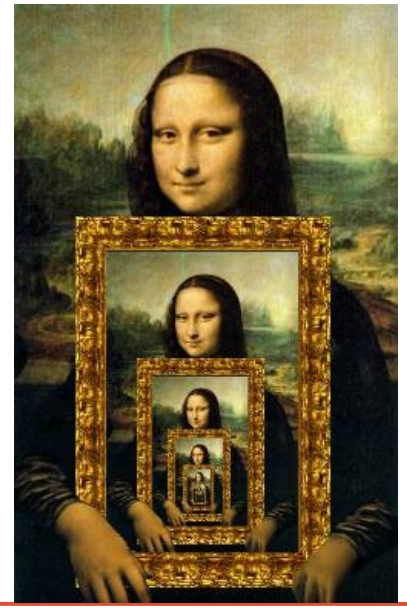
MORE STRINGS AND RECURSION



Problem Solving with Computers-I

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. Noe 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

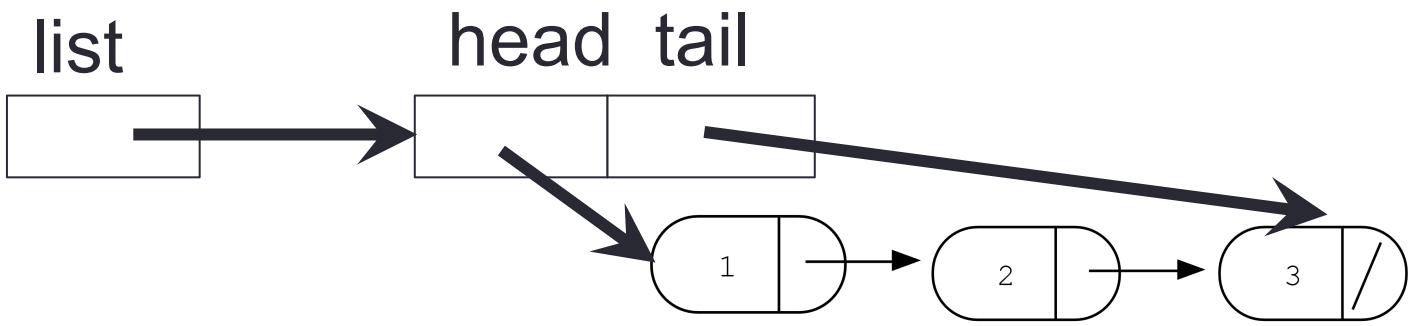
Dynamic memory allocation

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

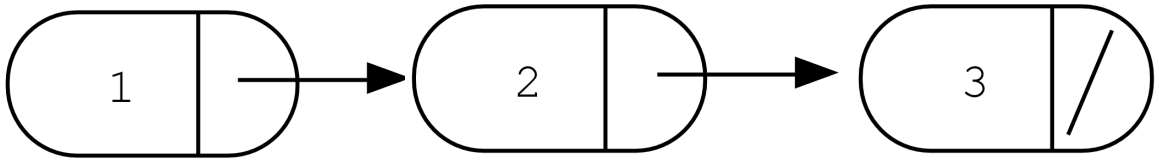
```
int *p= new int;
delete p;
```

Delete the list

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



Recursion on lists: compute the sum of all elements



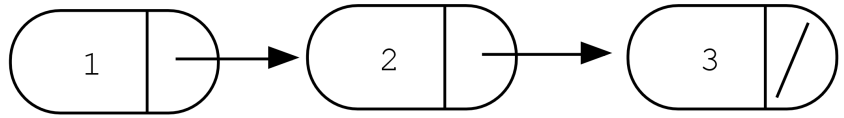

```
int sum(Node *head){

    return head->data+sum(head);
}
```

Which of the following is true about the given implementation?

A. It is correct
B. It will not return the correct sum
C. It will result in a segfault
D. It never ends

Recursion on lists: delete a value recursively

void deleteNodeRecursive(LinkedList *list, int value)



Node* deleteNodeRecursiveHelper(Node *head, int value)

Recall the steps towards a recursive solution