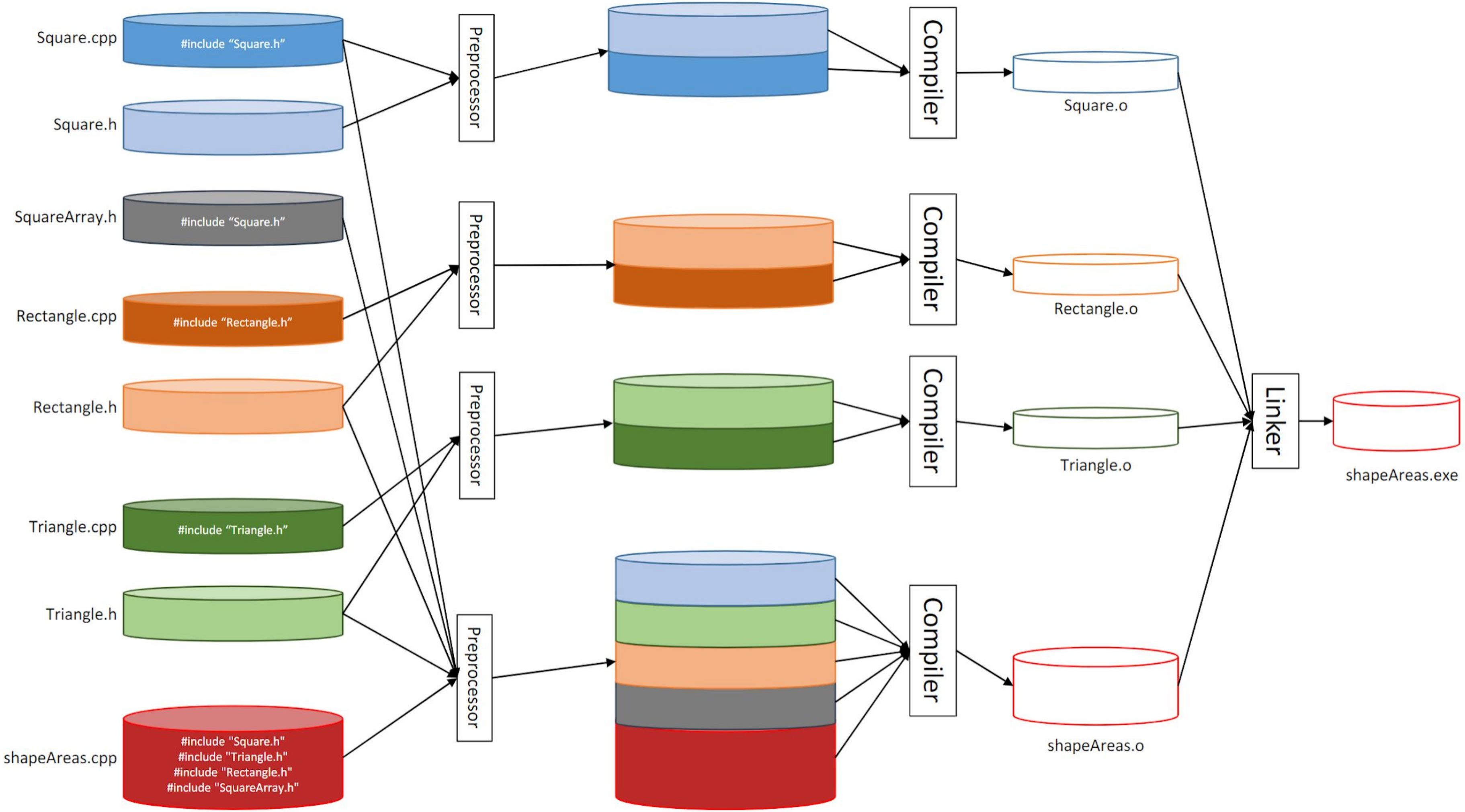
Discussion 7!

Multiple Source Files, Makefiles, Pointers

Multiple Header/Source Files



Square.cpp

#include “Square.h”

Square.h

SquareArray.h

#include “Square.h”

Rectangle.cpp

#include “Rectangle.h”

Rectangle.h

Triangle.cpp

#include “Triangle.h”

Triangle.h

shapeAreas.cpp

#include "Square.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "SquareArray.h"

Preprocessor

Preprocessor

Preprocessor

Preprocessor

Compiler

Compiler

Compiler

Compiler

Square.o

Rectangle.o

Triangle.o

shapeAreas.o

Linker

shapeAreas.exe

Troubleshooting

Don’t violate the one-definition rule!

- Make sure #ifndef name is different for all .h files
- Do not #include .cpp files!!

Makefiles

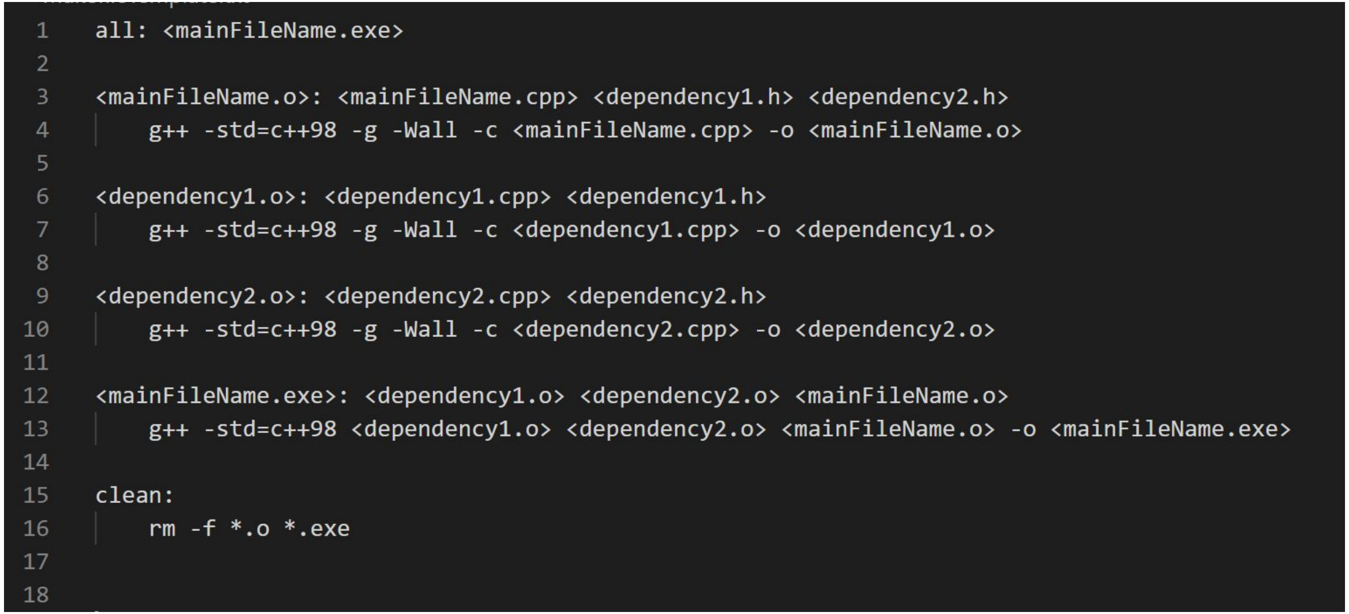
Makefile Example

```
git clone https://github.com/emolson16/makefile_example
```

```
cd makefile_example
```

https://drive.google.com/file/d/1_CRG_bVNU67dA3fyhrnueH-Y_ExIEo0X/view?usp=sharing

Makefile Template

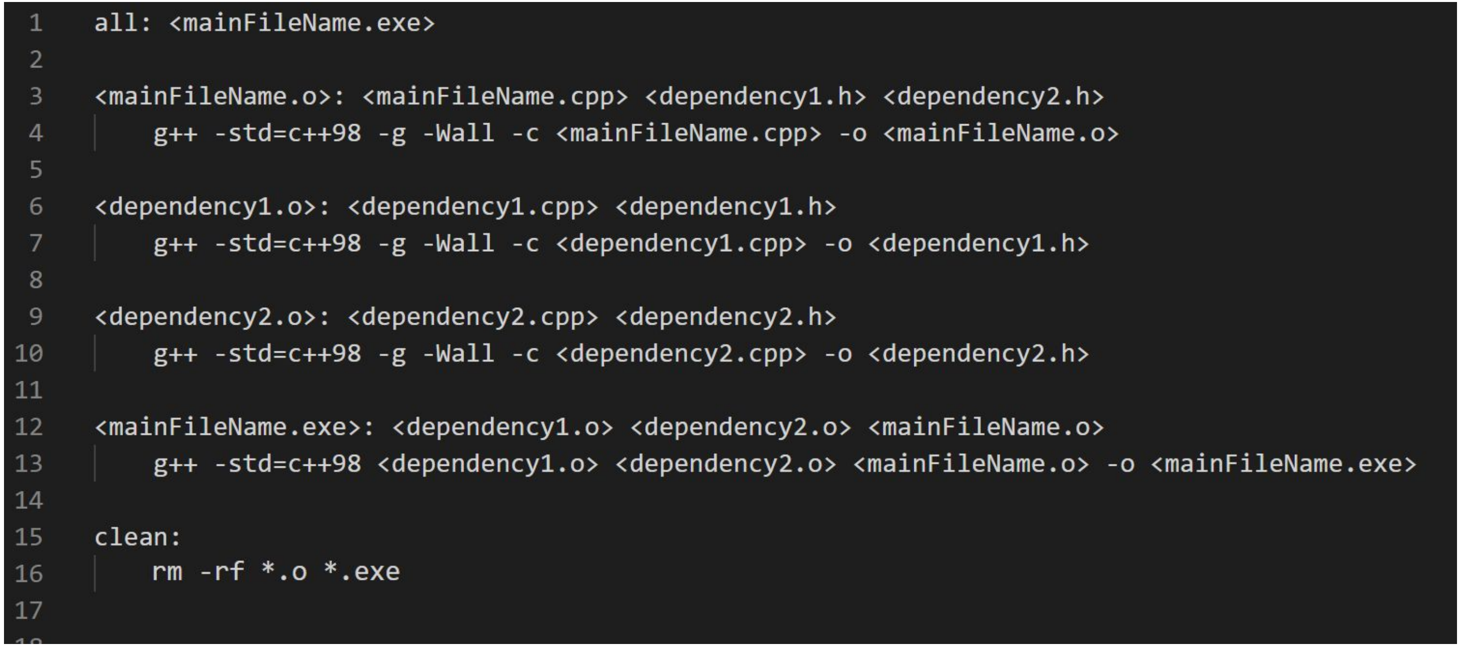


```
all: <mainFileName.exe>

<mainFileName.o>: <mainFileName.cpp> <dependency1.h> <dependency2.h>
    g++ -std=c++98 -g -Wall -c <mainFileName.cpp> -o <mainFileName.o>

<dependency1.o>: <dependency1.cpp> <dependency1.h>
    g++ -std=c++98 -g -Wall -c <dependency1.cpp> -o <dependency1.o>

<dependency2.o>: <dependency2.cpp> <dependency2.h>
    g++ -std=c++98 -g -Wall -c <dependency2.cpp> -o <dependency2.o>

<mainFileName.exe>: <dependency1.o> <dependency2.o> <mainFileName.o>
    g++ -std=c++98 <dependency1.o> <dependency2.o> <mainFileName.o> -o <mainFileName.exe>

clean:
    rm -f *.o *.exe


```

Link to txt: https://drive.google.com/file/d/1nW1MSl15REkLqJNf4Bwm3uJwTFViCm2V/view?usp=sharing

Debugging with Makefile

As mentioned before, if you want to debug, you have to add the ‘-g’ tag shown below



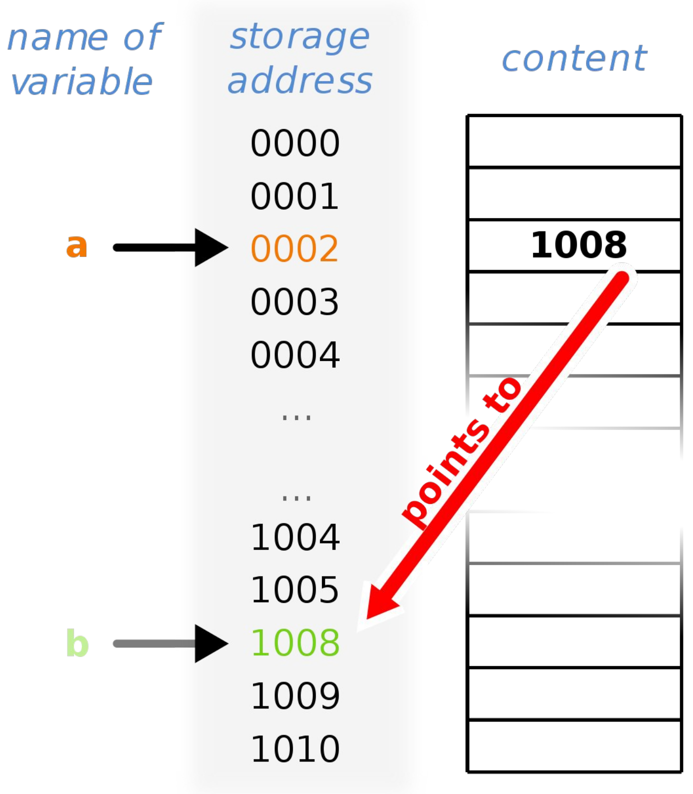
```
all: <mainFileName.exe>

<mainFileName.o>: <mainFileName.cpp> <dependency1.h> <dependency2.h>
    g++ -std=c++98 -g -Wall -c <mainFileName.cpp> -o <mainFileName.o>

<dependency1.o>: <dependency1.cpp> <dependency1.h>
    g++ -std=c++98 -g -Wall -c <dependency1.cpp> -o <dependency1.h>

<dependency2.o>: <dependency2.cpp> <dependency2.h>
    g++ -std=c++98 -g -Wall -c <dependency2.cpp> -o <dependency2.h>

<mainFileName.exe>: <dependency1.o> <dependency2.o> <mainFileName.o>
    g++ -std=c++98 <dependency1.o> <dependency2.o> <mainFileName.o> -o <mainFileName.exe>

clean:
    rm -rf *.o *.exe
```

Pointers

Pointers

Pointers store memory addresses

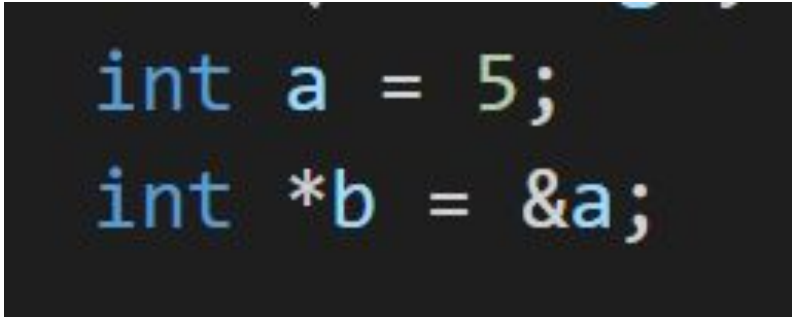
They “point” to an object in memory



Pointers

This declares ‘b’ to be a pointer to ‘a’

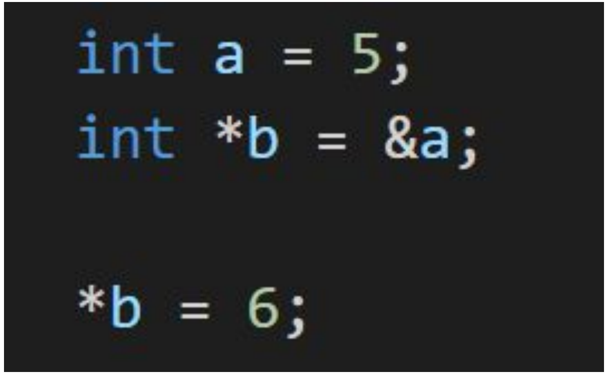
It stores a’s address



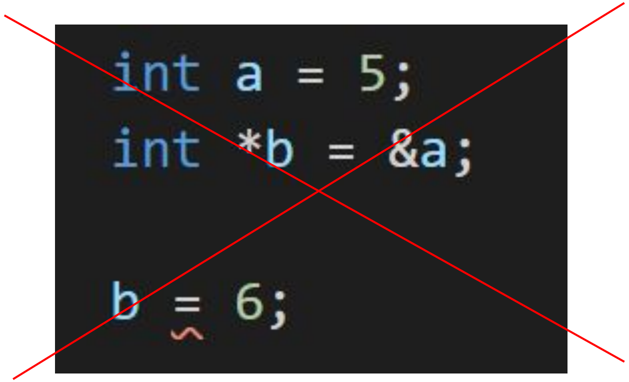
```
int a = 5;
int *b = &a;
```

Pointers

This sets a to be 6

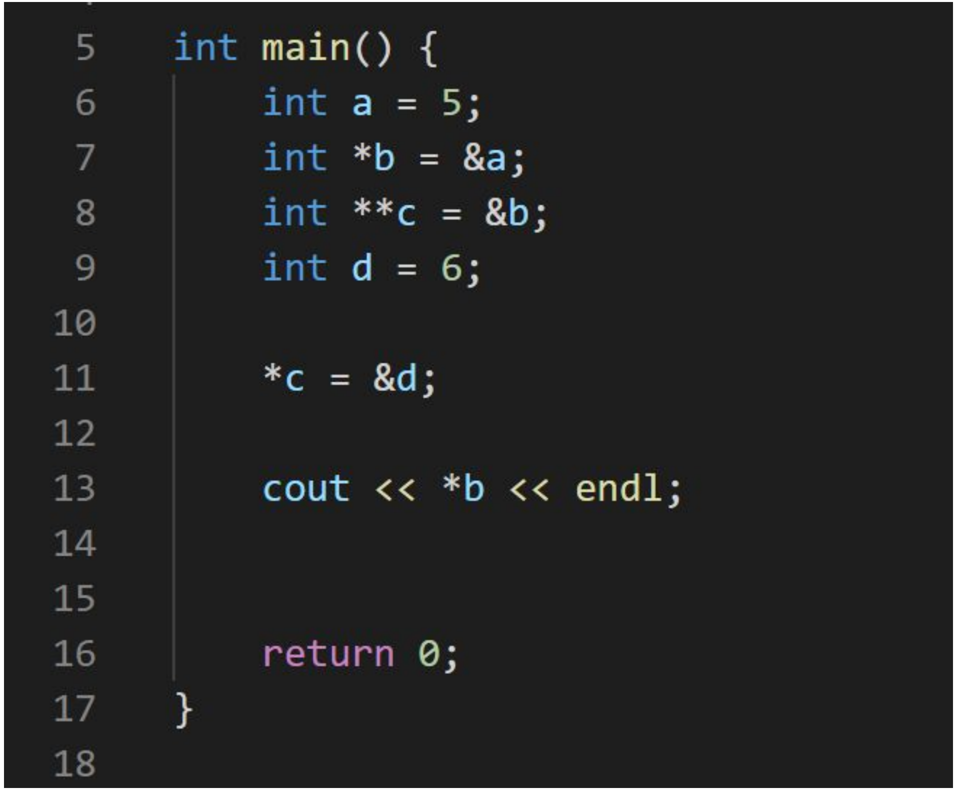


```
int a = 5;
int *b = &a;

*b = 6;
```



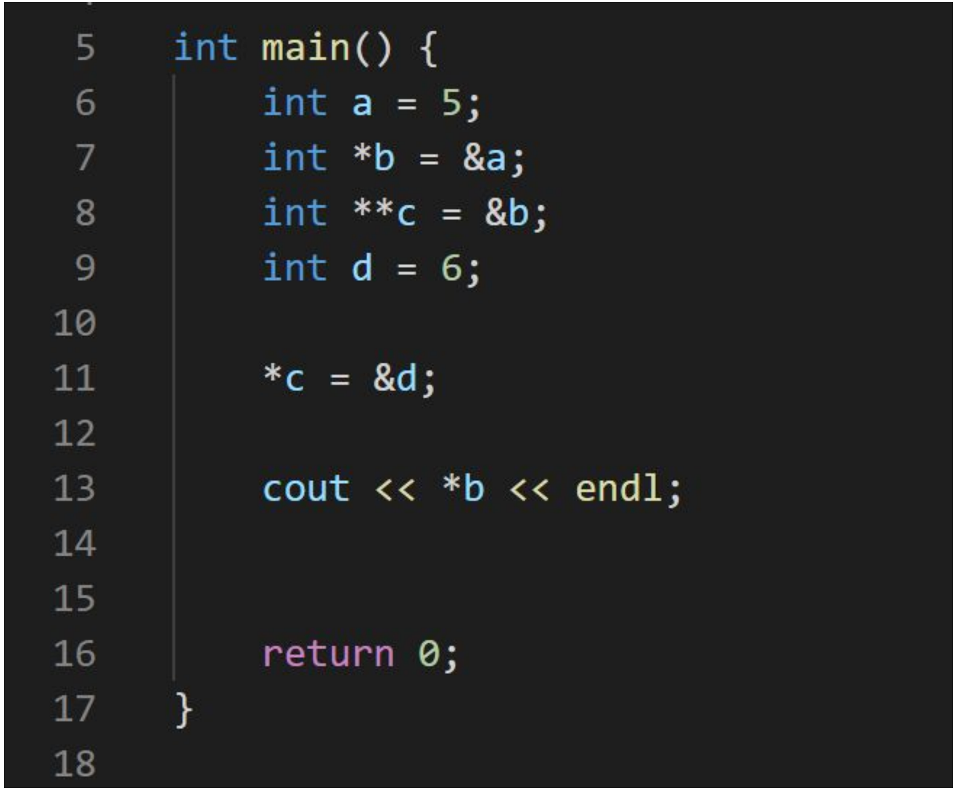
```
int a = 5;
int *b = &a;

b = 6;
```

What would this output?



```cpp
int main() {
    int a = 5;
    int *b = &a;
    int **c = &b;
    int d = 6;

    *c = &d;

    cout << *b << endl;


    return 0;
}

```

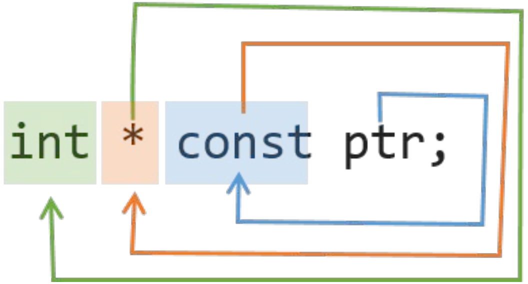
What would this output?



```cpp
int main() {
    int a = 5;
    int *b = &a;
    int **c = &b;
    int d = 6;

    *c = &d;

    cout << *b << endl;


    return 0;
}
```

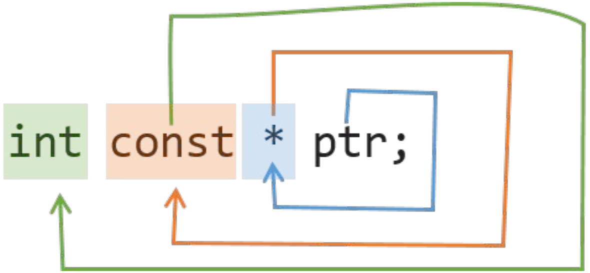
6

Pointers and Const



```
int * const ptr;
```

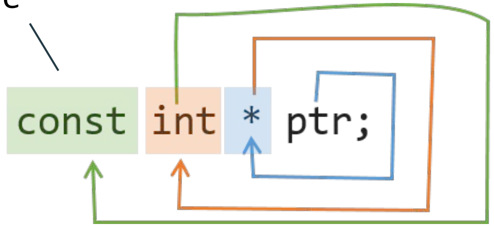
ptr is a const pointer to int



```
int const * ptr;
```

ptr is a pointer to const int

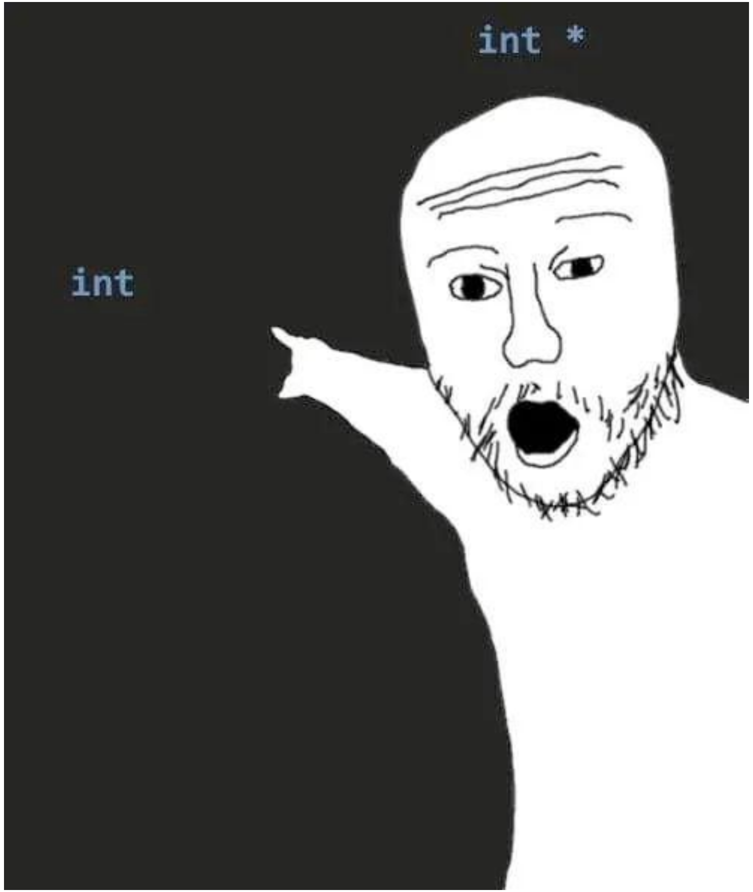
These
are the
same

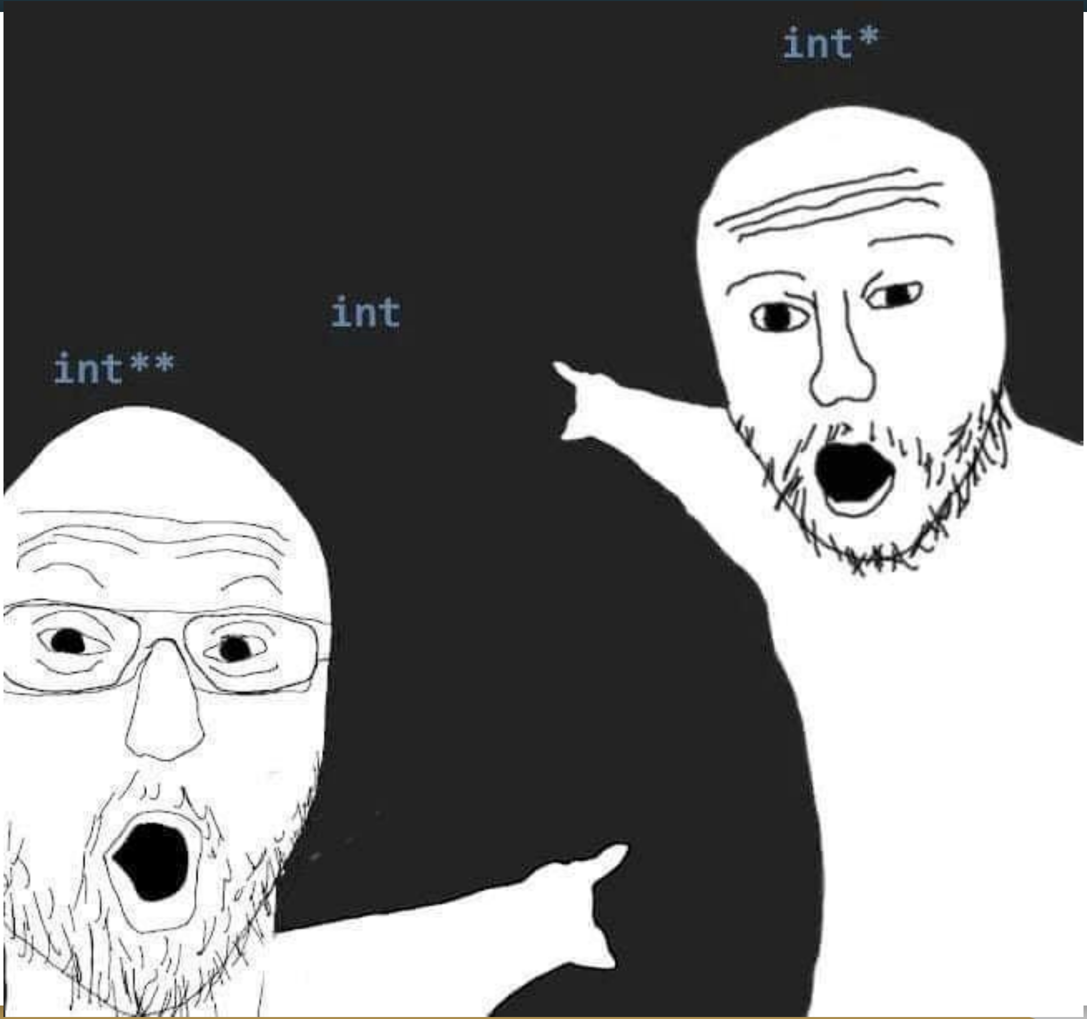


```
const int * ptr;
```

ptr is a pointer to int constant (i.e. const int)

And now some memes,
for your entertainment...


int *
int


int*
int
int**

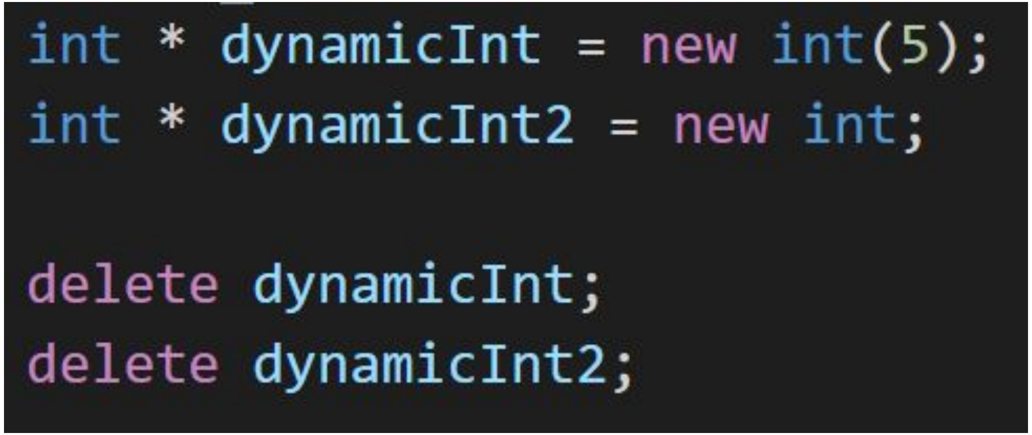
MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Dynamic Allocation

Dynamic Primitive Types

Creates a pointer on the stack

Creates new int on the heap

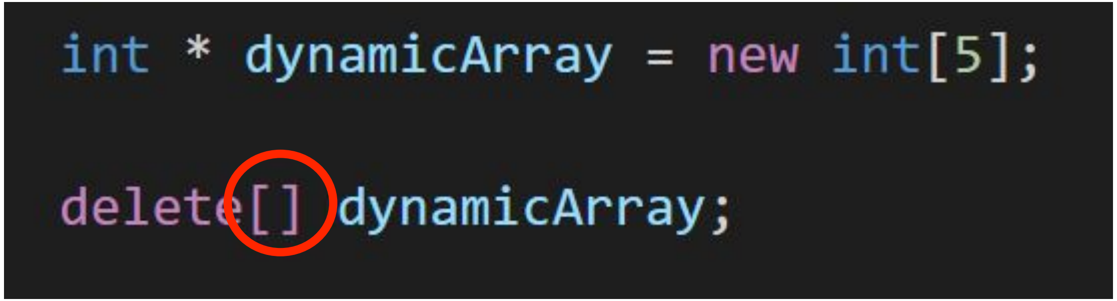


```
int * dynamicInt = new int(5);
int * dynamicInt2 = new int;

delete dynamicInt;
delete dynamicInt2;
```

Dynamic Arrays

Creates a pointer on the stack

Creates array of size 5 on the heap



```
int * dynamicArray = new int[5];

delete[] dynamicArray;
```

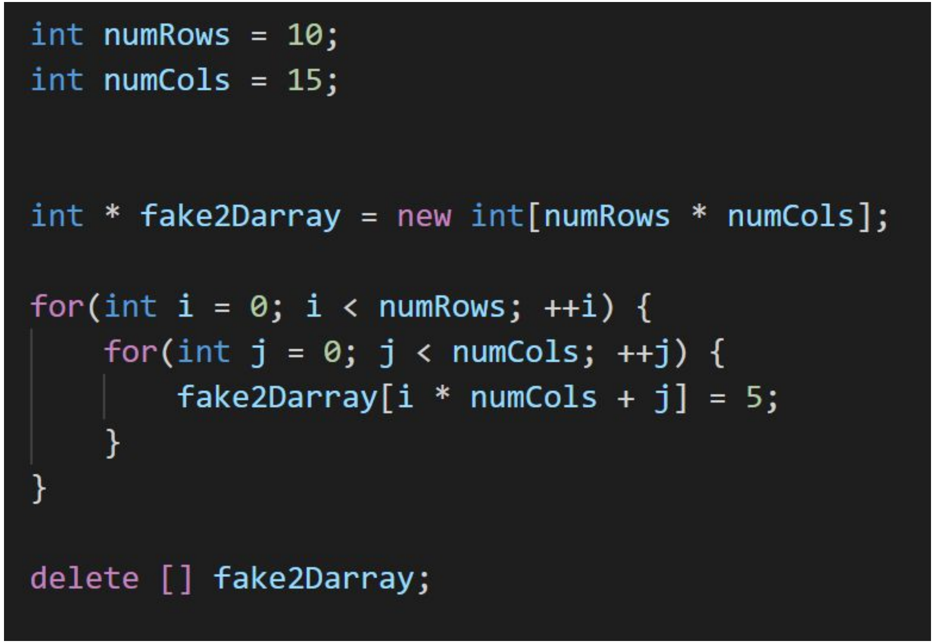
Dynamic 2D Arrays Take 1

Do the math yourself

Declaring is easy- use 1D array

Everytime you index into, you have to do math

Deleting is easy



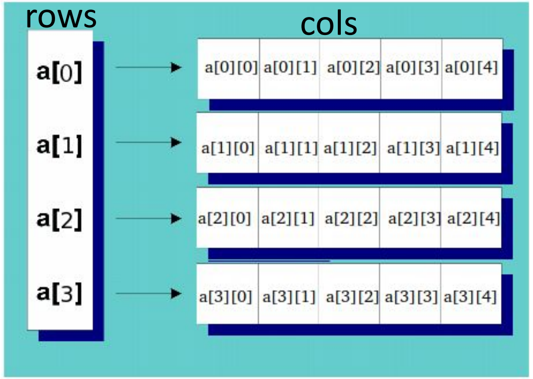
```
int numRows = 10;
int numCols = 15;


int * fake2Darray = new int[numRows * numCols];

for(int i = 0; i < numRows; ++i) {
    for(int j = 0; j < numCols; ++j) {
        fake2Darray[i * numCols + j] = 5;
    }
}

delete [] fake2Darray;
```

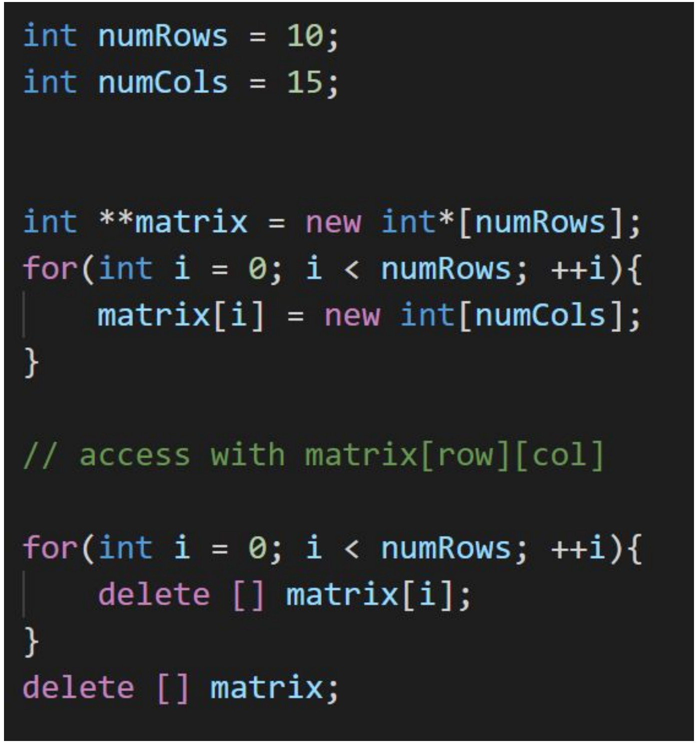
Dynamic 2D Arrays Take 2

Create array of arrays

More difficult to create and delete

Easy to index into


rows
cols
a[0]
a[1]
a[2]
a[3]
a[0][0] a[0][1] a[0][2] a[0][3] a[0][4]
a[1][0] a[1][1] a[1][2] a[1][3] a[1][4]
a[2][0] a[2][1] a[2][2] a[2][3] a[2][4]
a[3][0] a[3][1] a[3][2] a[3][3] a[3][4]



```
int numRows = 10;
int numCols = 15;


int **matrix = new int*[numRows];
for(int i = 0; i < numRows; ++i){
    matrix[i] = new int[numCols];
}

// access with matrix[row][col]

for(int i = 0; i < numRows; ++i){
    delete [] matrix[i];
}
delete [] matrix;
```

Heap hazards

- Memory leak
  - You forget to delete memory on the heap
- Orphaned memory
  - You lose the address of a heap object
- Double free
  - You delete heap memory multiple times
- Bad delete
  - Use delete with a pointer to a non-heap object
- Dangling pointers (style)
  - Keeping the address of a deleted heap object - set to null after