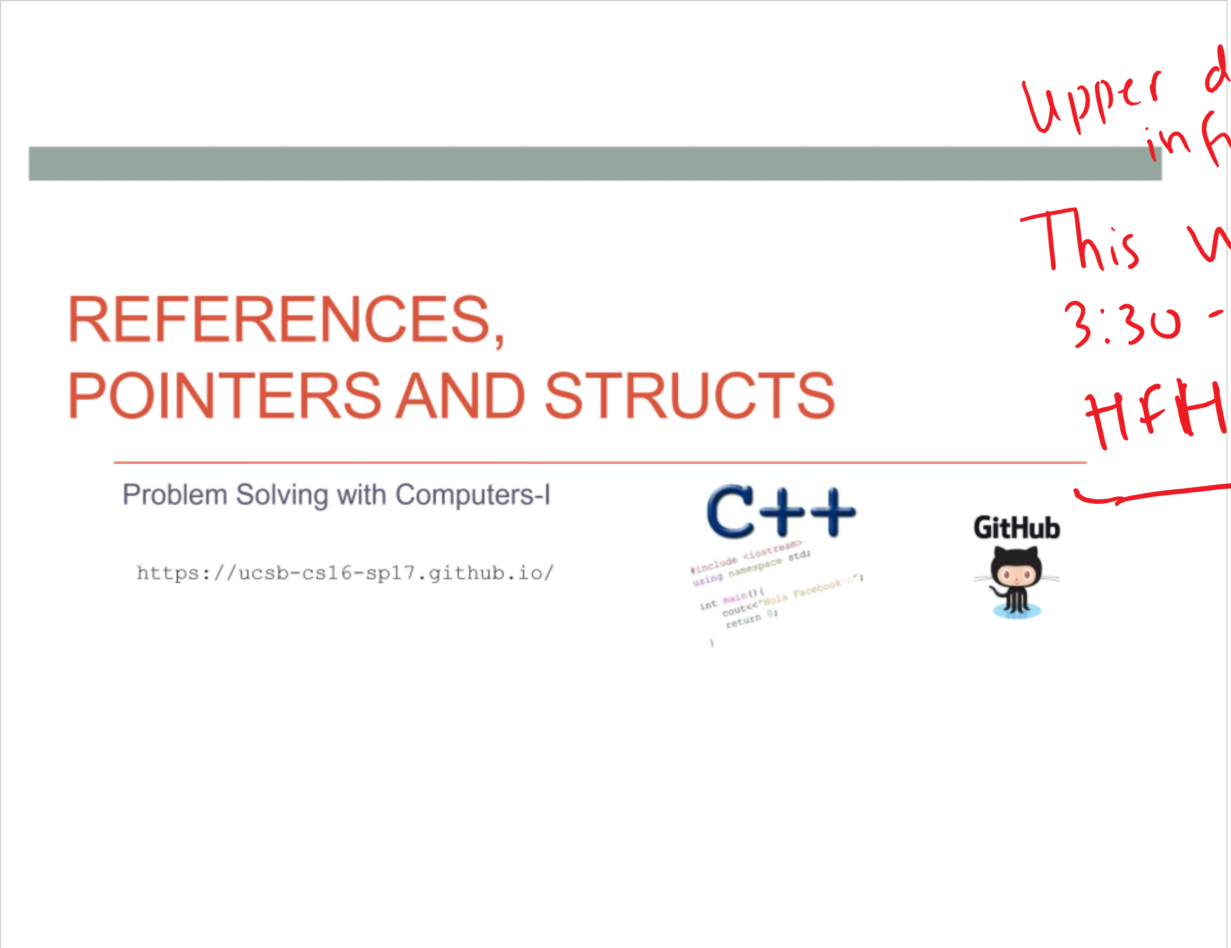
Lecture 10

Monday, May 8, 2017 12:47 PM



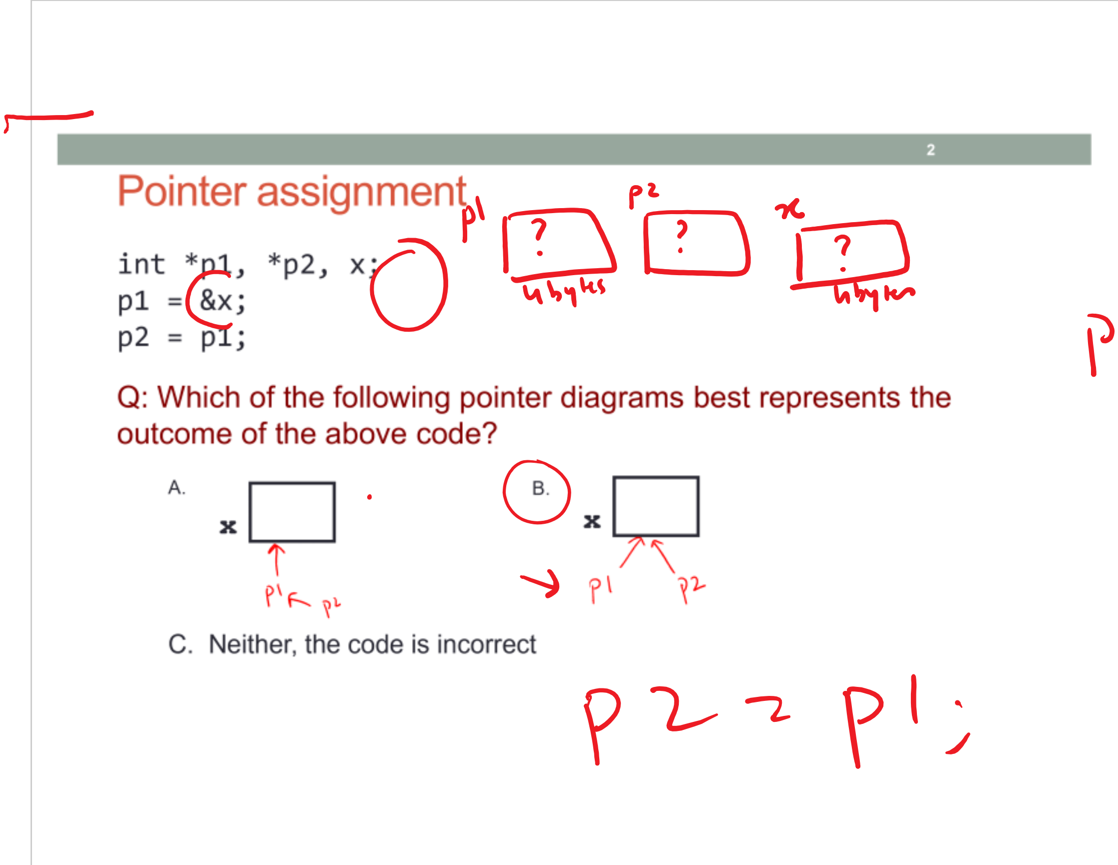
REFERENCES, POINTERS AND STRUCTS

Problem Solving with Computers-I

https://ucsb-cs16-sp17.github.io/

Upper div elective
info session

This Wed
3:30 - 5:00 pm
HFH 1132


2
p1
?
4 bytes
p2
?
x
?
4 bytes

Pointer assignment

```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A.

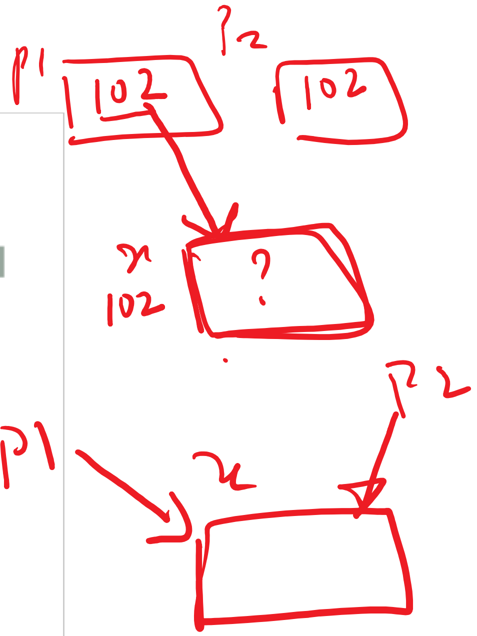
x
p1
p2

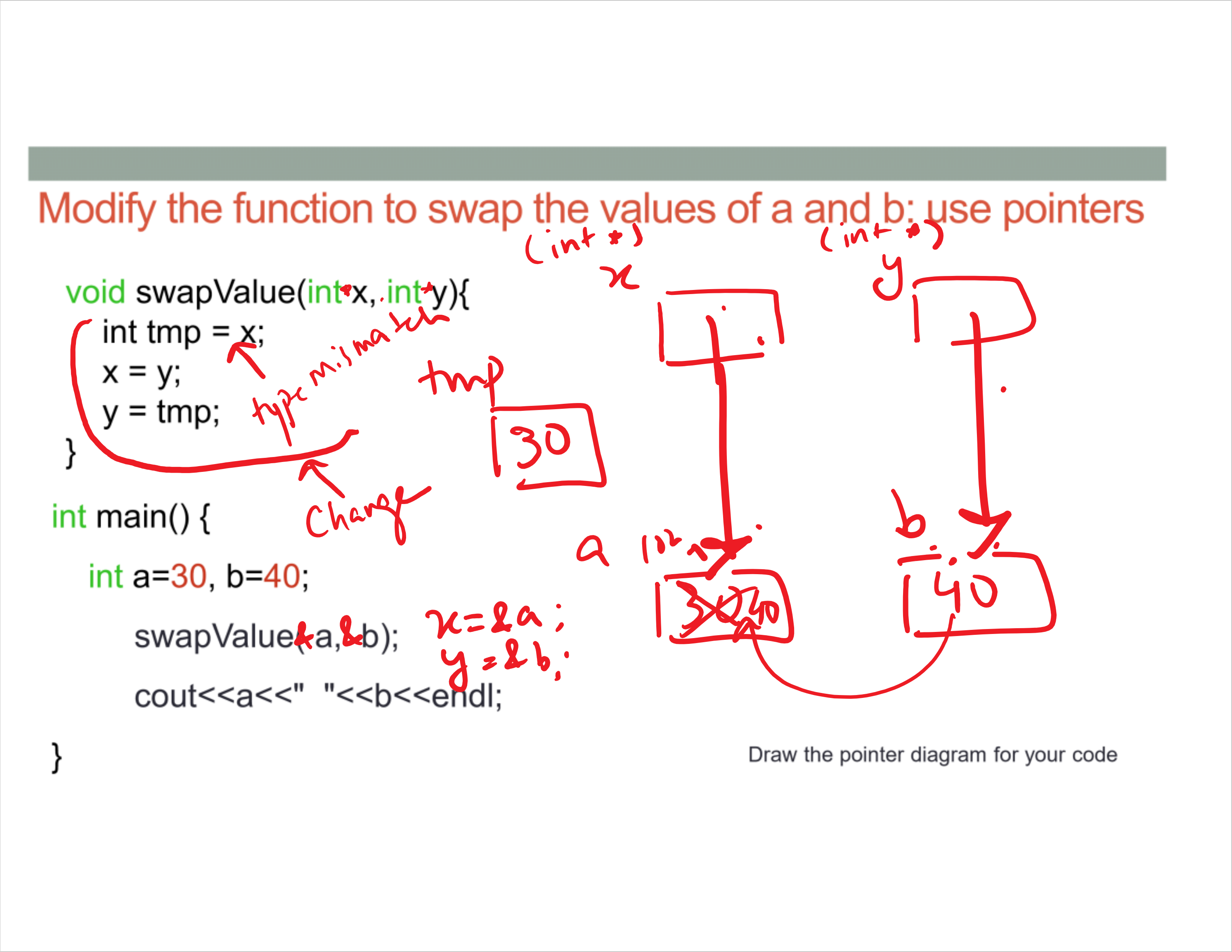
[✓ B.]

x
p1
p2

C. Neither, the code is incorrect

p2 = p1;





Modify the function to swap the values of a and b: use pointers

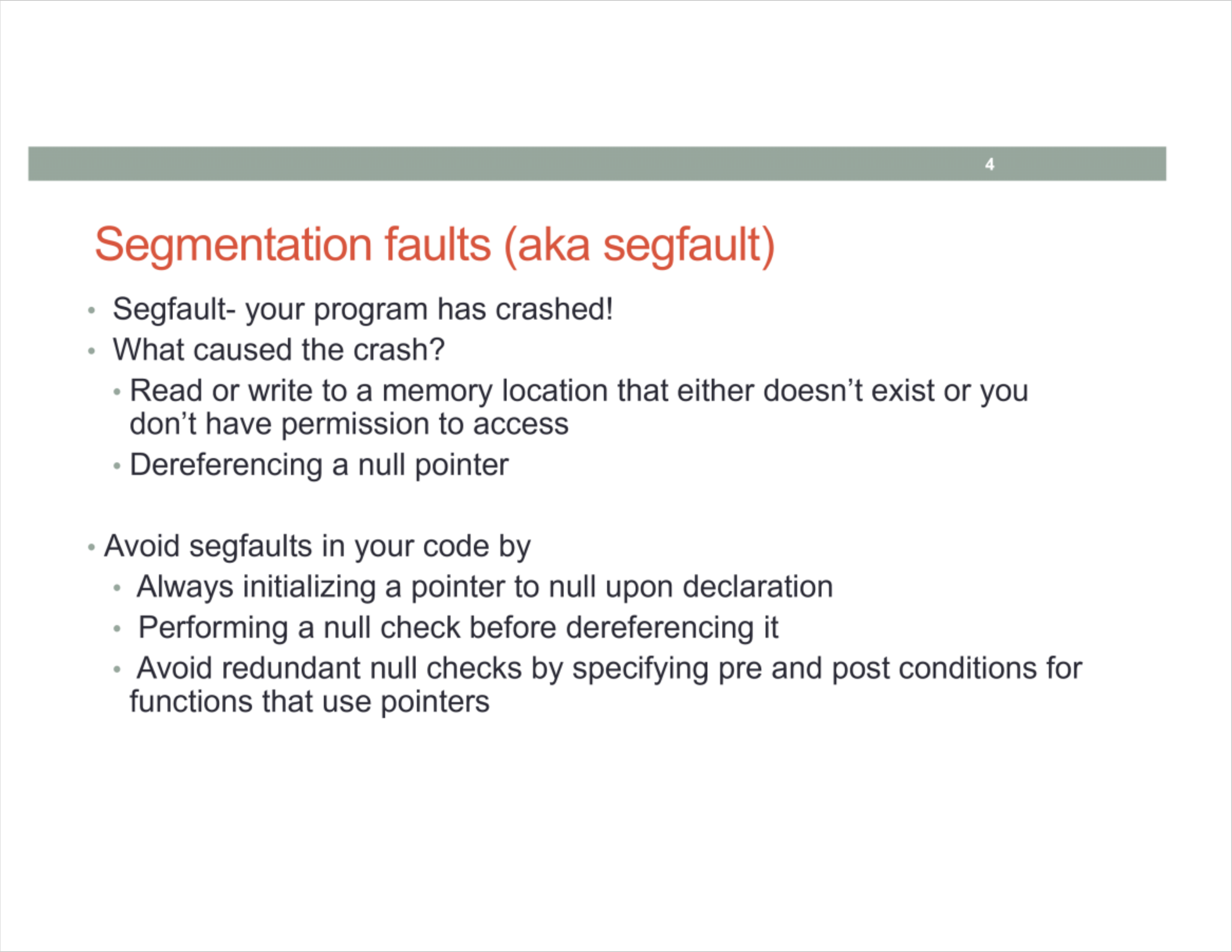
```
void swapValue(int*x, int*y){
  int tmp = x;
  x = y;
  y = tmp;
}

int main() {
   int a=30, b=40;
      swapValue(&a,&b);
      cout<<a<<"  "<<b<<endl;
}
```

x=&a;
y=&b;

Draw the pointer diagram for your code

Page 4

Monday, May 8, 2017 12:47 PM



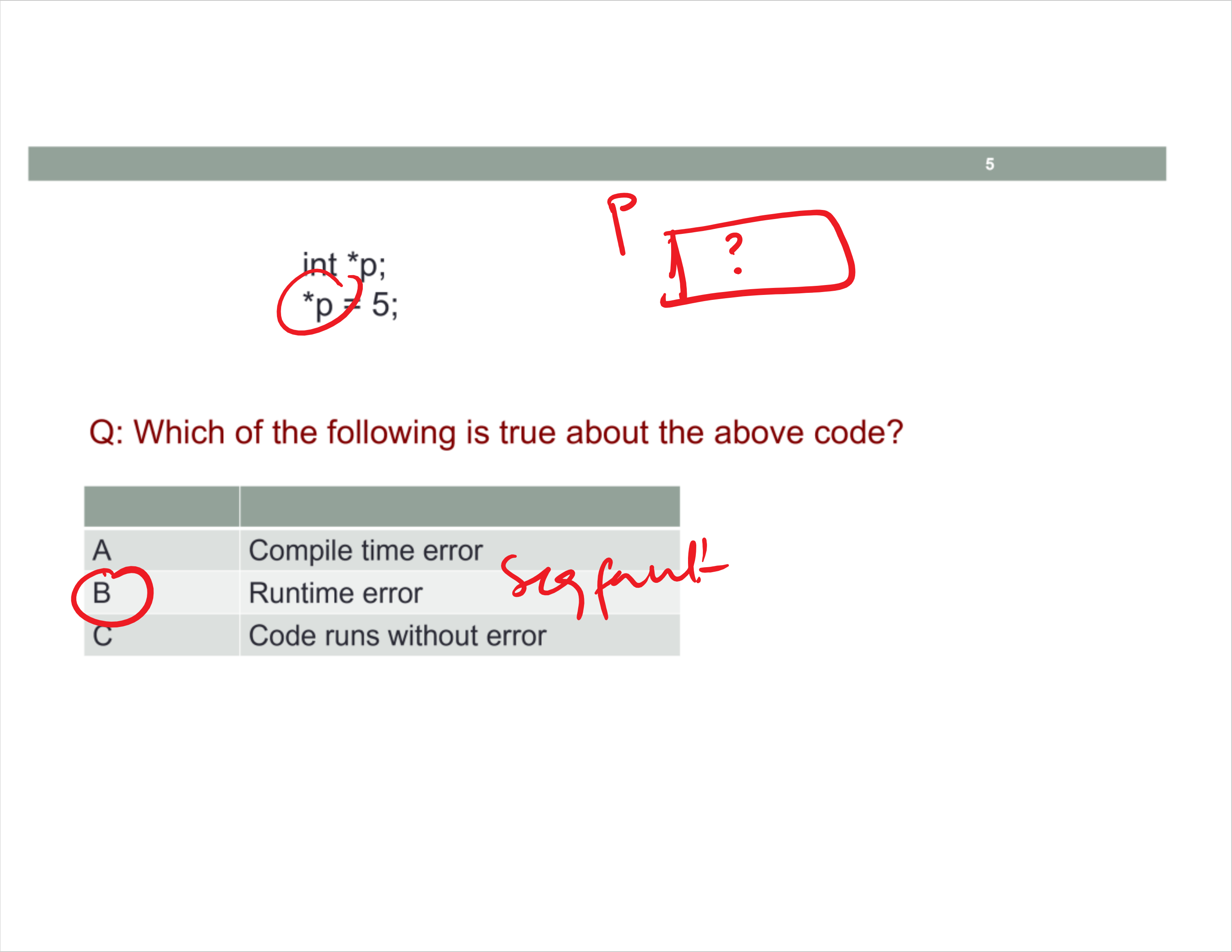
Segmentation faults (aka segfault)

- Segfault- your program has crashed!
- What caused the crash?
  - Read or write to a memory location that either doesn’t exist or you don’t have permission to access
  - Dereferencing a null pointer

- Avoid segfaults in your code by
  - Always initializing a pointer to null upon declaration
  - Performing a null check before dereferencing it
  - Avoid redundant null checks by specifying pre and post conditions for functions that use pointers

Page 5

Monday, May 8, 2017 12:47 PM



```
int *p;
*p = 5;
```

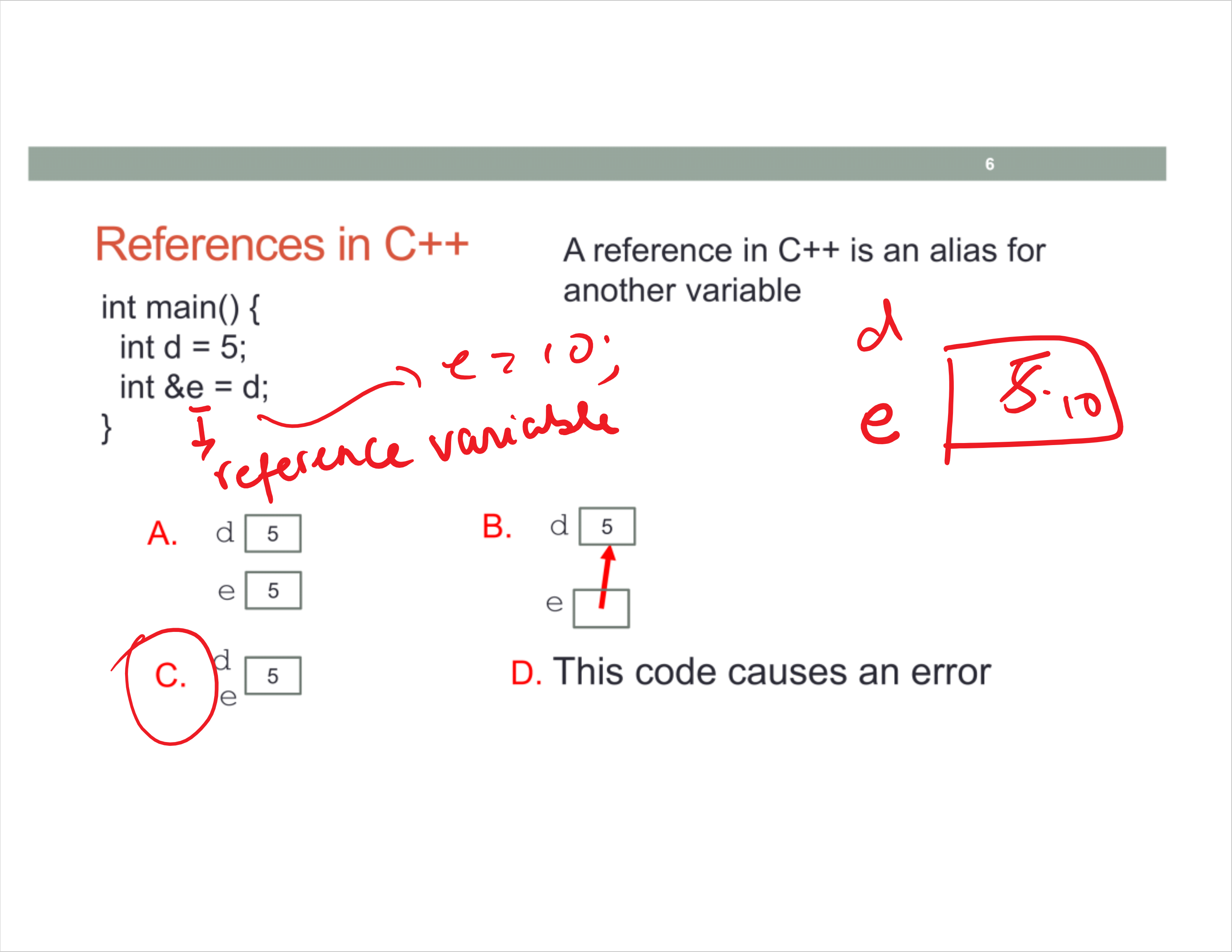
Q: Which of the following is true about the above code?

A	Compile time error
[✓ B]	Runtime error
C	Code runs without error

seg fault!

Page 6

Monday, May 8, 2017 12:47 PM



6

References in C++

```
int main() {
  int d = 5;
  int &e = d;
}
```

e = 10;

reference variable

A reference in C++ is an alias for another variable

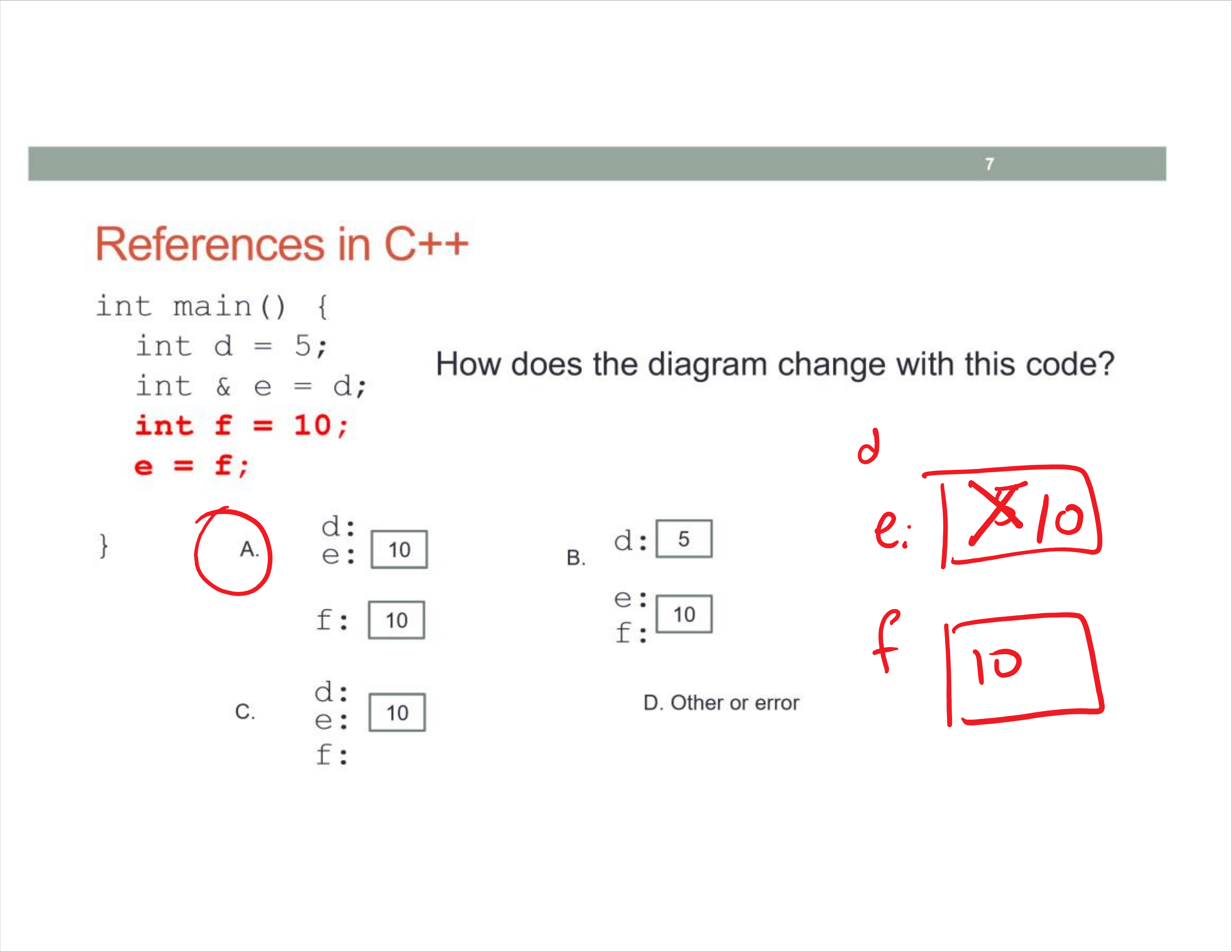
A.

B.

[✓ C.]

D. This code causes an error

Monday, May 8, 2017 12:47 PM



References in C++

```
int main() {
   int d = 5;
   int & e = d;
   int f = 10;
   e = f;

}
```

How does the diagram change with this code?

[✓ A.]

d:
e: 10

f: 10

B.

d: 5

e:
f: 10

C.

d:
e:
f: 10

D. Other or error

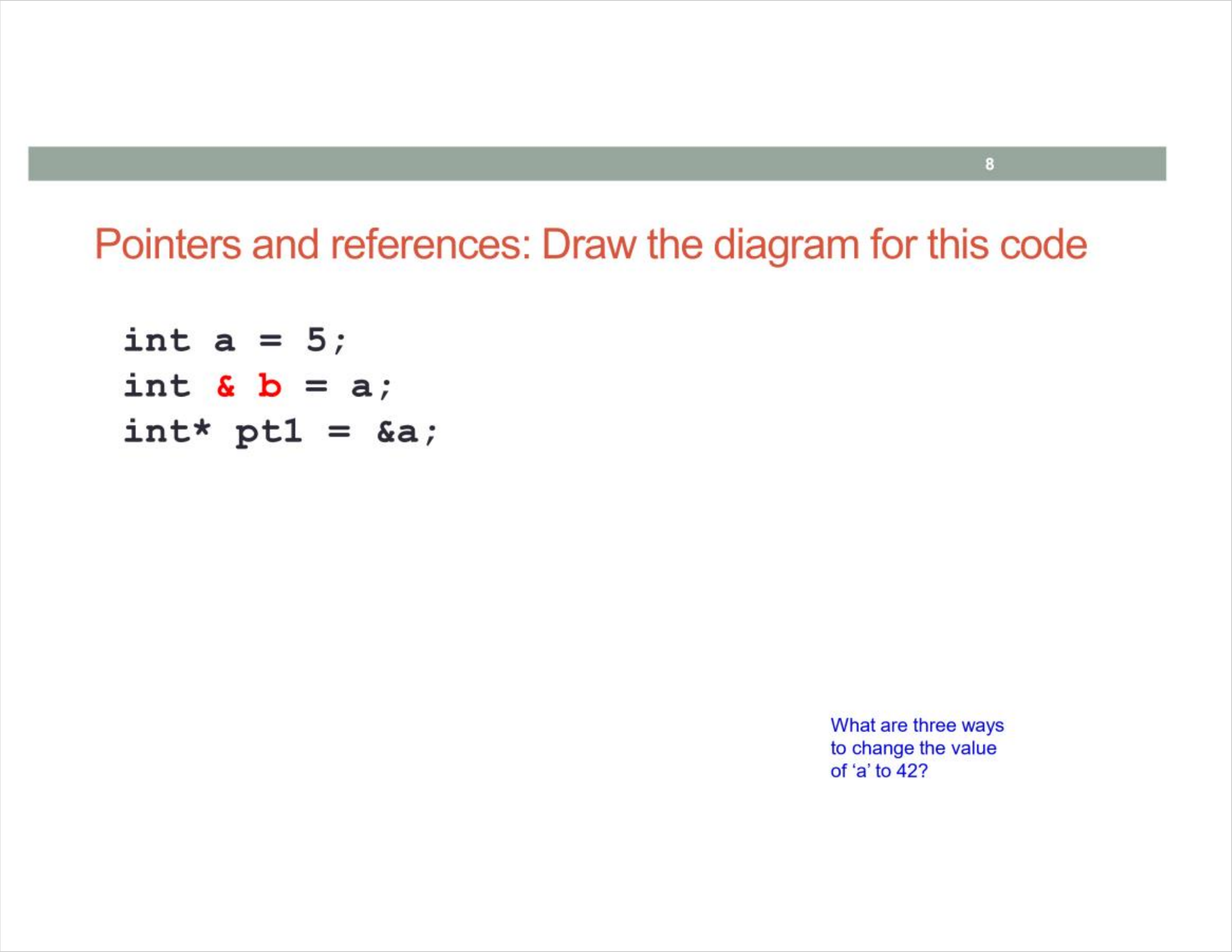
d

e: ~~5~~ 10

f 10

Page 8

Monday, May 8, 2017 12:47 PM



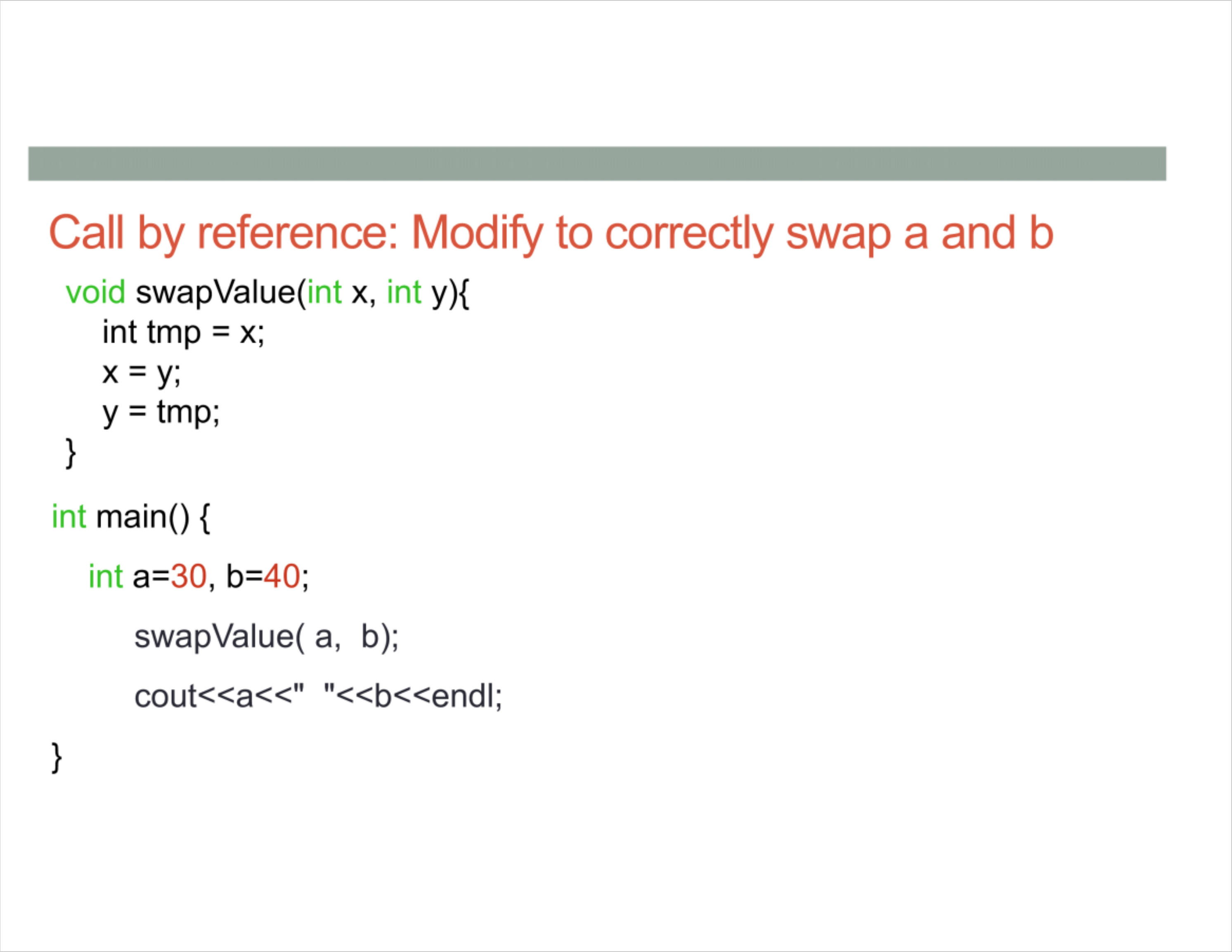
Pointers and references: Draw the diagram for this code

```
int a = 5;
int & b = a;
int* pt1 = &a;
```

What are three ways to change the value of ‘a’ to 42?

Page 9

Monday, May 8, 2017 12:47 PM



Call by reference: Modify to correctly swap a and b

```
 void swapValue(int x, int y){
    int tmp = x;
    x = y;
    y = tmp;
 }

int main() {

   int a=30, b=40;

       swapValue( a,  b);

       cout<<a<<"  "<<b<<endl;

}
```

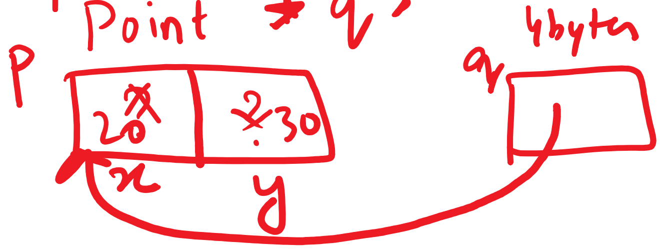
C++ structures

- A **struct** is a data structure composed of simpler data types.

```
struct Point {
    double x;
    double y;
};
```

```
struct Point {
    int x;
    double y;
}
```

```
Point p;
p.x = 100;
p.y = 50;
Point *q;
```



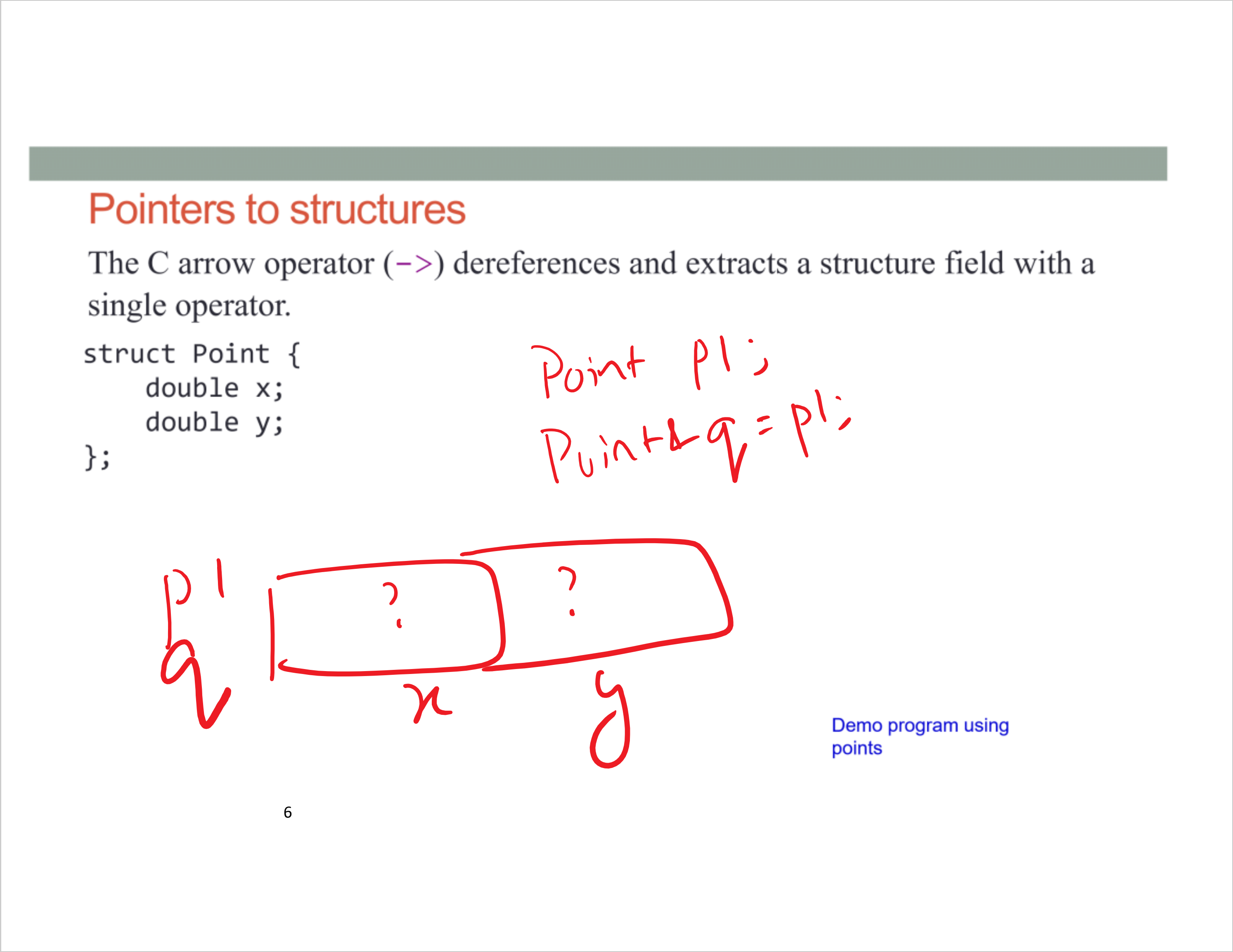
```
q = &p;
(*q).x = 20;
(*q).y = 30;
```

→

```
q->x=20;
q->y=30;
```

Page 11

Monday, May 8, 2017 12:47 PM



Pointers to structures

The C arrow operator (->) dereferences and extracts a structure field with a single operator.

```
struct Point {
    double x;
    double y;
};
```

Point p1;

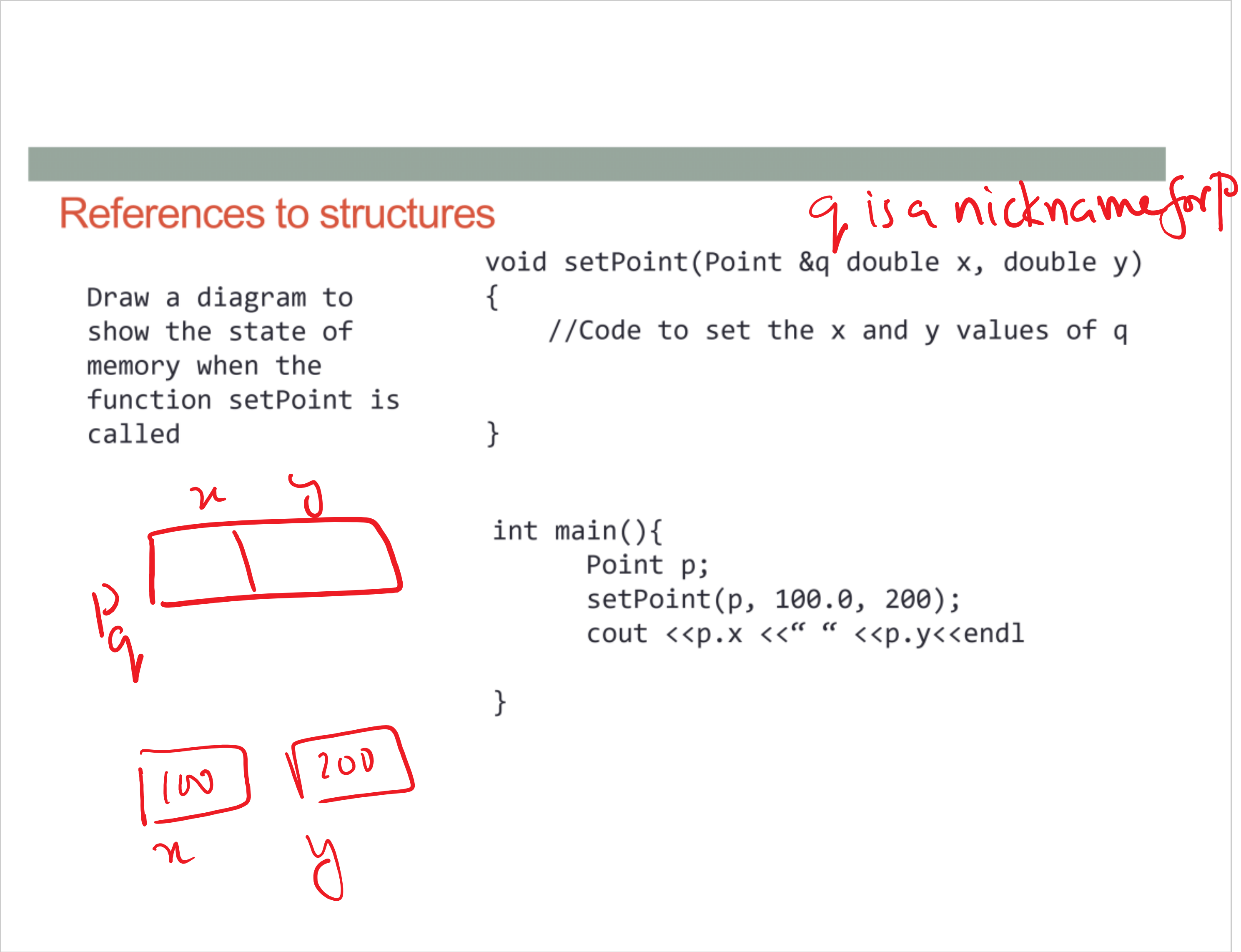
Point& q = p1;

Demo program using points

6

Page 12

Monday, May 8, 2017 12:47 PM



References to structures

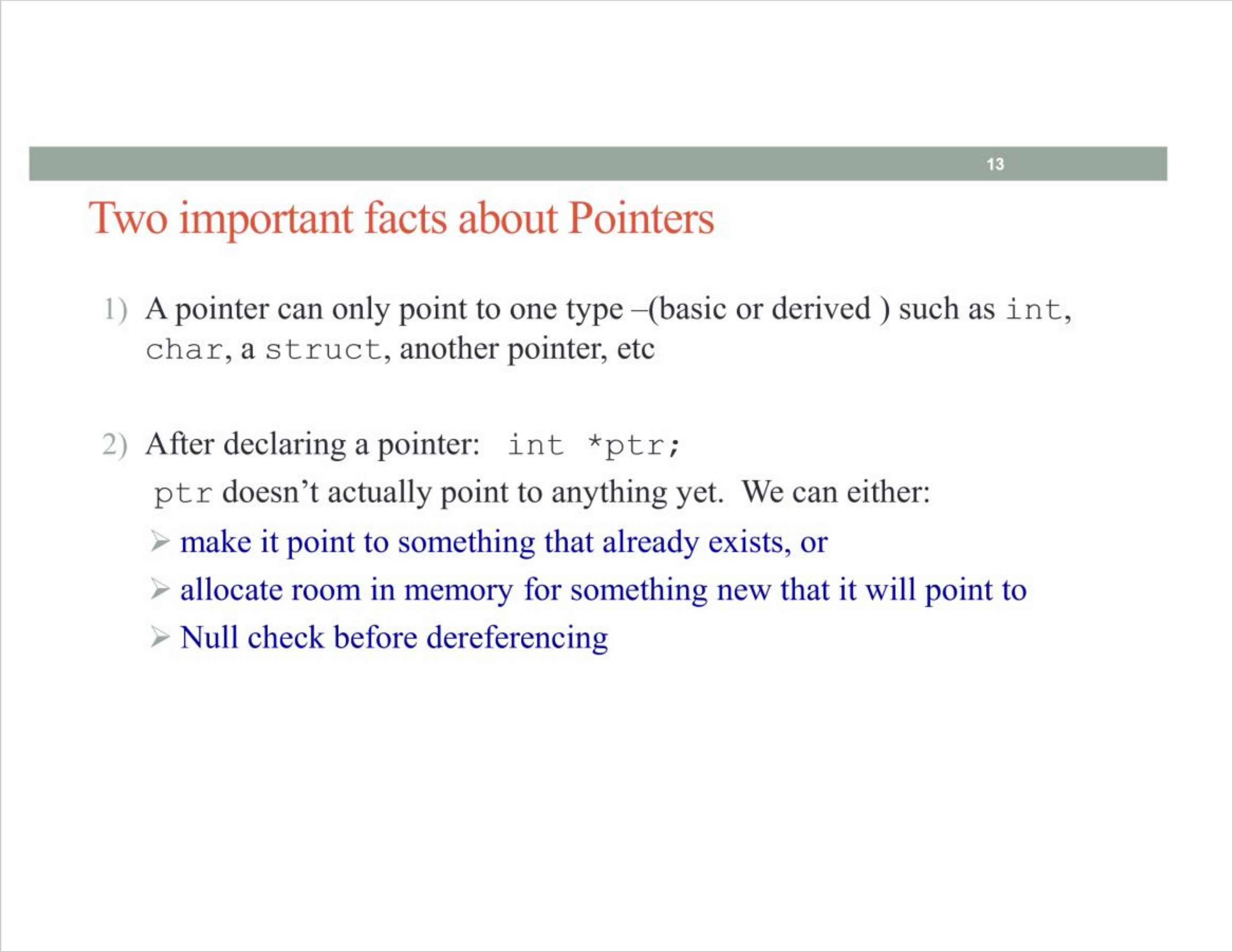
q is a nickname for p

Draw a diagram to show the state of memory when the function setPoint is called

```
void setPoint(Point &q double x, double y)
{
    //Code to set the x and y values of q


}


int main(){
      Point p;
      setPoint(p, 100.0, 200);
      cout <<p.x <<“ “ <<p.y<<endl

}
```

Page 13

Monday, May 8, 2017 12:47 PM



Two important facts about Pointers

1) A pointer can only point to one type –(basic or derived) such as `int`, `char`, a `struct`, another pointer, etc

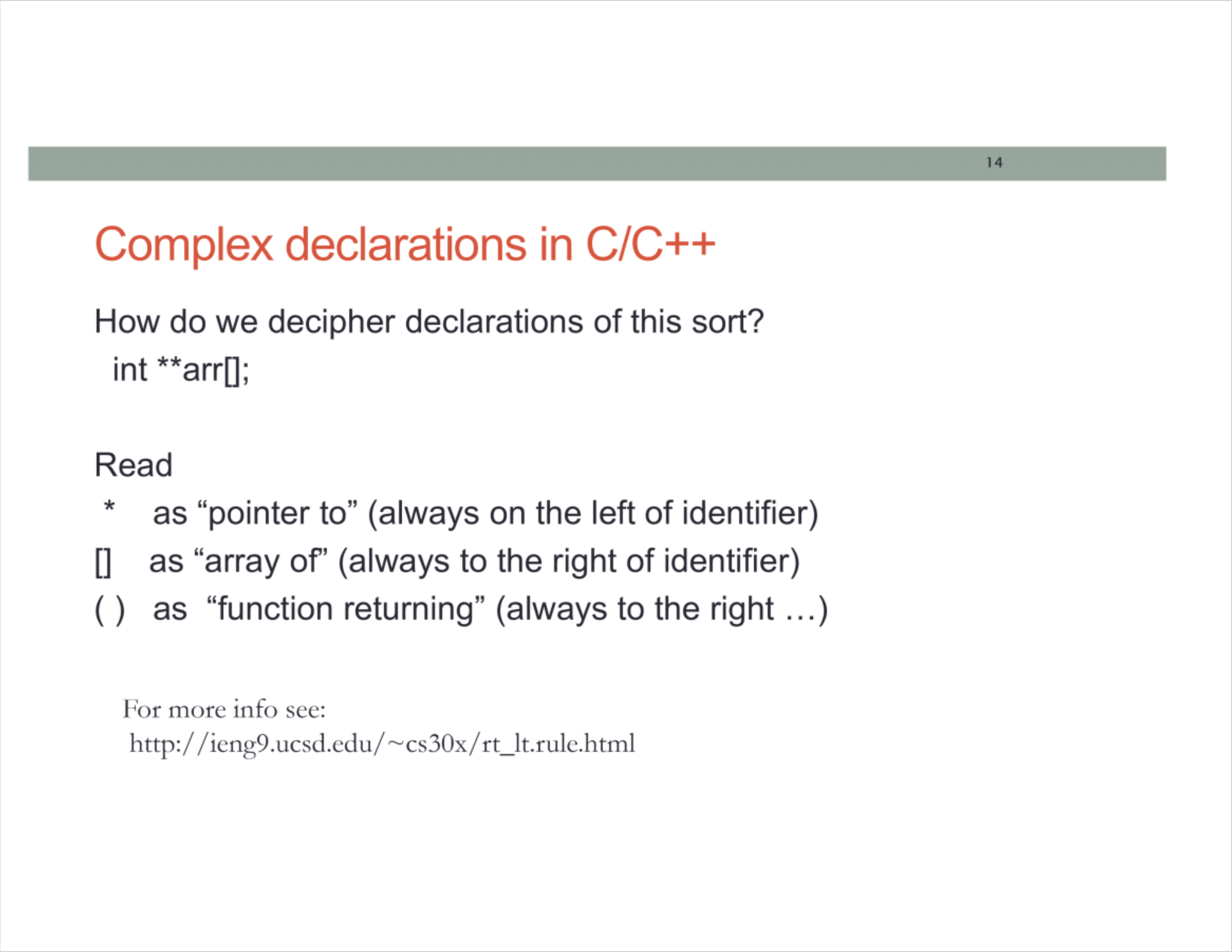
2) After declaring a pointer: `int *ptr;`

   `ptr` doesn't actually point to anything yet. We can either:

   - make it point to something that already exists, or
   - allocate room in memory for something new that it will point to
   - Null check before dereferencing

Page 14

Monday, May 8, 2017 12:47 PM



Complex declarations in C/C++

How do we decipher declarations of this sort?

```
int **arr[];
```

Read

* as “pointer to” (always on the left of identifier)

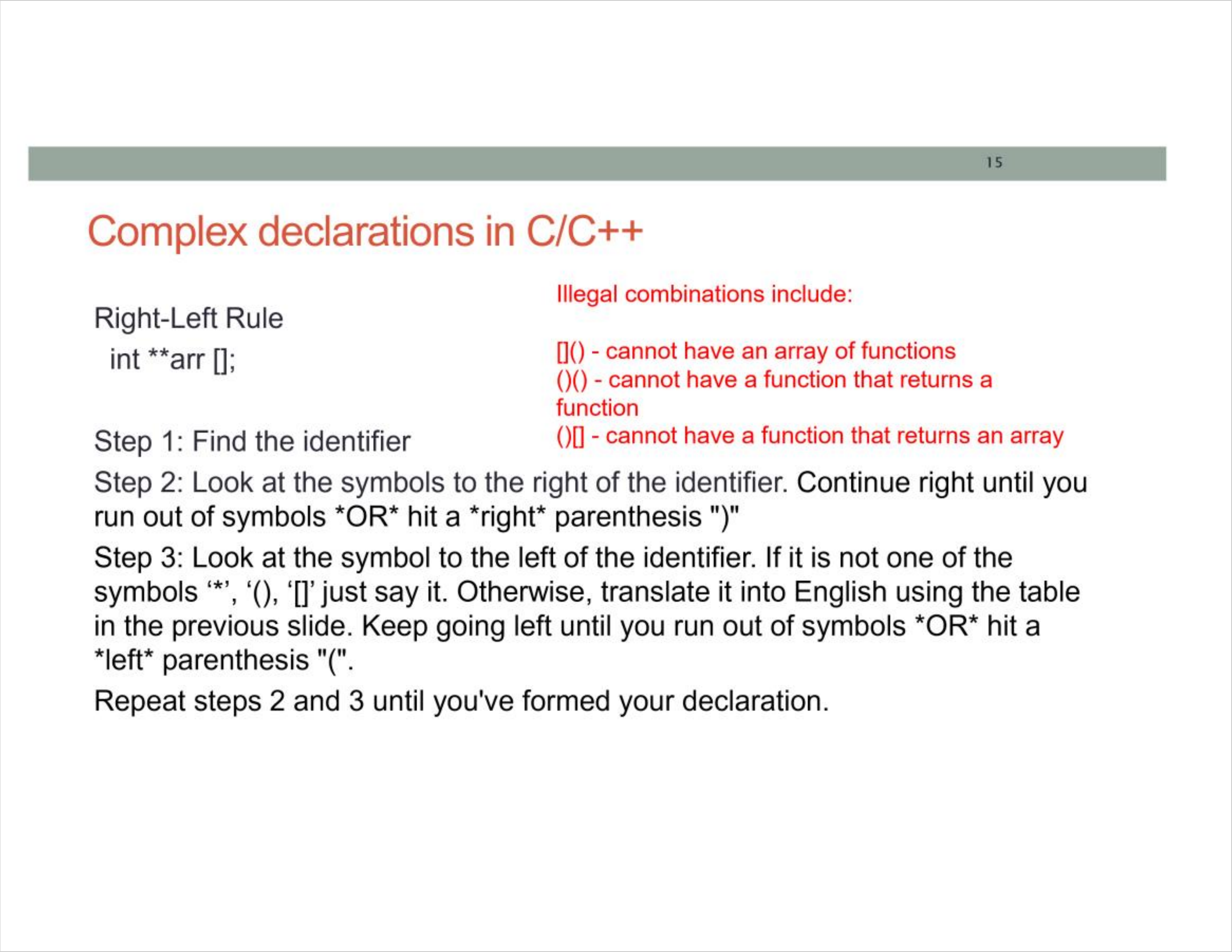
[] as “array of” (always to the right of identifier)

( ) as “function returning” (always to the right …)

For more info see:
http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html

Page 15

Monday, May 8, 2017 12:47 PM



Complex declarations in C/C++

Right-Left Rule

```
int **arr [];
```

Illegal combinations include:

[]() - cannot have an array of functions
()() - cannot have a function that returns a function
()[] - cannot have a function that returns an array

Step 1: Find the identifier

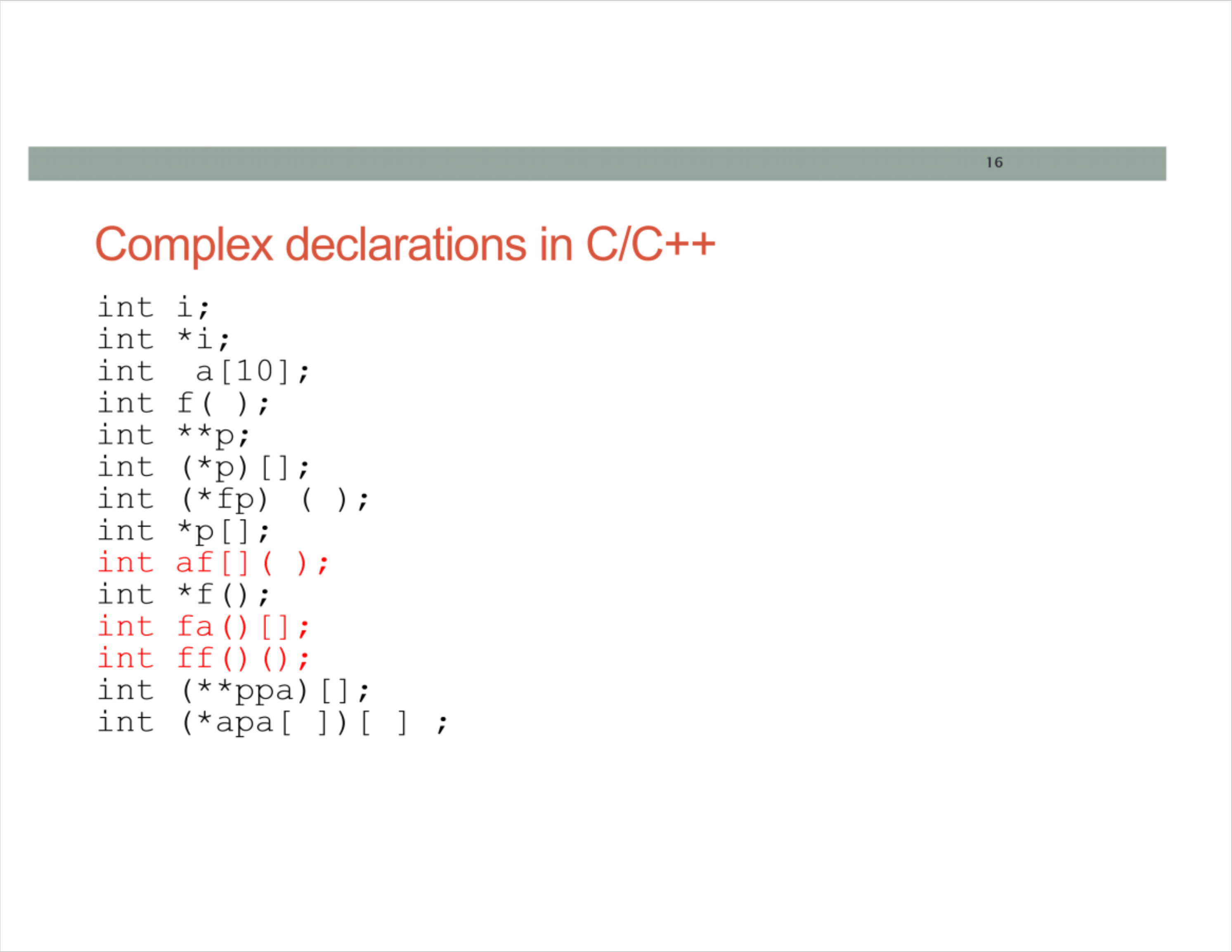
Step 2: Look at the symbols to the right of the identifier. Continue right until you run out of symbols *OR* hit a *right* parenthesis ")"

Step 3: Look at the symbol to the left of the identifier. If it is not one of the symbols '*', '(), '[]' just say it. Otherwise, translate it into English using the table in the previous slide. Keep going left until you run out of symbols *OR* hit a *left* parenthesis "(".

Repeat steps 2 and 3 until you've formed your declaration.

Page 16

Monday, May 8, 2017 12:47 PM

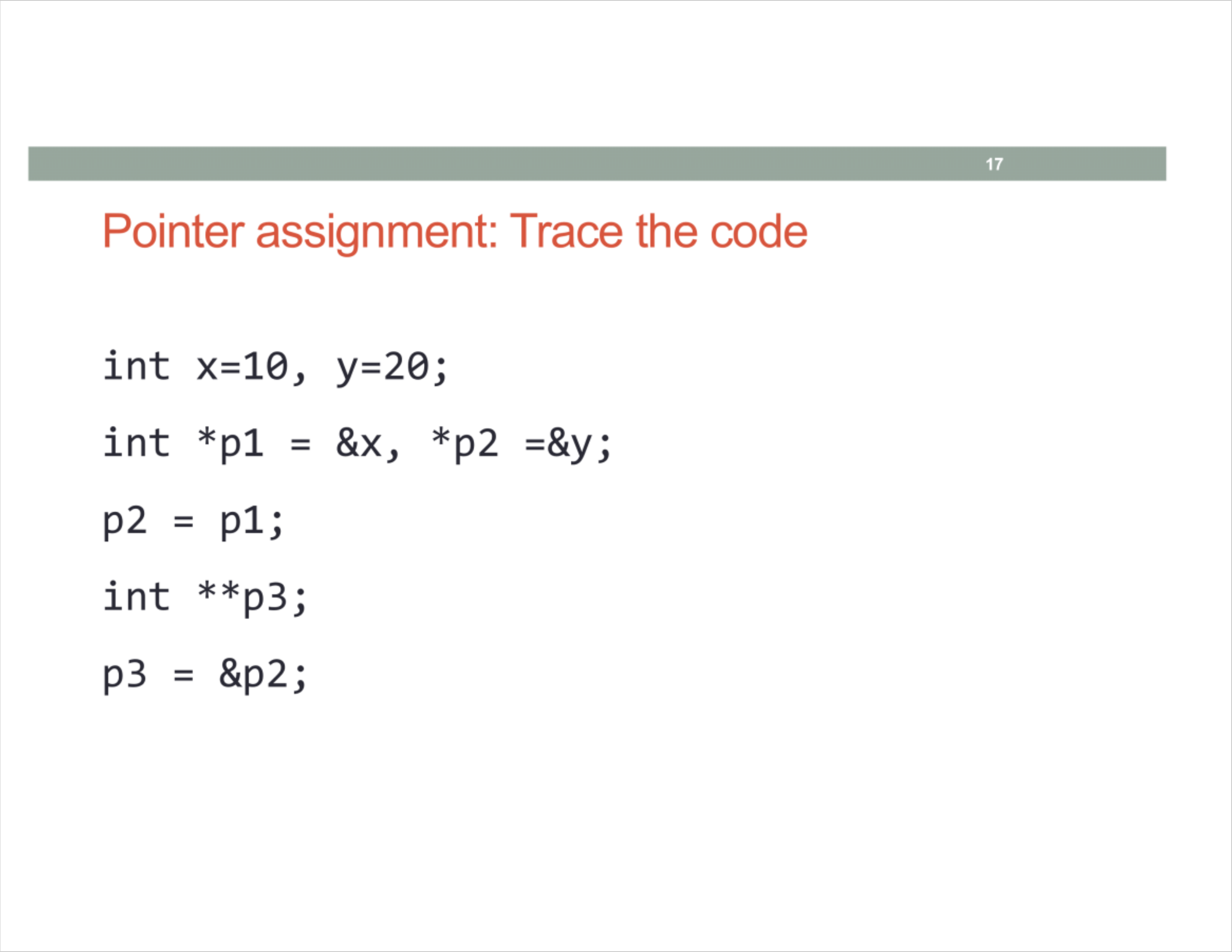


Complex declarations in C/C++

```
int i;
int *i;
int  a[10];
int f( );
int **p;
int (*p)[];
int (*fp)  ( );
int *p[];
int af[]( );
int *f();
int fa()[];
int ff()();
int (**ppa)[];
int (*apa[ ])[ ] ;
```

Page 17

Monday, May 8, 2017 12:47 PM

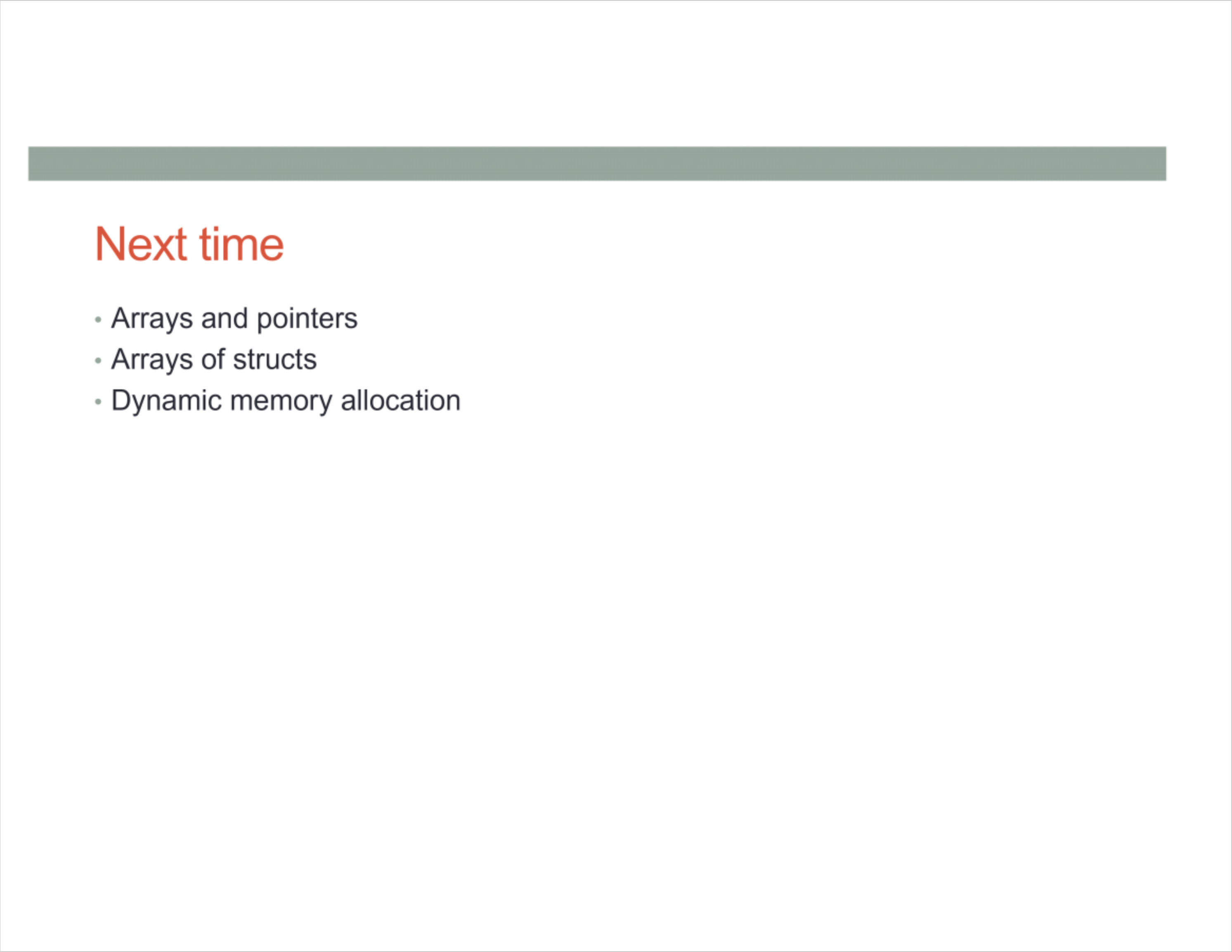


Pointer assignment: Trace the code

```
int x=10, y=20;
int *p1 = &x, *p2 =&y;
p2 = p1;
int **p3;
p3 = &p2;
```

Page 18

Monday, May 8, 2017 12:47 PM



Next time

- Arrays and pointers
- Arrays of structs
- Dynamic memory allocation