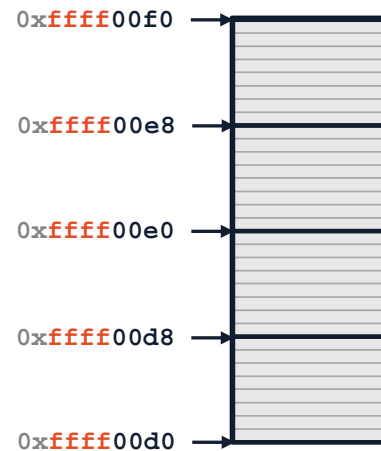


# Stack Memory and Pointers

Prof. Wade Fagen-Ulmschneider

**I** ILLINOIS

In C++, the programmer has control over the memory and lifecycle of every variable! By default, variables live in stack memory.



# A Variable

Every C++ variable has four things:

- A name
- A type
- A value
- A location in memory (“memory address”)

```
int primeNumber = 7;
```

# A Variable's Memory Address

In C++, the **&** operator returns the memory address of a variable.

## cpp-memory/addressOf.cpp

```
8 #include <iostream>
9
10 int main() {
11     int num = 7;
12
13     std::cout << "Value: " << num << std::endl;
14     std::cout << "Address: " << &num << std::endl;
15
16     return 0;
17 }
```

# Stack Memory

By default, every variable in C++ is placed in stack memory.

Stack memory is associated with the current function and the memory's lifecycle is tied to the function:

- When the function returns or ends, the stack memory of that function is released.

# Stack Memory

Stack memory always starts from high addresses and grows down:



## cpp-memory/foo.cpp

```
8 #include <iostream>
9
10 void foo() {
11     int x = 42;
12     std::cout << " x in foo(): " << x << std::endl;
13     std::cout << "&x in foo(): " << &x << std::endl;
14 }
15
16 int main() {
17     int num = 7;
18     std::cout << " num in main(): " << num << std::endl;
19     std::cout << "&num in main(): " << &num << std::endl;
20
21     foo();
22
23     return 0;
24 }
```



# Pointer

A pointer is a variable that stores the memory address of the data.

- *Simply put: pointers are a level of indirection from the data.*

In C++, a pointer is defined by adding an `*` to the type of the variable.

- Integer pointer: `int * p = &num;`

# Dereference Operator

Given a pointer, a level of indirection can be removed using the dereference operator `*`.

```
int num = 7;  
int * p = &num;  
int value_in_num = *p;  
*p = 42;
```

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

```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp

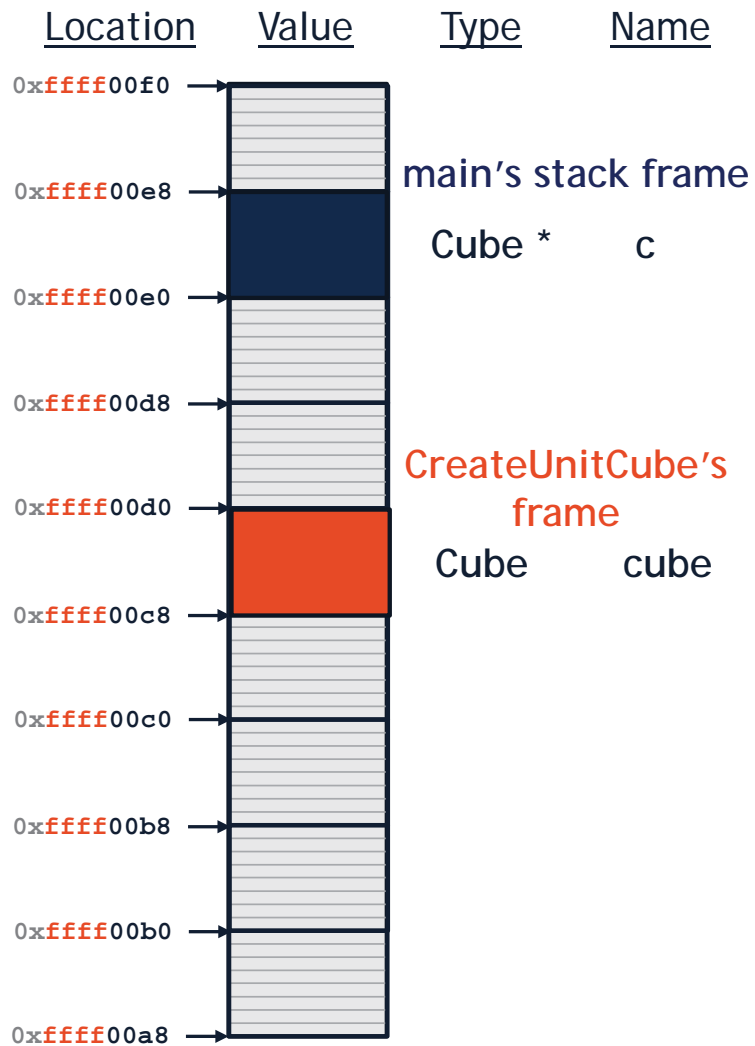
<u>Location</u>	<u>Value</u>	<u>Type</u>	<u>Name</u>
0xffff00f0 →			
0xffff00e8 →		main's stack frame	
0xffff00e0 →		Cube *	c
0xffff00d8 →			
0xffff00d0 →			
0xffff00c8 →			
0xffff00c0 →			
0xffff00b8 →			
0xffff00b0 →			
0xffff00a8 →			

```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp

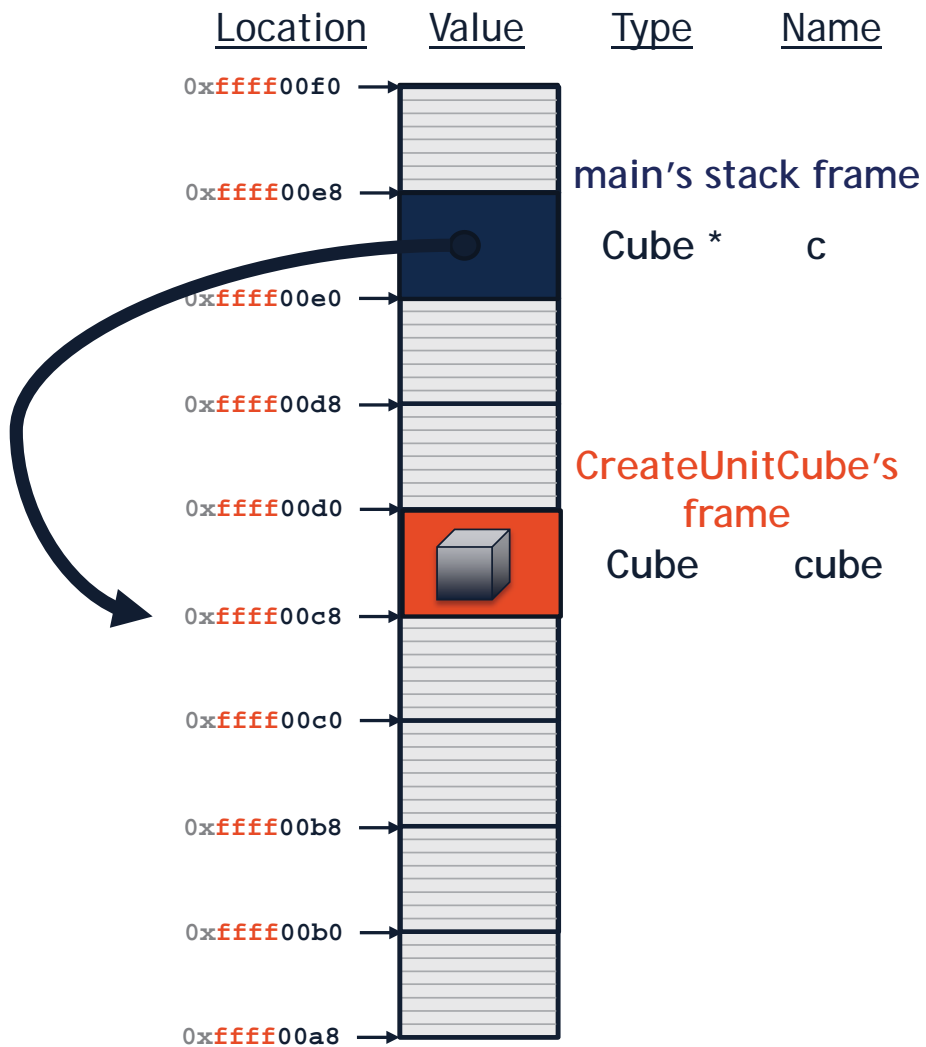


```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp

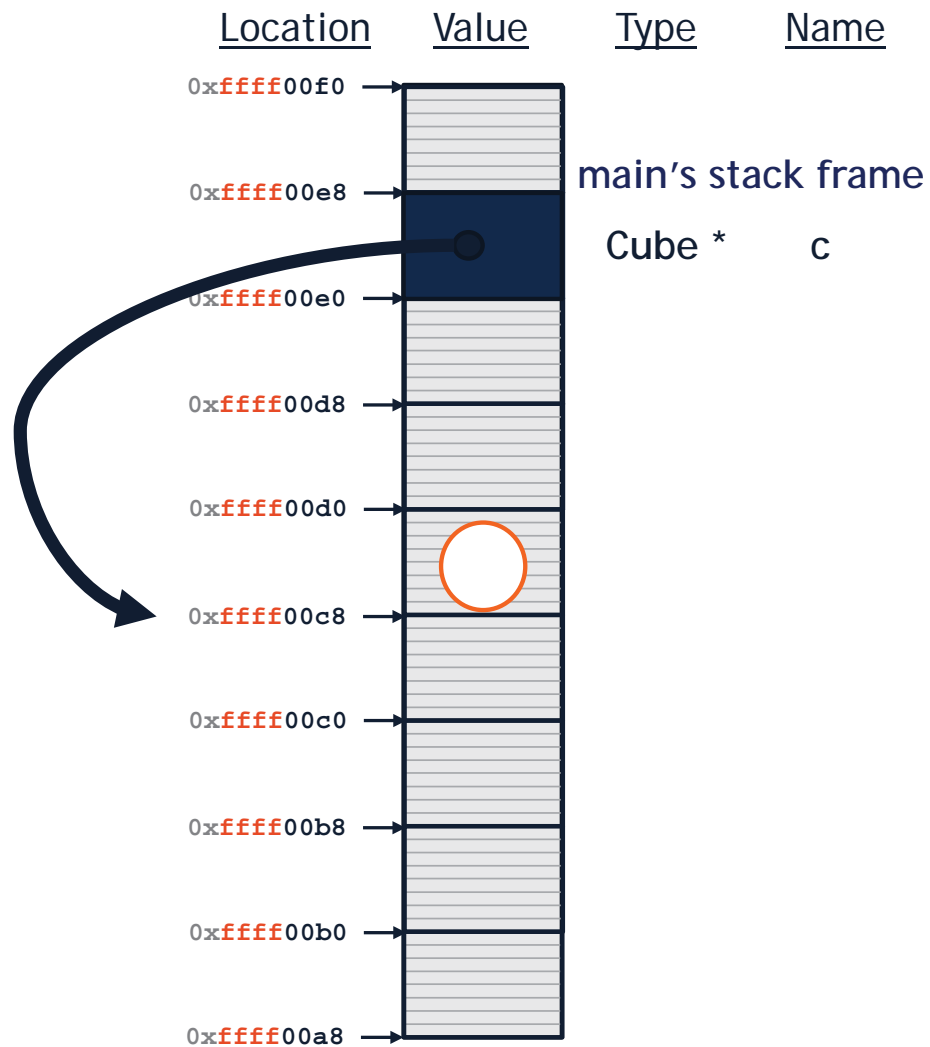


```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp

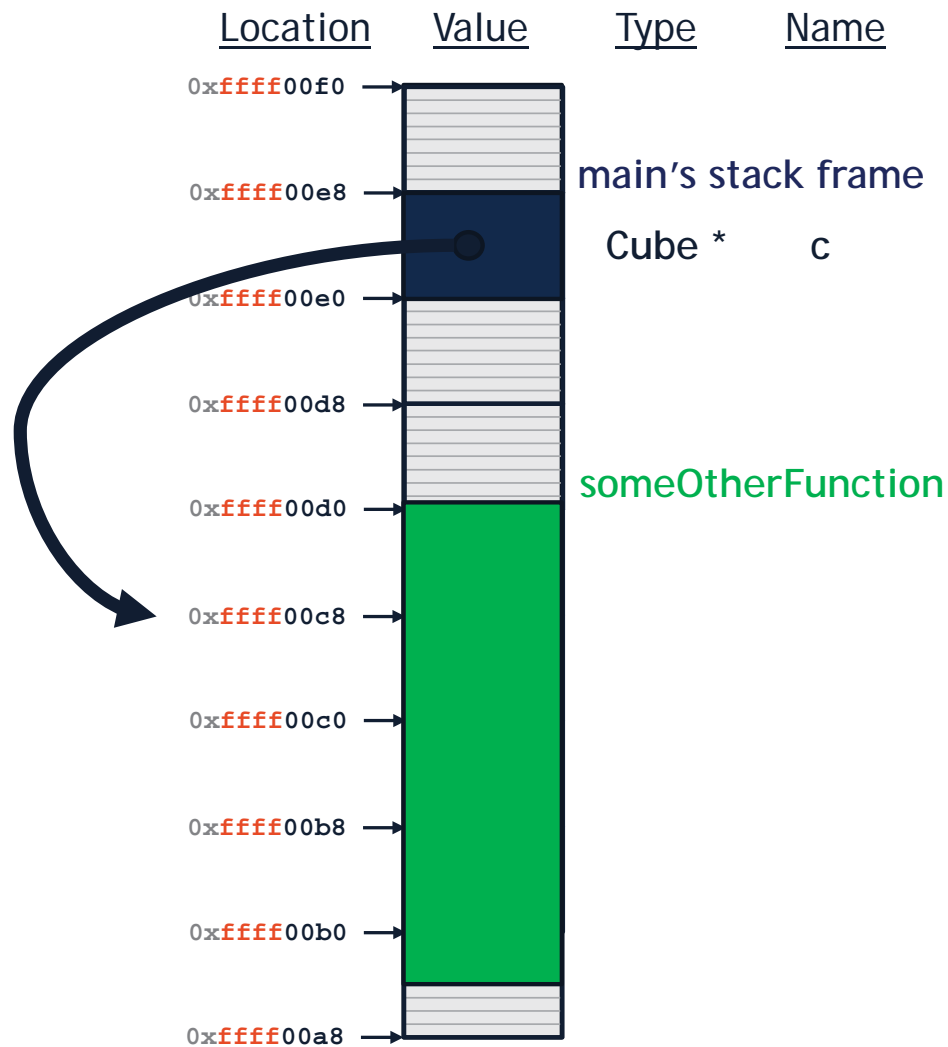


```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp



```

10 #include "Cube.h"
11 using uiuc::Cube;
12
13 Cube *CreateUnitCube() {
14     Cube cube;
15     cube.setLength(15);
16     return &cube;
17 }
18
19 int main() {
20     Cube *c = CreateUnitCube();
21     someOtherFunction();
22     double a = c->getSurfaceArea();
23     double v = c->getVolume();
24     return 0;
25 }

```

puzzle.cpp



## cpp-memory/main.cpp

```
10 int main() {
11     int num = 7;
12     std::cout << " num: " << num << std::endl;
13     std::cout << "&num: " << &num << std::endl;
14
15     int *p = &num;
16     std::cout << " p: " << p << std::endl;
17     std::cout << "&p: " << &p << std::endl;
18     std::cout << "*p: " << *p << std::endl;
19
20     *p = 42;
21     std::cout << "*p changed to 42" << std::endl;
22     std::cout << " num: " << num << std::endl;
23
24     return 0;
25 }
```