

MORE STRINGS AND RECURSION



Problem Solving with Computers-I

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



C++

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

int main()
{
    cout << "Hola Facebook!";
    return 0;
}
```



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>

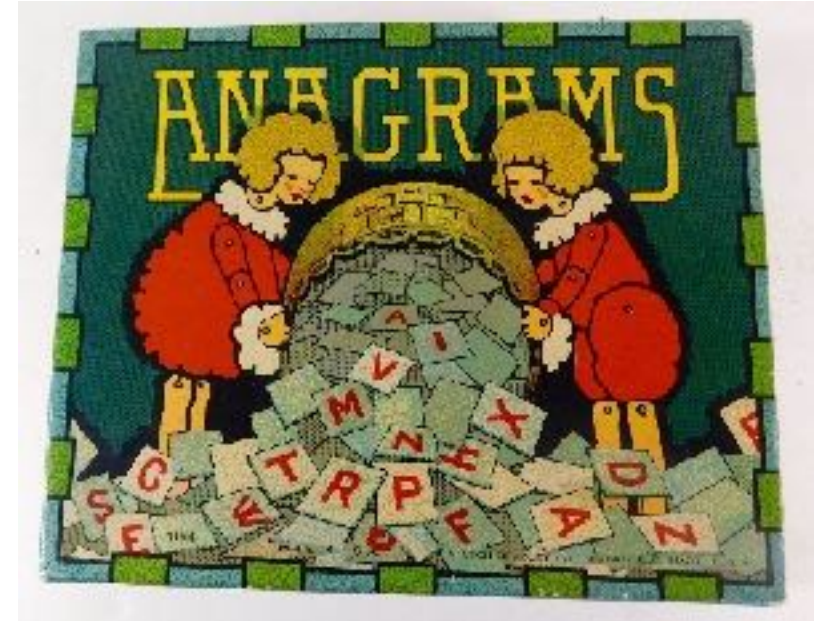
Lab 08: anagrams

```
bool isAnagram(string s1, string s2)
```

Diba == Adib

Rats and Mice == In cat's dream

Waitress == A stew, Sir?



Lab 08: Palindromes

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

deTartraTED

WasItACarOrACatISaw

Understanding the arguments of isPalindrome

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

What is the data type of s1?

- A. C string
- B. String class object
- C. A constant pointer
- D. All of the above
- E. None of the above

Lab 08: Understanding the arguments of isPalindrome

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

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
- 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

Lab 08: Steps in a recursive implementation

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

1. What is the base case ?
2. What is the key assumption when writing the recursive step?
3. What is the recursive step?

deTartraTED
WasItACarOrACatISaw

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;
```


Dynamic arrays

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

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!)

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;  
  
}
```

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

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

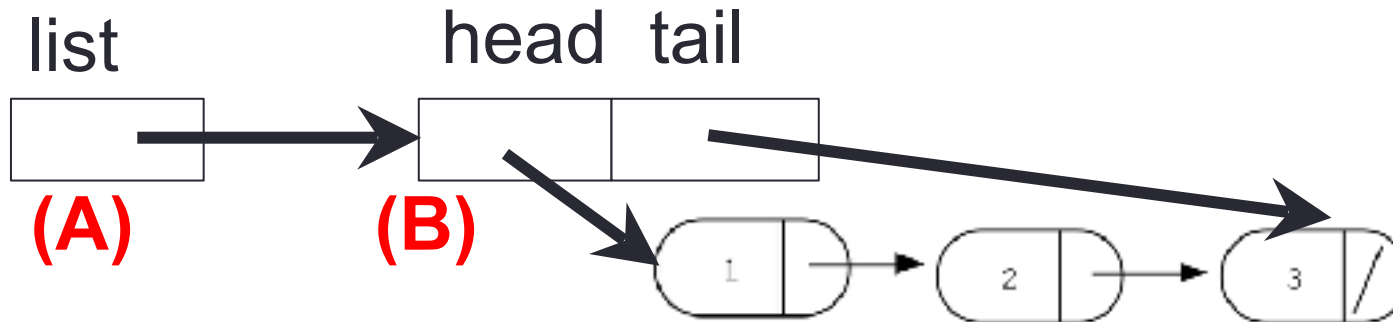
```
int * f2(int num){  
    int mem2[num];  
    return(mem2);  
}
```

- A. f1
- B. f2
- C. Both

Deleting the list

```
int freeLinkedList(LinkedList * list){...}
```

Which data objects are deleted by the statement: **delete list;**



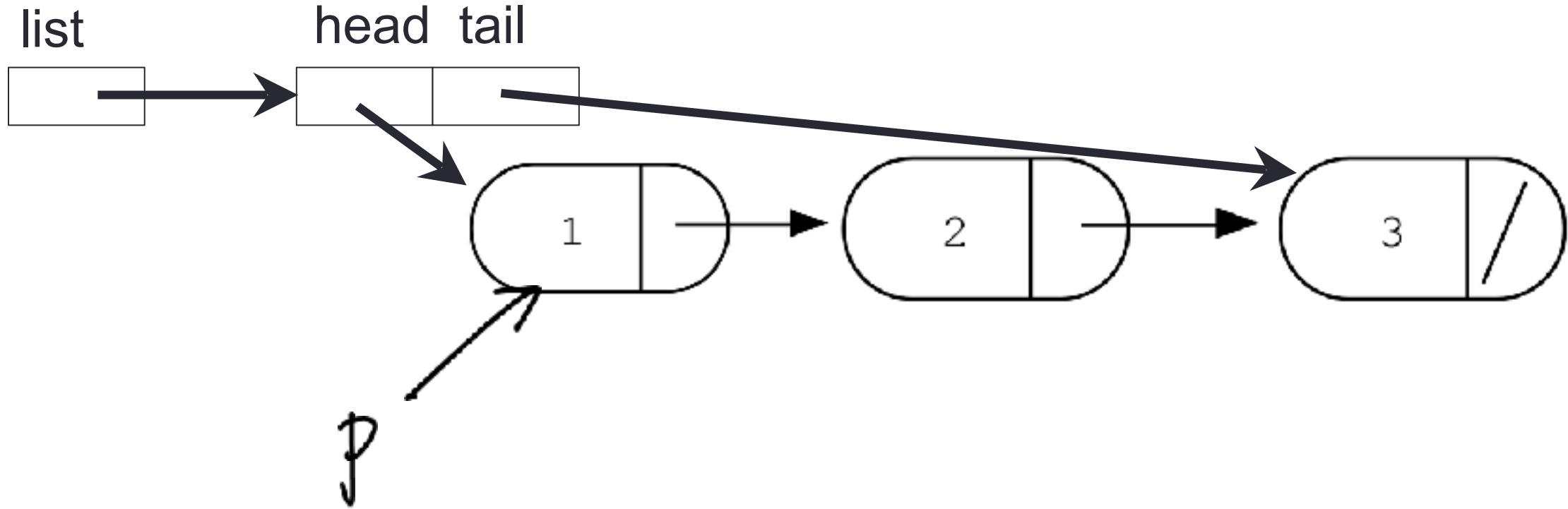
(C) All nodes of the linked list

(D) B and C

(E) All of the above

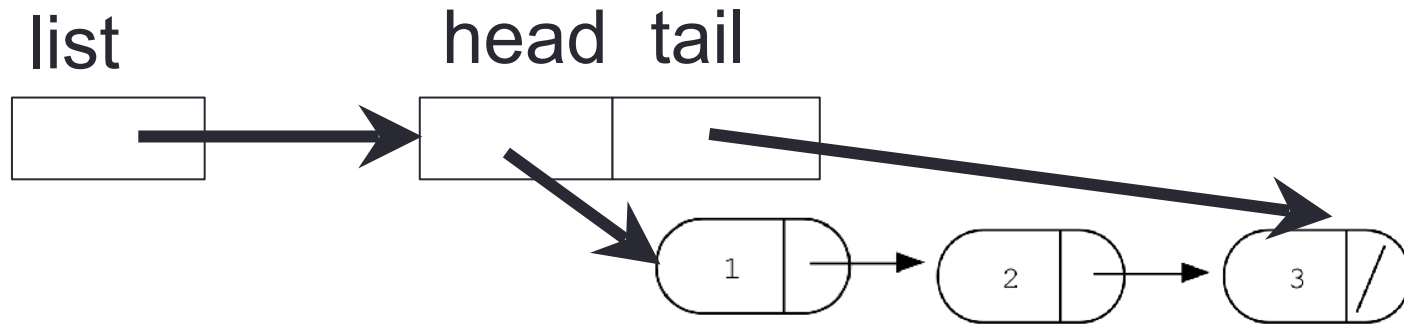
Does this result in a memory leak?

Delete node 2 in the list



Delete the list

```
int freeLinkedList(LinkedList * list);
```



Next time

- Advanced problems in recursion on linked-lists