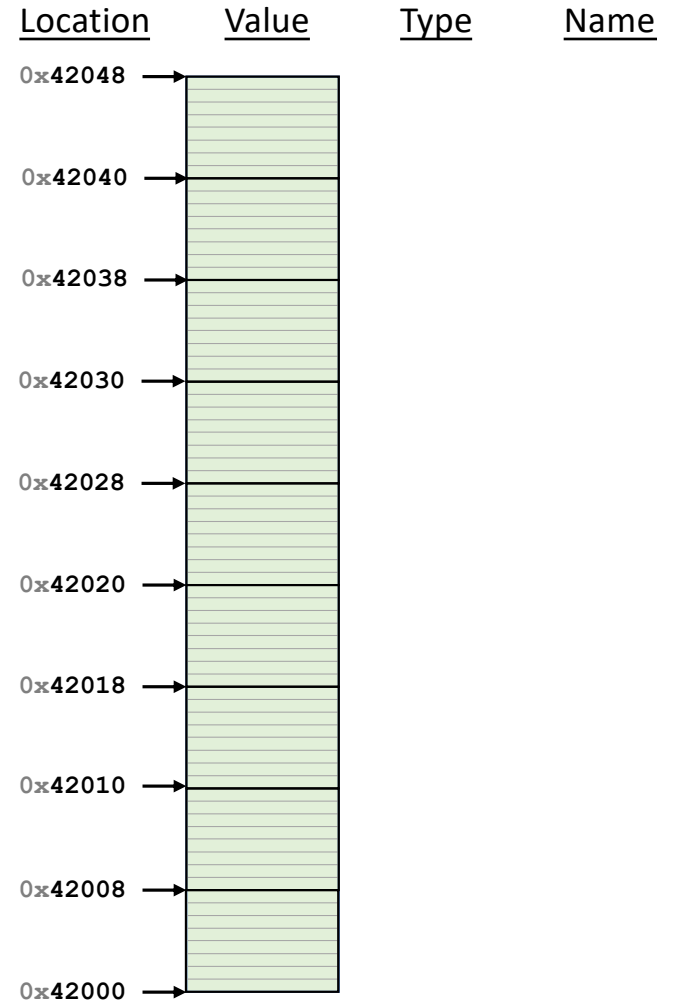
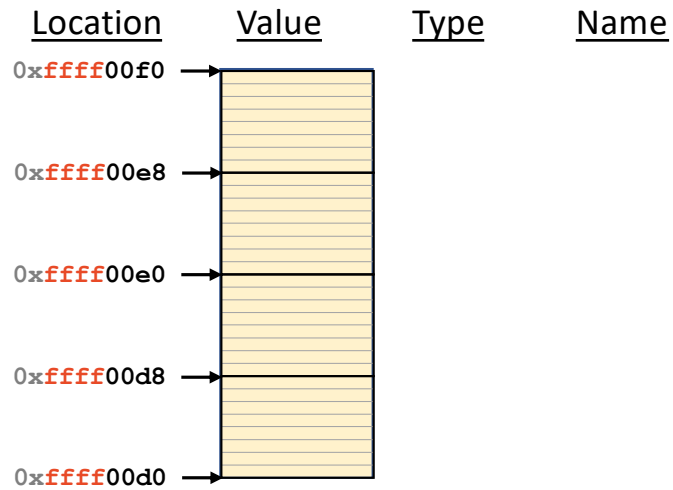


```

8 int main() {
9     int *p = new int;
10    cs225::Cube *c = new cs225::Cube(10);
11
12    return 0;
13 }

```

heap1.cpp



copy.cpp

```
1 #include <iostream>
2 using std::cout;
3 using std::endl;
4
5 int main() {
6     int i = 2, j = 4, k = 8;
7     int *p = &i, *q = &j, *r = &k;
8
9     k = i;
10    cout << i << j << k << *p << *q << *r << endl;
11
12    p = q;
13    cout << i << j << k << *p << *q << *r << endl;
14
15    *q = *r;
16    cout << i << j << k << *p << *q << *r << endl;
17 }
```

Pointers and References

A variable containing an instance of an object:

```
1 Cube s1;
```

A reference variable of a Cube object:

```
2 Cube & r1 = s1;
```

A variable containing a pointer to a Cube object:

```
3 Cube * p1;
```




Reference Variable

A reference variable is an alias to an existing variable.

Key Idea: Modifying the reference variable modifies the variable being aliased.



Reference Variable

Three facts about reference variables:

1.

2.

3.

Reference Variable

A reference variable is an alias to an existing variable.

```
1 #include <iostream>
2
3 int main() {
4     int i = 7;
5     int & j = i;    // j is an alias of i
6
7     j = 4;
8     std::cout << i << " " << j << std::endl;
9
10    i = 2;
11    std::cout << i << " " << j << std::endl;
12    return 0;
13 }
```

heap-puzzle1.cpp

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

heap-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 }
```

heap-puzzle3.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int *x;
6     int size = 3;
7
8     x = new int[size];
9
10    for (int i = 0; i < size; i++) {
11        x[i] = i + 3;
12    }
13
14    delete[] x;
15 }
16
17
```