Lecture 16: Recursion on strings (contd)

Wednesday, May 31, 2017 12:39 PM

MORE STRINGS AND RECURSION

Problem Solving with Computers-I

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

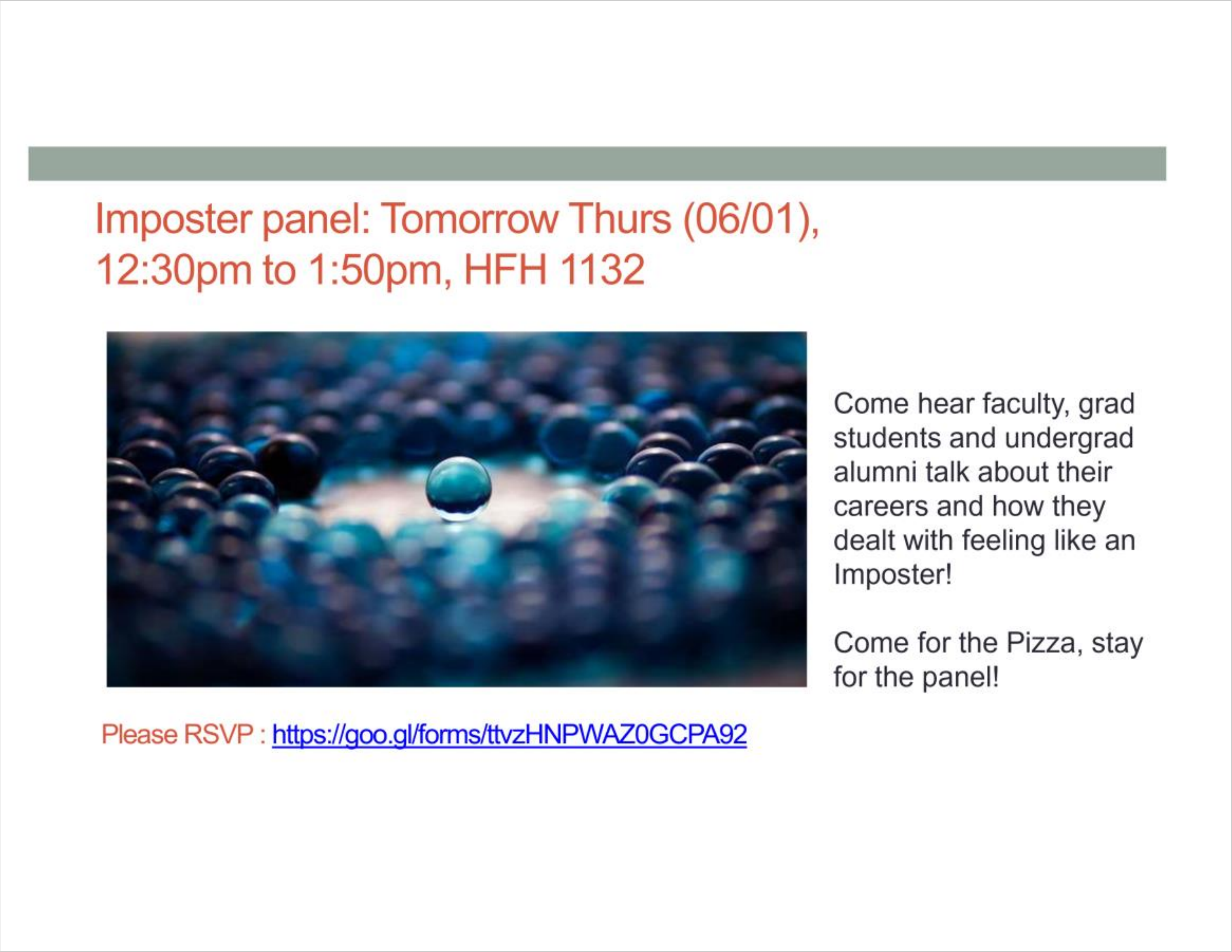
```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook :)";
    return 0;

}
```

Page 2

Wednesday, May 31, 2017 12:39 PM

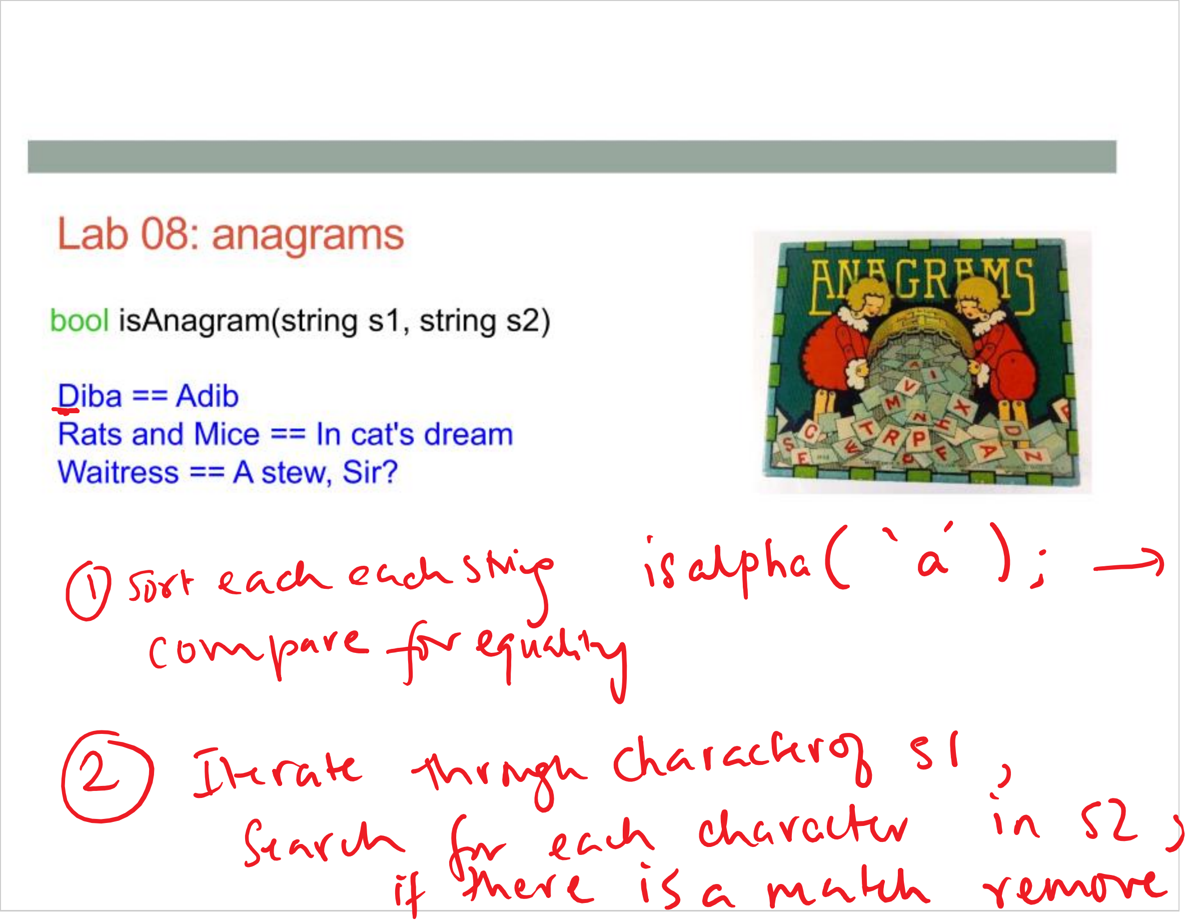


Imposter panel: Tomorrow Thurs (06/01), 12:30pm to 1:50pm, HFH 1132

Come hear faculty, grad students and undergrad alumni talk about their careers and how they dealt with feeling like an Imposter!

Come for the Pizza, stay for the panel!

Please RSVP : https://goo.gl/forms/ttvzHNPWAZ0GCPA92



Lab 08: anagrams

bool isAnagram(string s1, string s2)

Diba == Adib
Rats and Mice == In cat's dream
Waitress == A stew, Sir?

① Sort each each string
compare for equality

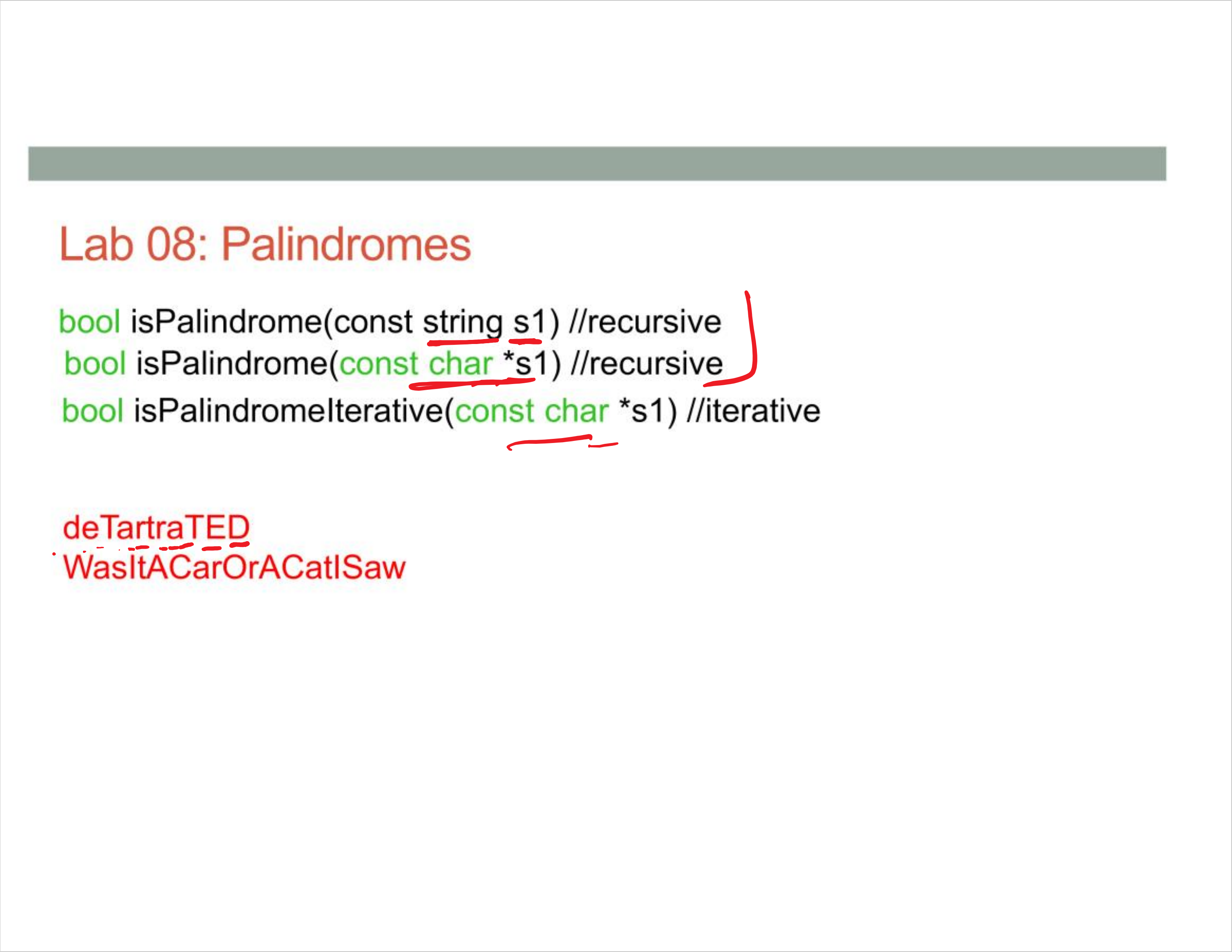
isalpha('a'); → true

② Iterate through character of s1,
Search for each character in s2,
if there is a match remove character from s2

s2.find( )
s2.erase()

Page 4

Wednesday, May 31, 2017 12:39 PM



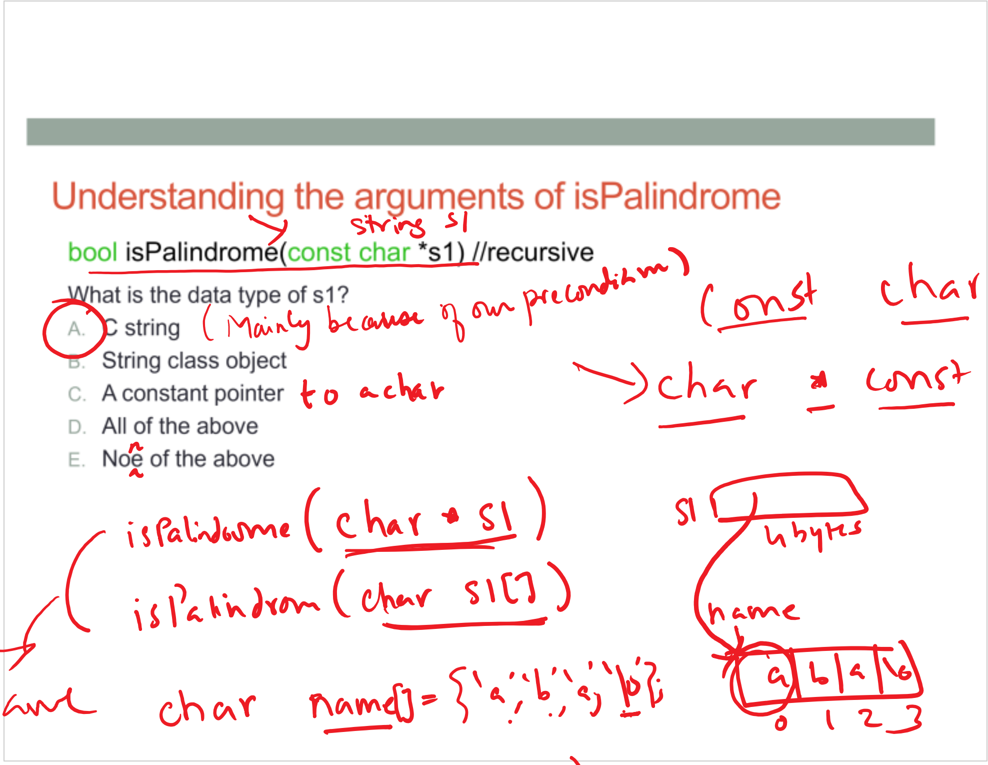
Lab 08: Palindromes

```
bool isPalindrome(const string s1) //recursive
 bool isPalindrome(const char *s1) //recursive
bool isPalindromeIterative(const char *s1) //iterative
```

deTartraTED
WasItACarOrACatISaw

Page 5

Wednesday, May 31, 2017 12:39 PM



Understanding the arguments of isPalindrome

string s1

bool isPalindrome(const char *s1) //recursive

What is the data type of s1?

[✓ A.] C string (Mainly because of our precondition)

B. String class object

C. A constant pointer to a char

D. All of the above

E. Noe of the above

const char * s1; // s1 is a pointer to a const char

char * const s1; // s1 is a const pointer to a char

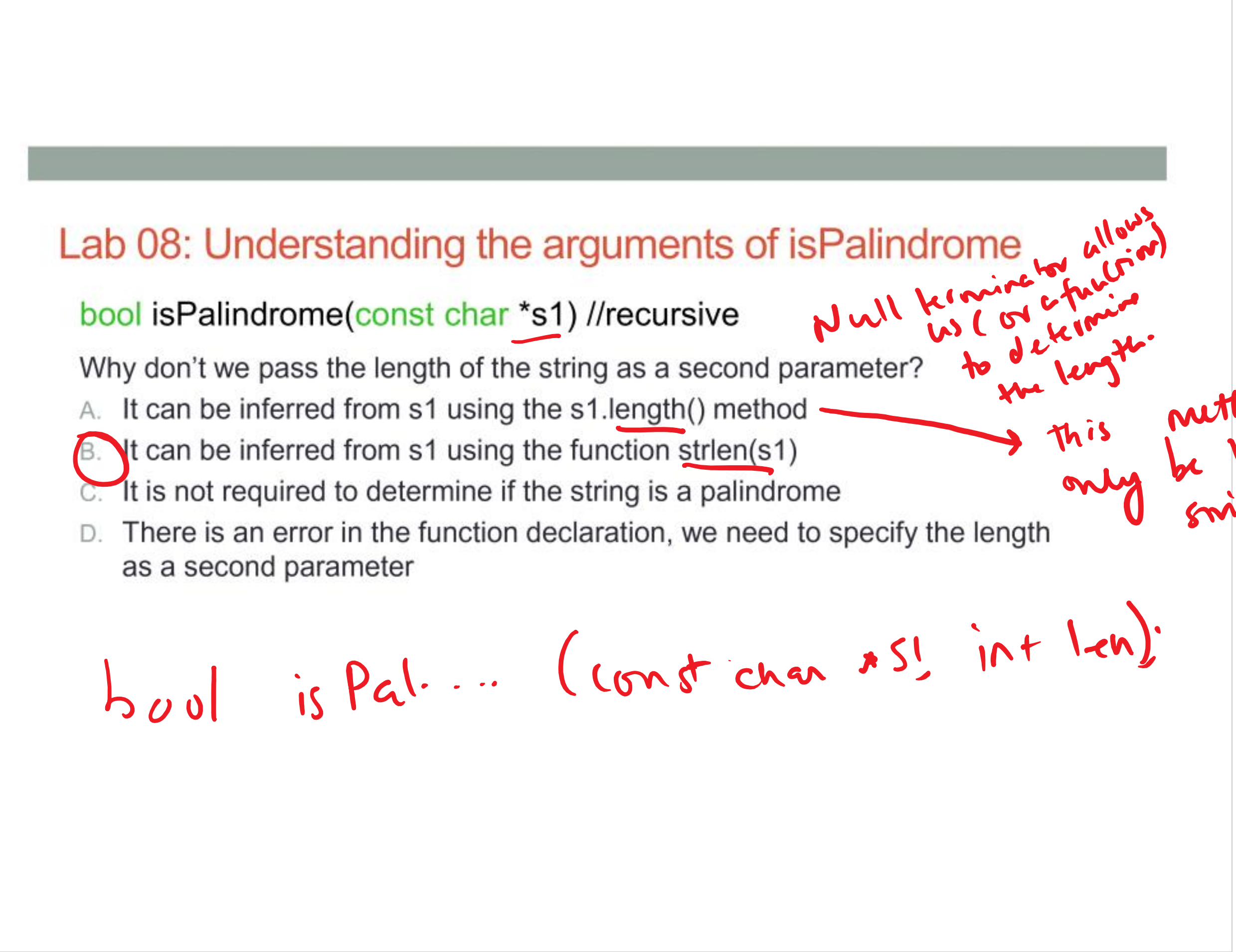
isPalindome(char * s1)

isPalindrom(char s1[])

same

char name[] = {'a','b','a','\0'};

isPalindrome(name);



Lab 08: Understanding the arguments of isPalindrome

```
bool isPalindrome(const char *s1) //recursive
```

Null terminator allows us (or a function) to determine the length.

Why don't we pass the length of the string as a second parameter?

A. It can be inferred from s1 using the s1.length() method

this method can only be used on string class objects

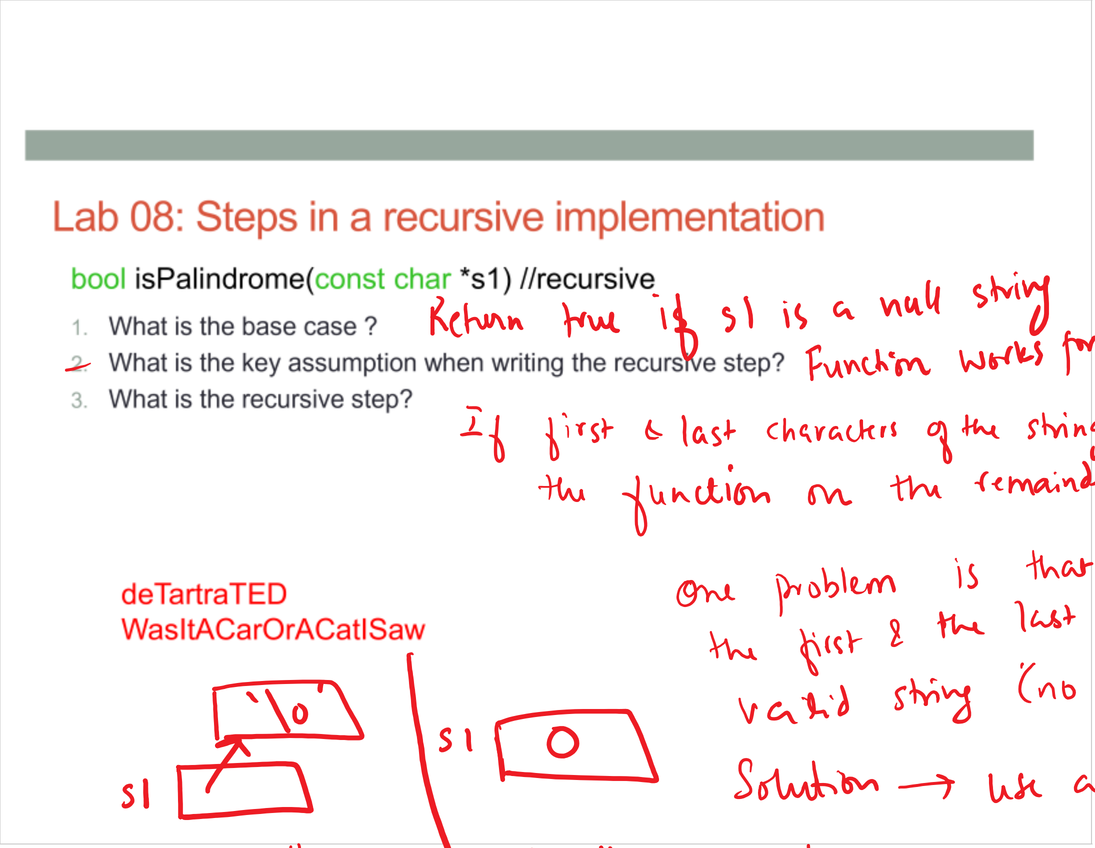
[✓ B.] It can be inferred from s1 using the function strlen(s1)

C. It is not required to determine if the string is a palindrome

D. There is an error in the function declaration, we need to specify the length as a second parameter

bool isPal... (const char *s1, int len);

Wednesday, May 31, 2017 12:39 PM



Lab 08: Steps in a recursive implementation

```
bool isPalindrome(const char *s1) //recursive
```

1. What is the base case ? Return true if s1 is a null string
2. What is the key assumption when writing the recursive step? Function works for a string of length n-1
3. What is the recursive step?

If first & last characters of the string are equal, call the function on the remainder of the string

deTartraTED
WasItACarOrACatISaw

s1 is a null string or s1 is pointing to null

s1 is null

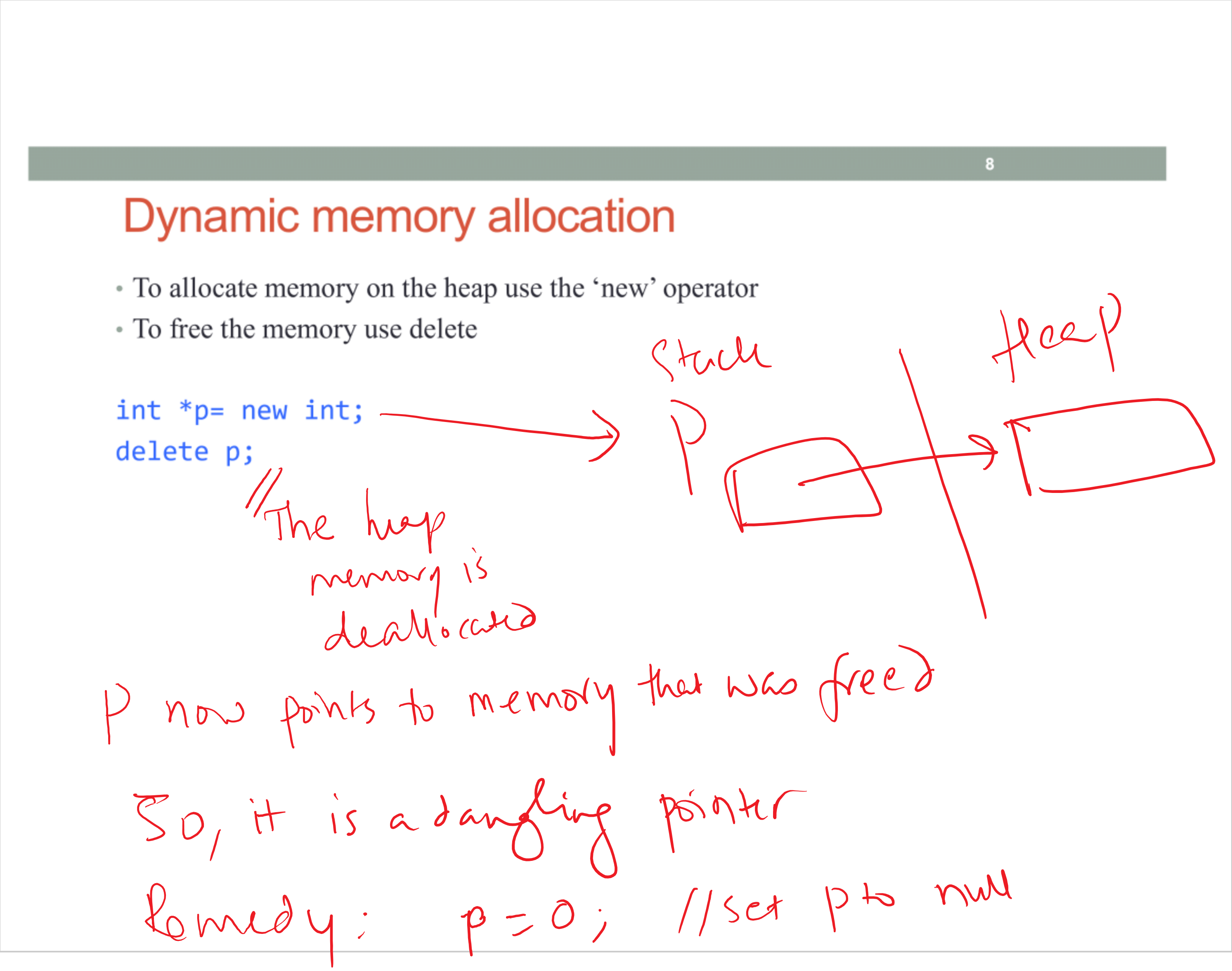
One problem is that the characters between the first & the last character are not a valid string (no null terminator)

Solution → use a helper function that takes a char array and the length of the array as parameters

```
bool isPalindromeHelper(const char *arr, int len) {
        if (len <= 1)
            return true;
        if ( arr[0] == arr[len-1] )
            return isPalindromeHelper(arr+1, len-2);
        else
            return false;
}
```

Page 8

Wednesday, May 31, 2017 12:39 PM



8

Dynamic memory allocation

- To allocate memory on the heap use the 'new' operator
- To free the memory use delete

```
int *p= new int;
delete p;
```

//The heap memory is deallocated

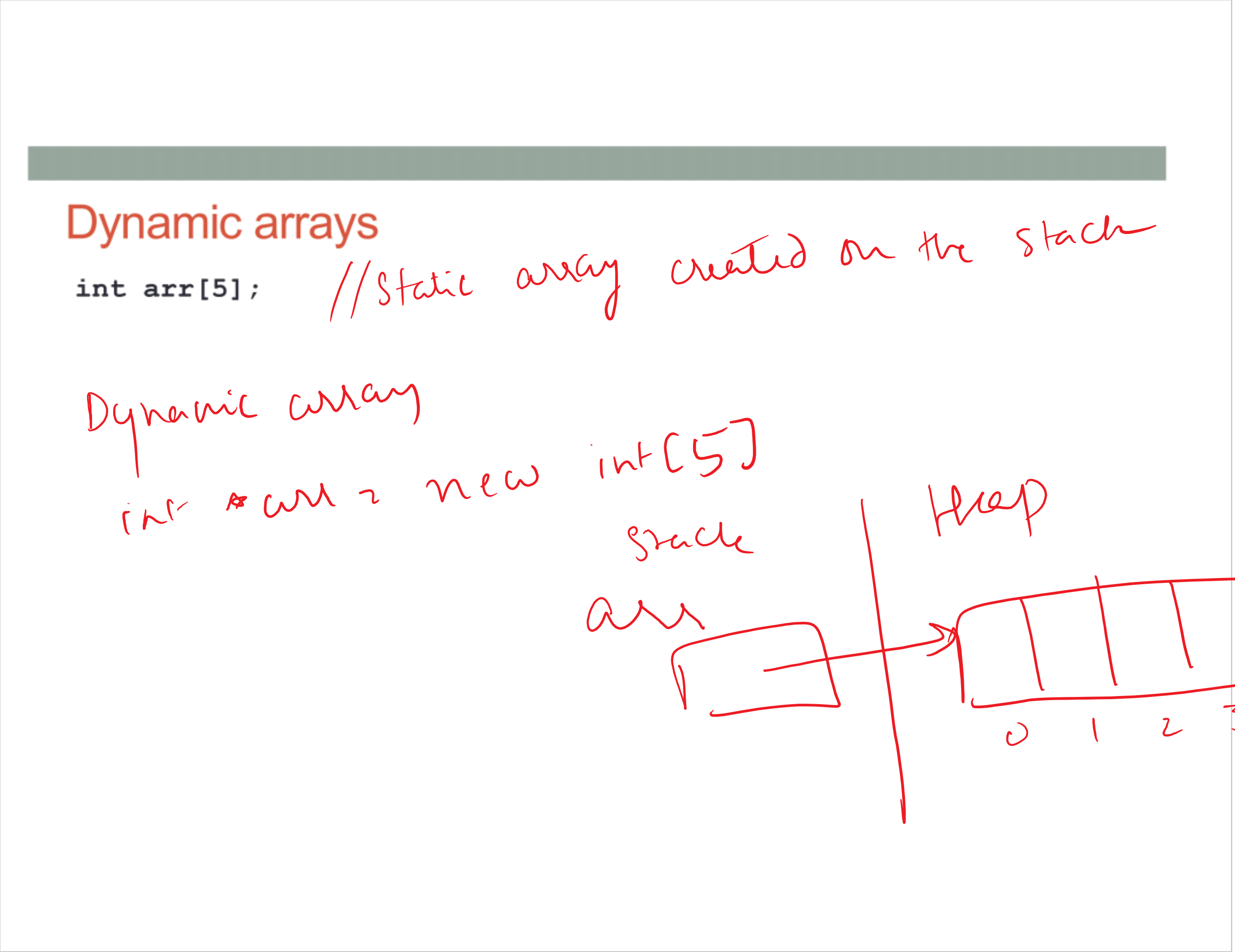
p now points to memory that was freed

So, it is a dangling pointer

Remedy: p = 0; //set p to null

Page 9

Wednesday, May 31, 2017 12:39 PM



Dynamic arrays

```
int arr[5];
```

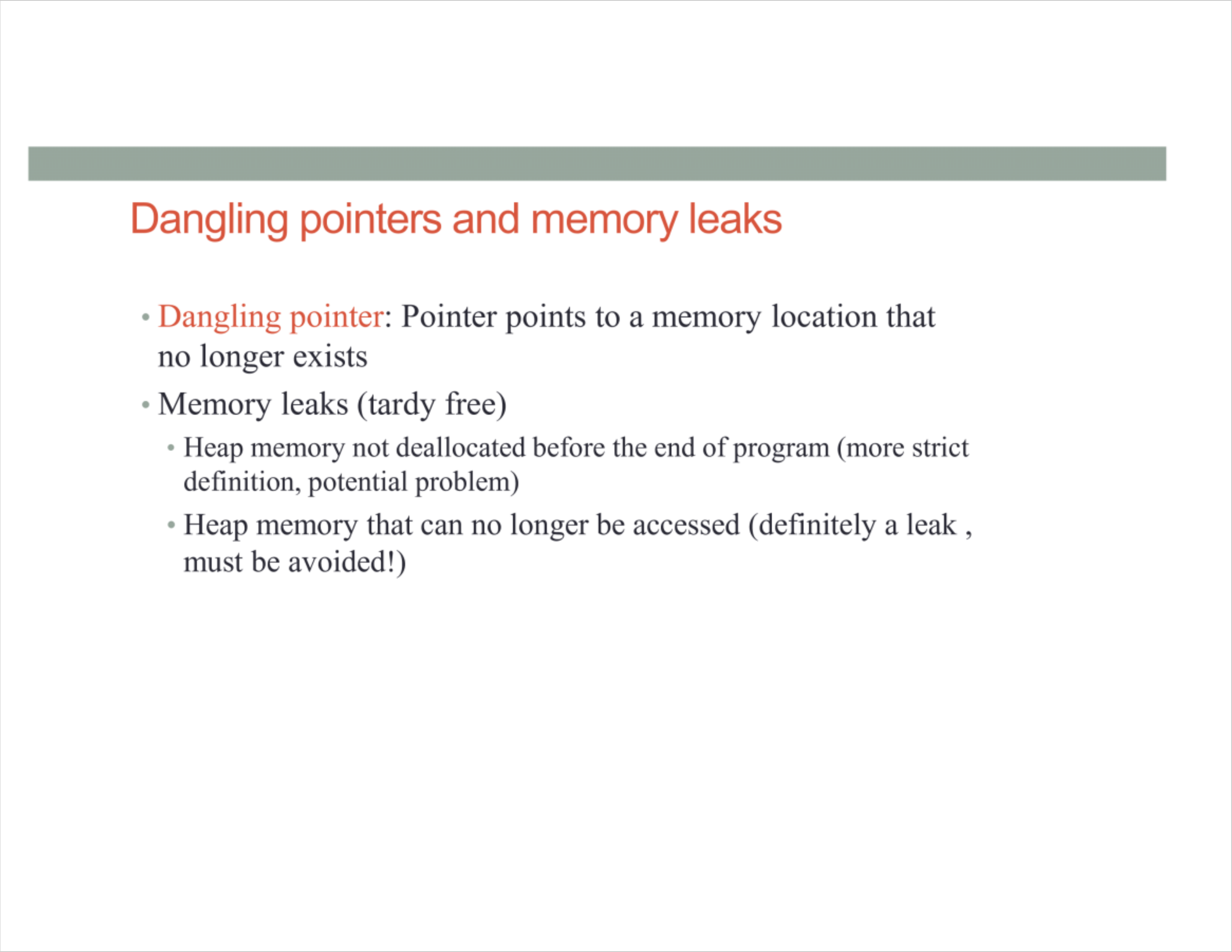
//Static array created on the stack

Dynamic array

int *arr = new int[5]

Page 10

Wednesday, May 31, 2017 12:39 PM

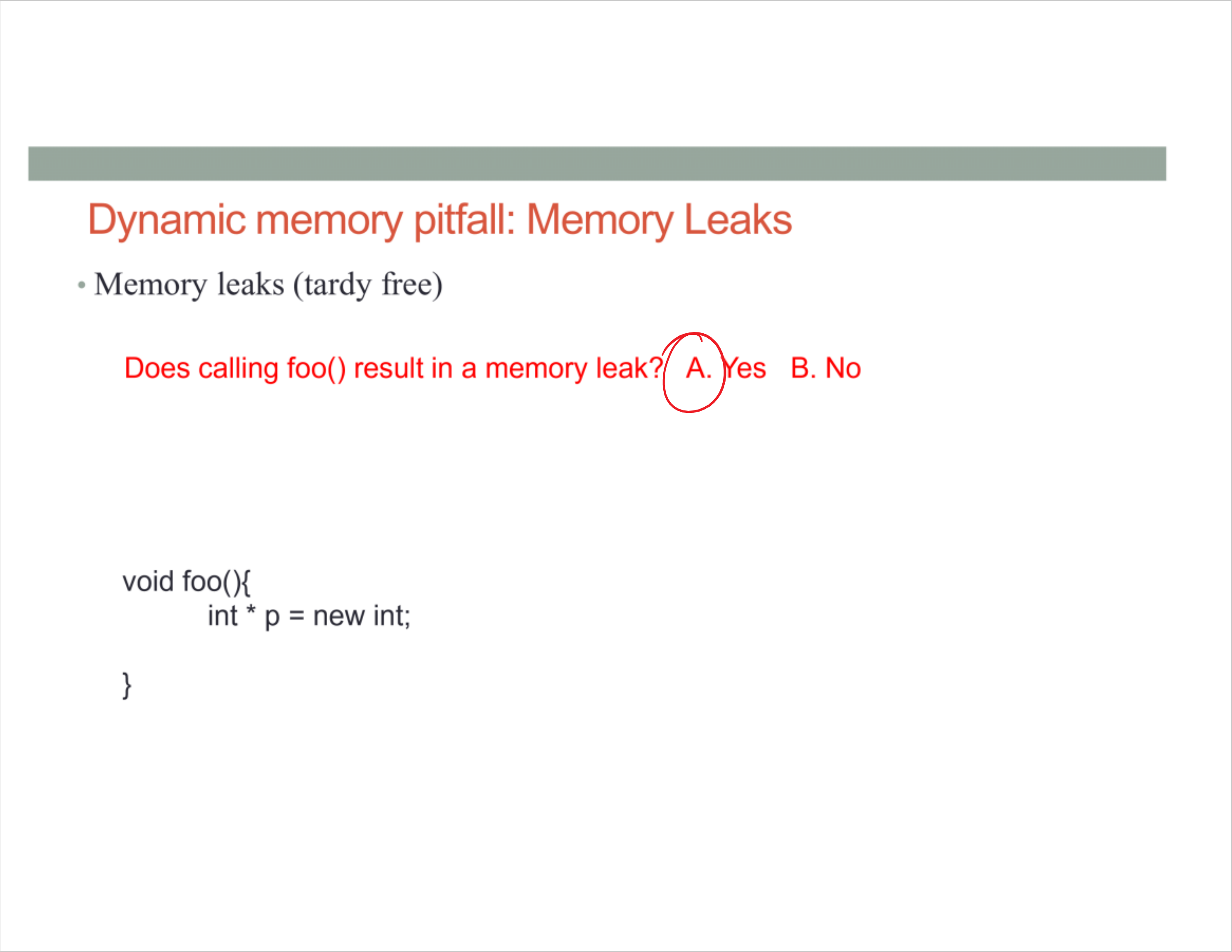


Dangling pointers and memory leaks

- Dangling pointer: Pointer points to a memory location that no longer exists
- Memory leaks (tardy free)
  - Heap memory not deallocated before the end of program (more strict definition, potential problem)
  - Heap memory that can no longer be accessed (definitely a leak , must be avoided!)

Page 11

Wednesday, May 31, 2017 12:39 PM



Dynamic memory pitfall: Memory Leaks

- Memory leaks (tardy free)

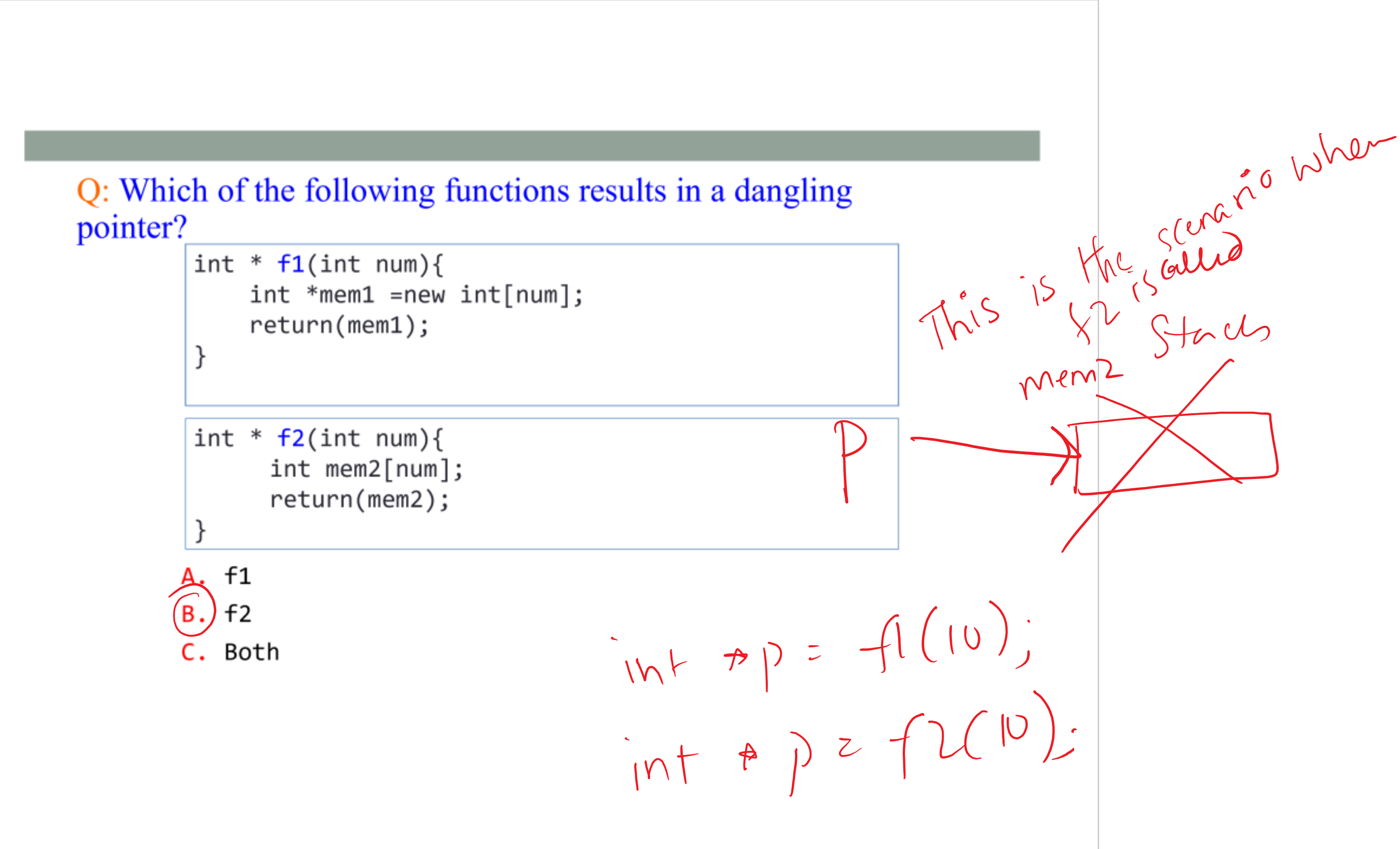
Does calling foo() result in a memory leak? [✓ A. Yes] B. No

```
void foo(){
        int * p = new int;

}
```

Page 12

Wednesday, May 31, 2017 12:39 PM



Q: Which of the following functions results in a dangling pointer?

```
int * f1(int num){
    int *mem1 =new int[num];
    return(mem1);
}
```

```
int * f2(int num){
      int mem2[num];
      return(mem2);
}
```

A. f1

[✓ B.] f2

C. Both

This is the scenario when f2 is called

int *p = f1(10);

int *p = f2(10);