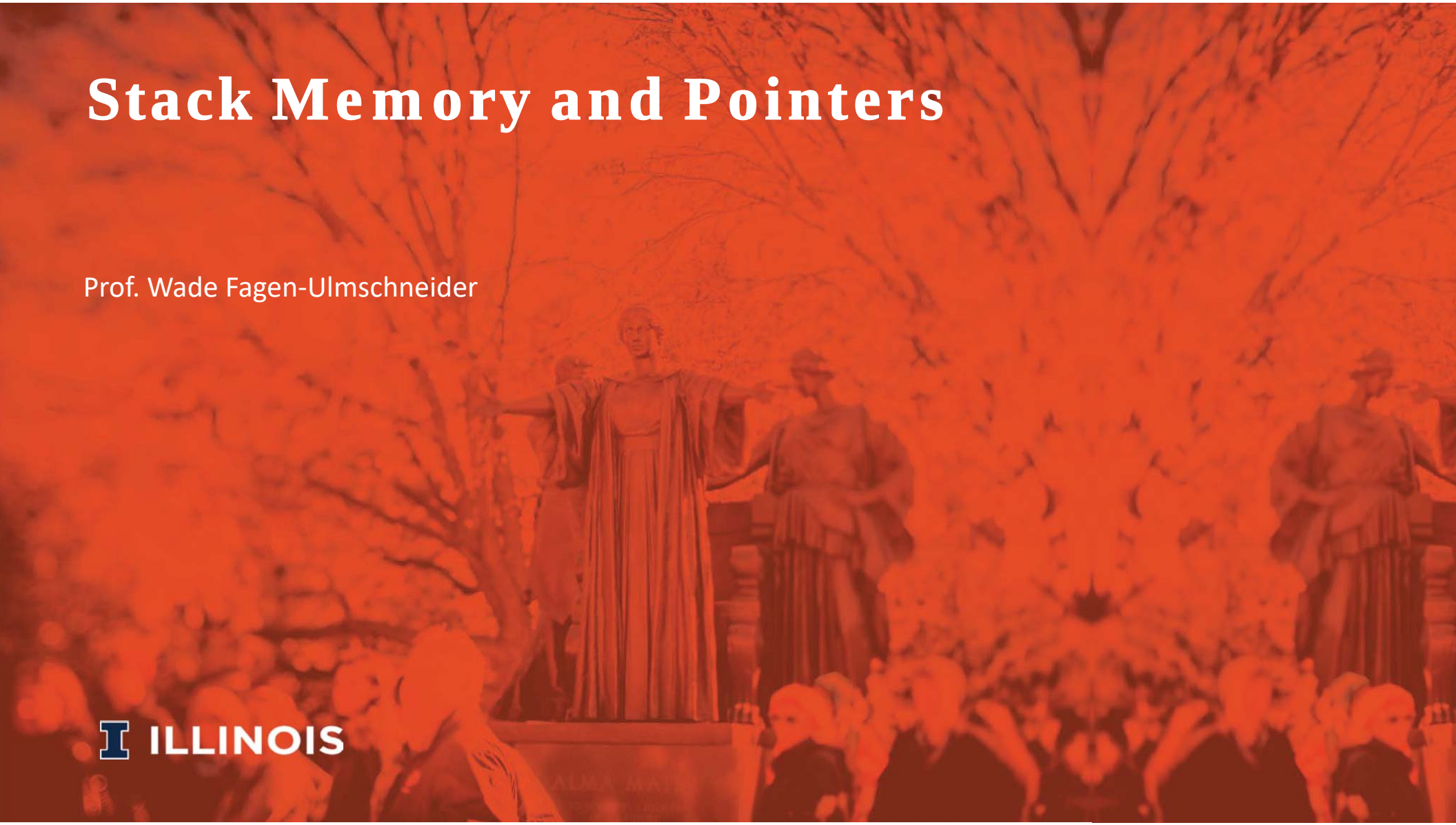
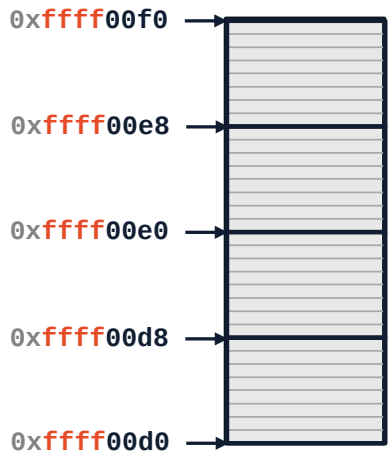
Stack Memory and Pointers

Prof. Wade Fagen-Ulmschneider



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

```cpp
#include <iostream>

int main() {
  int num = 7;

  std::cout << "Value: " << num << std::endl;
  std::cout << "Address: " << &num << std::endl;

  return 0;
}
```

Stack Memory

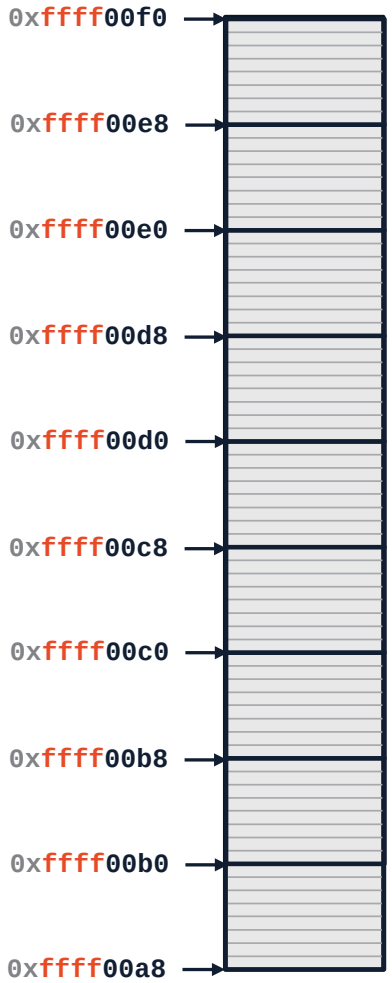
By default, every variable in C++ is placed in **stack memory.**

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

```cpp
#include <iostream>

void foo() {
  int x = 42;
  std::cout << " x in foo(): " << x << std::endl;
  std::cout << "&x in foo(): " << &x << std::endl;
}

int main() {
  int num = 7;
  std::cout << " num in main(): " << num << std::endl;
  std::cout << "&num in main(): " << &num << std::endl;

  foo();

  return 0;
}
```

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.

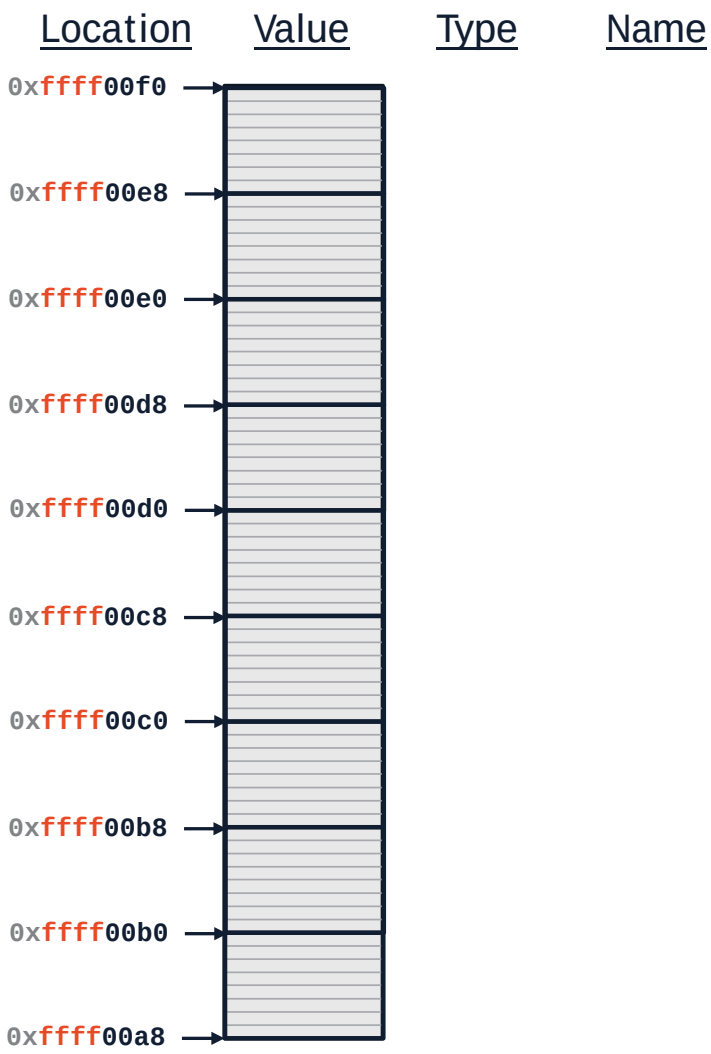
- Integer pointer: `int * p = &num;`

Type for Pointer

Dereference Operator

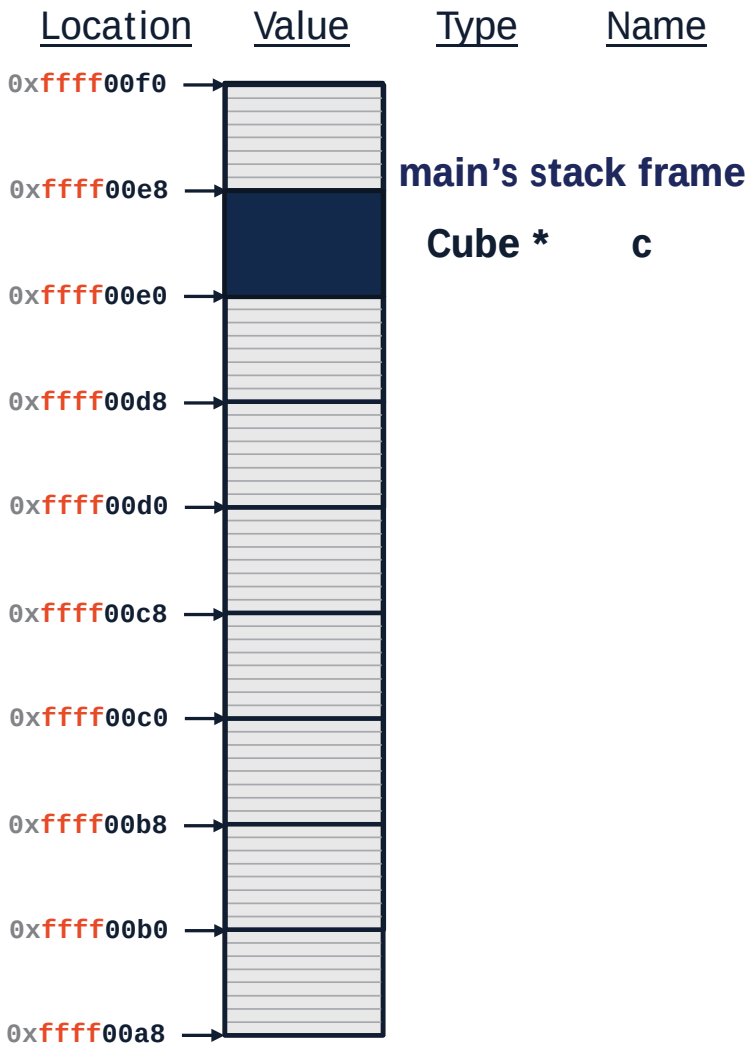
Given a pointer, a level of indirection can be ren
the dereference operator `*`.

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



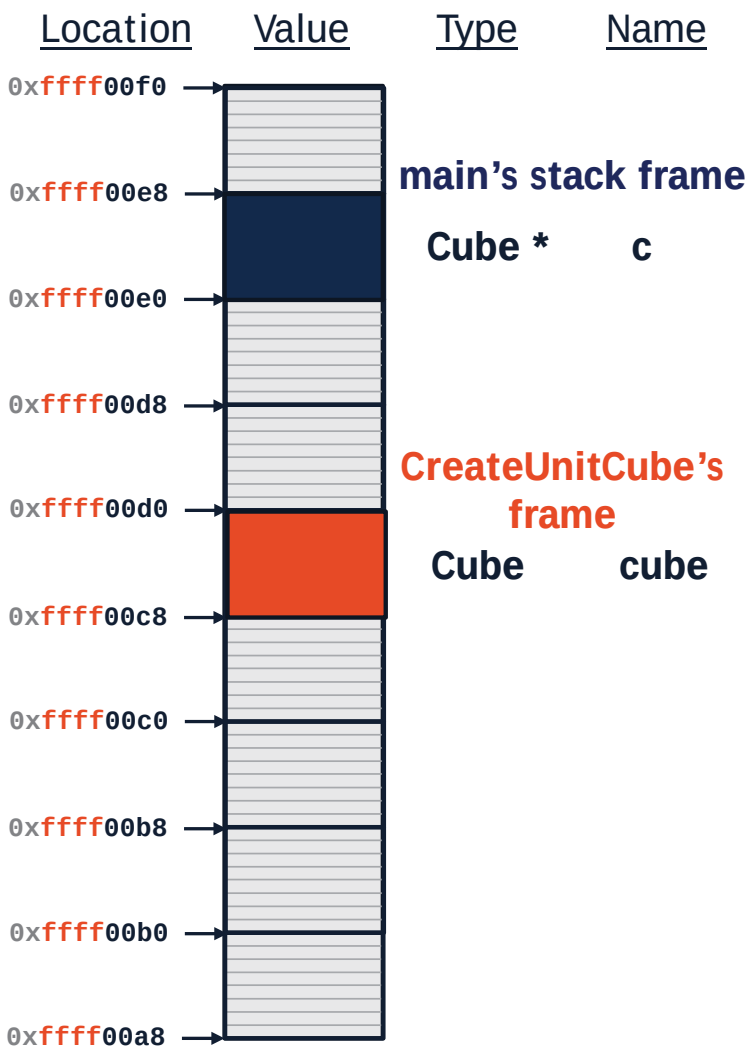
puzzle.cpp

```
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 }
```



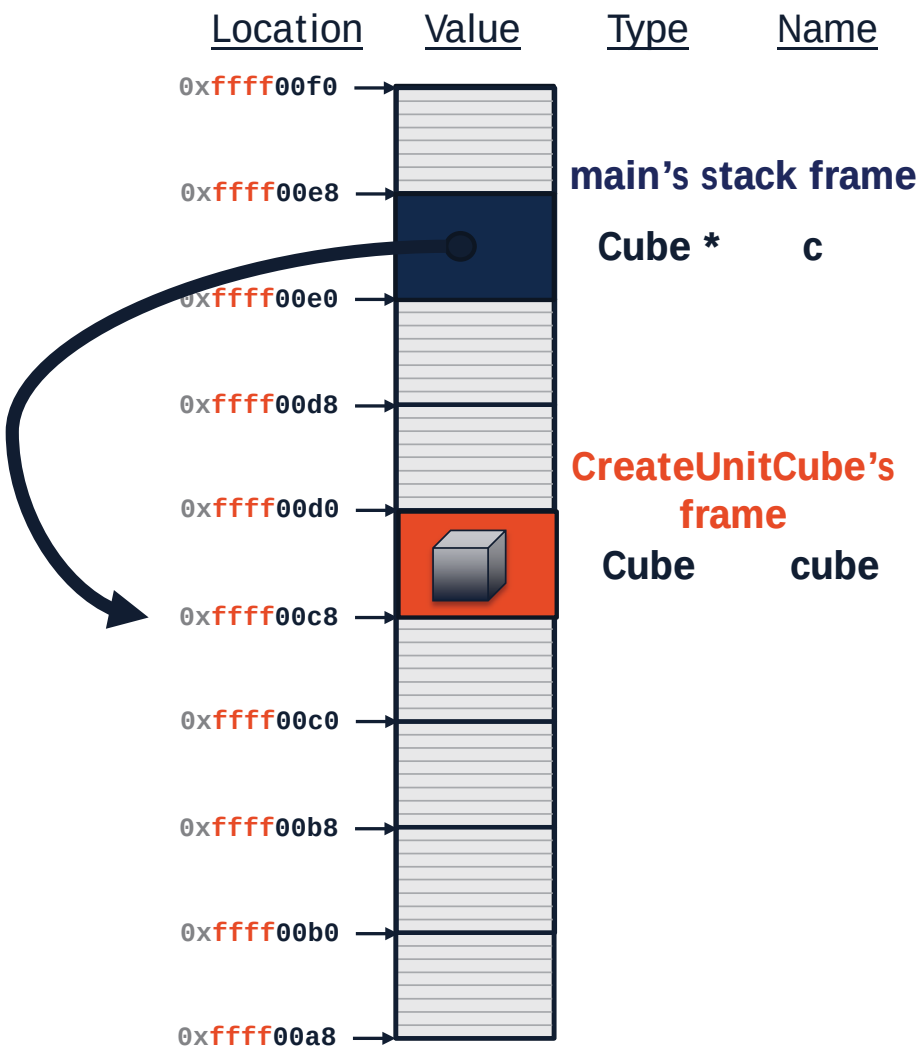
puzzle.cpp

```
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 }
```



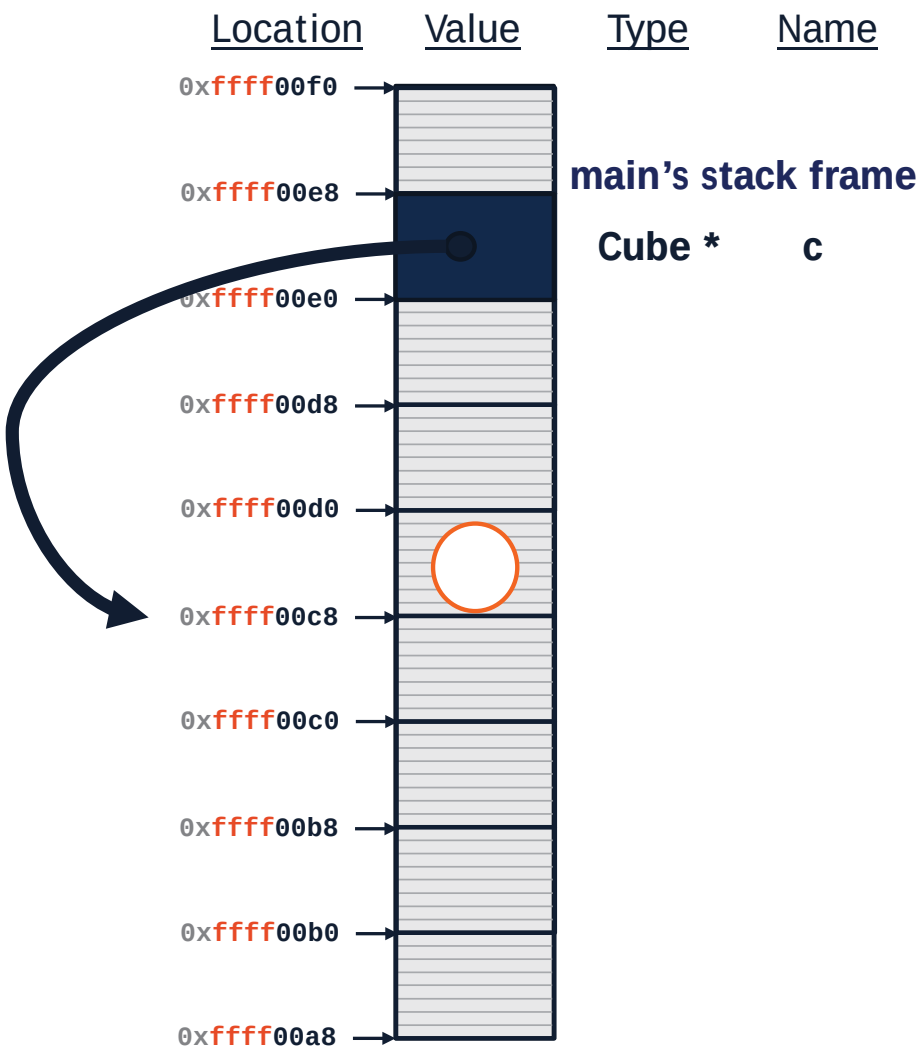
puzzle.cpp

```
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 }
```



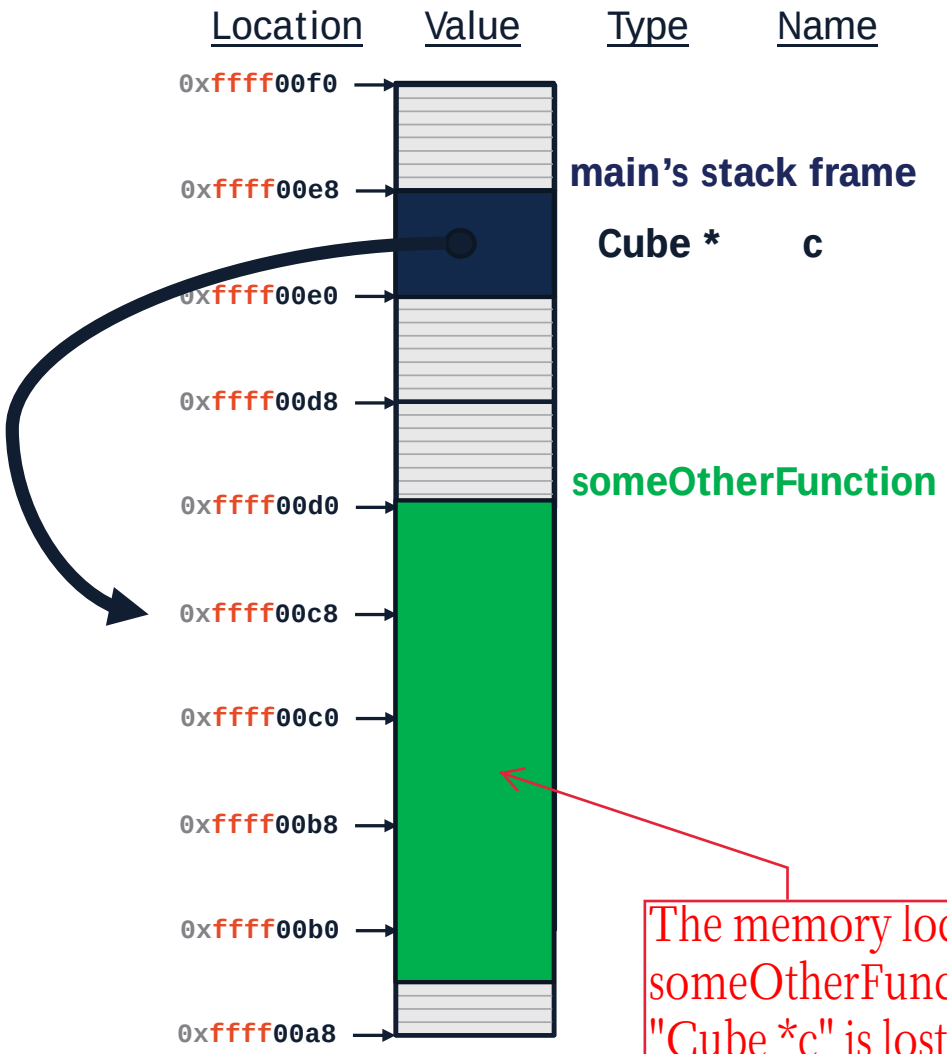
puzzle.cpp

```
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

```
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 }
```

The memory location for "Cube *c" is reused by someOtherFunction and is overwritten. The information of "Cube *c" is lost.
Never return a reference of local variable.

cpp-memory/main.cpp

```cpp
int main() {
  int num = 7;
  std::cout << " num: " << num << std::endl;
  std::cout << "&num: " << &num << std::endl;

  int *p = &num;
  std::cout << " p: " << p << std::endl;
  std::cout << "&p: " << &p << std::endl;
  std::cout << "*p: " << *p << std::endl;

  *p = 42;
  std::cout << "*p changed to 42" << std::endl;
  std::cout << " num: " << num << std::endl;

  return 0;
}
```