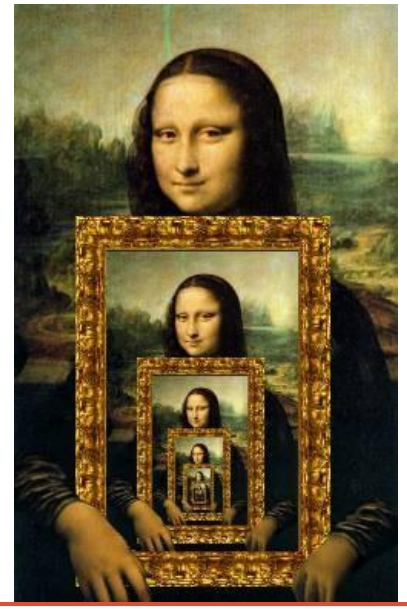
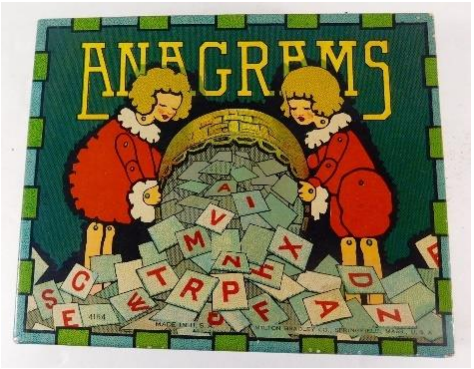
MORE STRINGS AND RECURSION



Problem Solving with Computers-I

https://ucsb-cs16-wi17.github.io/

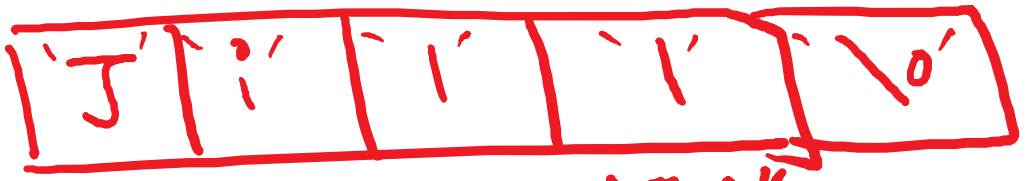


Take out your Homework 13

Q1: How are ordinary arrays of characters and C-strings similar and how are they dissimilar?

Discuss with your neighbor (3 minutes)

C strings are char arrays that have a null terminator ('\0') character at the end e.g.



The ascii value of '\0' is 0.

String literals are const C strings.
(i.e. strings that cannot be modified)

char name[] = "Jill";
↑ C string

char *name = "Jill";
↑ String literal

Which of the following is not a C string? (related to Q1)

A. `char mystr[5] = "John";`

B. `char mystr[] = "Mary";`

C. `const char *mystr = "Jill";`

D. `char mystr[4]= {'J', 'i', 'l', 'l'};` No null terminator

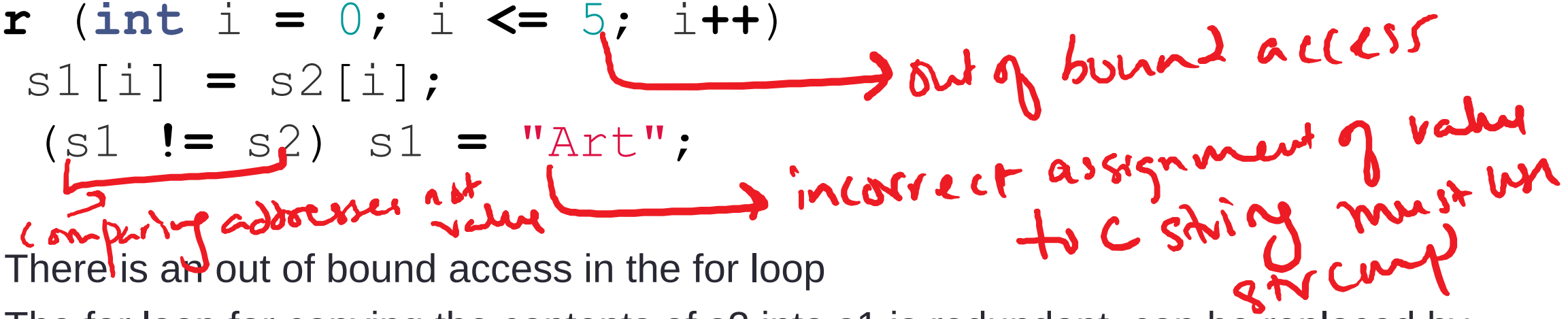
E. All of the above are C strings

We can access the elements of a C string just like we access elements of a char array.

```
mystr[i] = 'a';
```

Q2: Which of the following statements is FALSE about the given code?

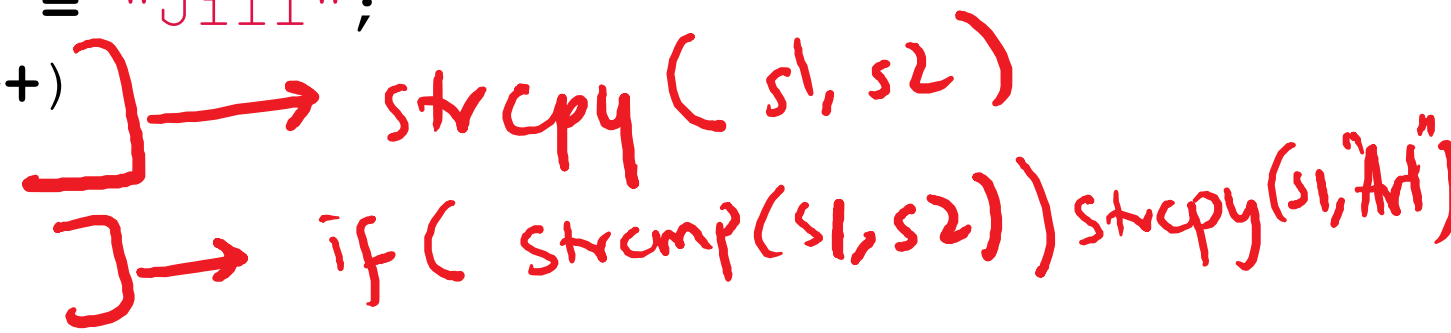
```
char s1[5] = "Mark", s2[5] = "Jill";
for (int i = 0; i <= 5; i++)
    s1[i] = s2[i];
if (s1 != s2) s1 = "Art";
```



A. There is an out of bound access in the for loop
B. The for loop for copying the contents of s2 into s1 is redundant, can be replaced by s1 = s2;
C. The logic for comparing the inequality of two strings in the if statement is incorrect.
D. The body of the if statement is incorrect: cannot change the base address of an array

C String Standard Functions #include <cstring>

```
char s1[5] = "Mark", s2[5] = "Jill";
for (int i = 0; i <= 5; i++)
    s1[i] = s2[i];
if (s1 != s2) s1 = "Art";
```



- **`int strlen(char *string);`**
  - Returns the length not counting of `string` the null terminator
- **`int strcmp(char *str1, char *str2);`**
  - return 0 if `str1` and `str2` are identical (how is this different from `str1 == str2`?)
- **`int strcpy(char *dst, char *src);`**
  - copy the contents of string `src` to the memory at `dst`. The caller must ensure that `dst` has enough memory to hold the data to be copied.
- **`char* strcat(char *s1, char *s2);`**
  - concatenate the contents of string `s2` to `s2`and returns pointer to resulting string

Q3: What is the output of the following code? (solo vote)

```
char s1[4] = "abc", s2[4] = "ABC";

if (strcmp(s1, s2)) cout << "Hi!";

else cout << "Hey!";
```

A. Hi!
B. Hey!
C. Compiler error
D. Runtime error

"abc" & "ABC" are not equal, strcmp has negation logic

C strings vs. String class: What is the output of the code?

```
string s1 = "Mark";
string s2 = "Jill";
for (int i = 0; i <= s1.length(); i++)
     s2[i] = s1[i];
if (s1 == s2) s1 = "Art";
cout<<s1<<“ ”<<s2<<endl;
```

A. Mark Jill
B. Mark Mark
C. Art Mark
D. Compiler error
E. Run-time error

The C++ string class methods

```
string fruit = "Apple";
int len = fruit.length();
int pos= fruit.find('l');
string part= fruit.substr(1,3);
fruit.erase(2,3);
fruit.insert(2,"ricot");
fruit.replace(2,5,"ple");
```

// len = 5
// pos = 3
part = "ppl"
// fruit = "Ap"
// fruit = "Apricot"
fruit = "Apple"

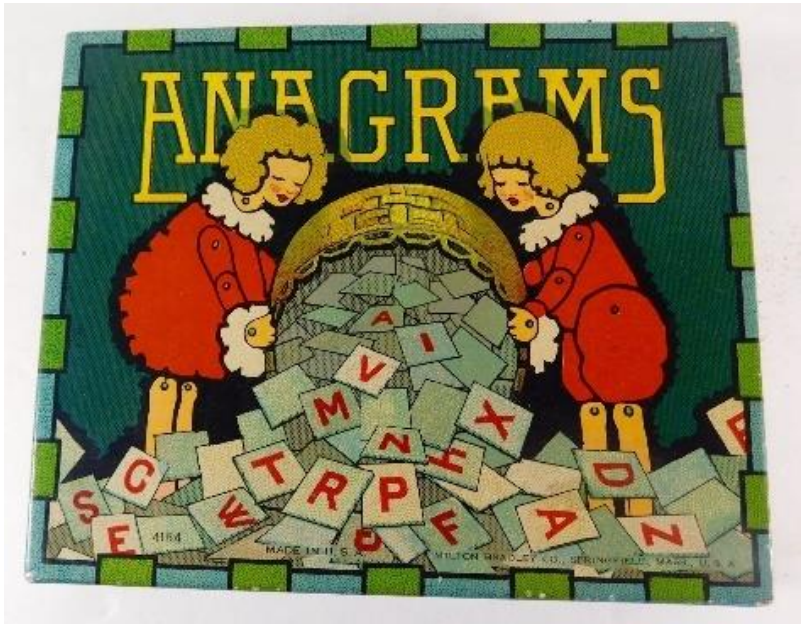
Check out cctype for checks and conversions on characters

```
fruit[0]= tolower(fruit[0]);
isalpha(fruit[0])
```

A → a (only works on individual chars)

→ returns true if fruit[0] is an alphabet 'a'-'z' or 'A'-'Z'

Lab 08: anagrams and palindromes



bool isAnagram(string s1, string s2)

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

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

deTartraTED
WasItACarOrACatISaw

Why don't we pass the length of the string?

Lab 08: anagrams and palindromes

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

deTartraTED

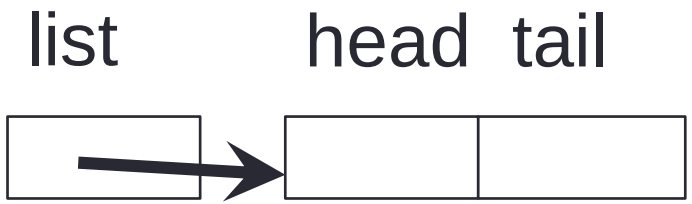
WasItACarOrACatISaw

because it can be deduced!

Why don't we pass the length of the string?

void deleteNodeRecursive(LinkedList *list, int value)

Case 1: EMPTY LIST

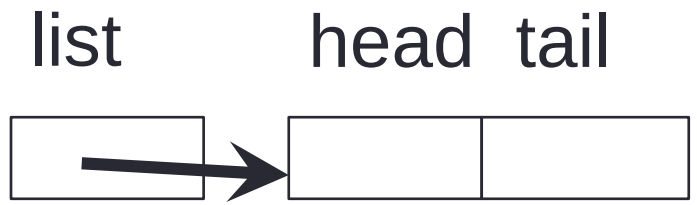



```
int empty[0]={};
LinkedList *list = arrayToLinkedList(empty,4);
ASSERT_EQUALS( "null", linkedListToString(list));
deleteNodeRecursive(list, 61);
assertEquals( "null",linkedListToString(list);
```

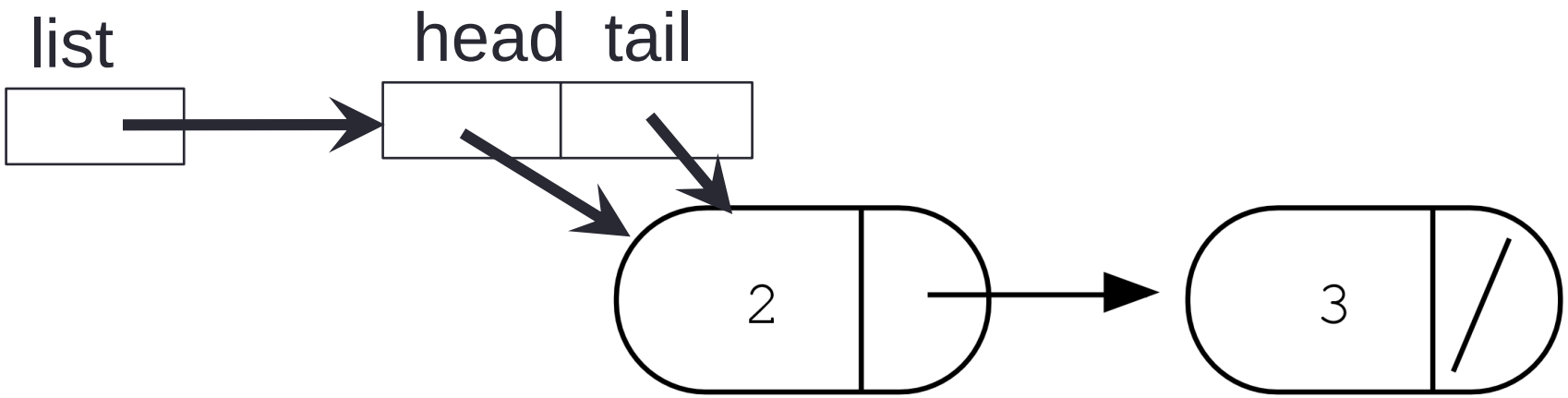
Form a group of four people

1. Each pair: Come up with the next test case (list of size 1) and the code for that case
2. Exchange and review your code
3. Come up with the next logical test case

Case 2: One node

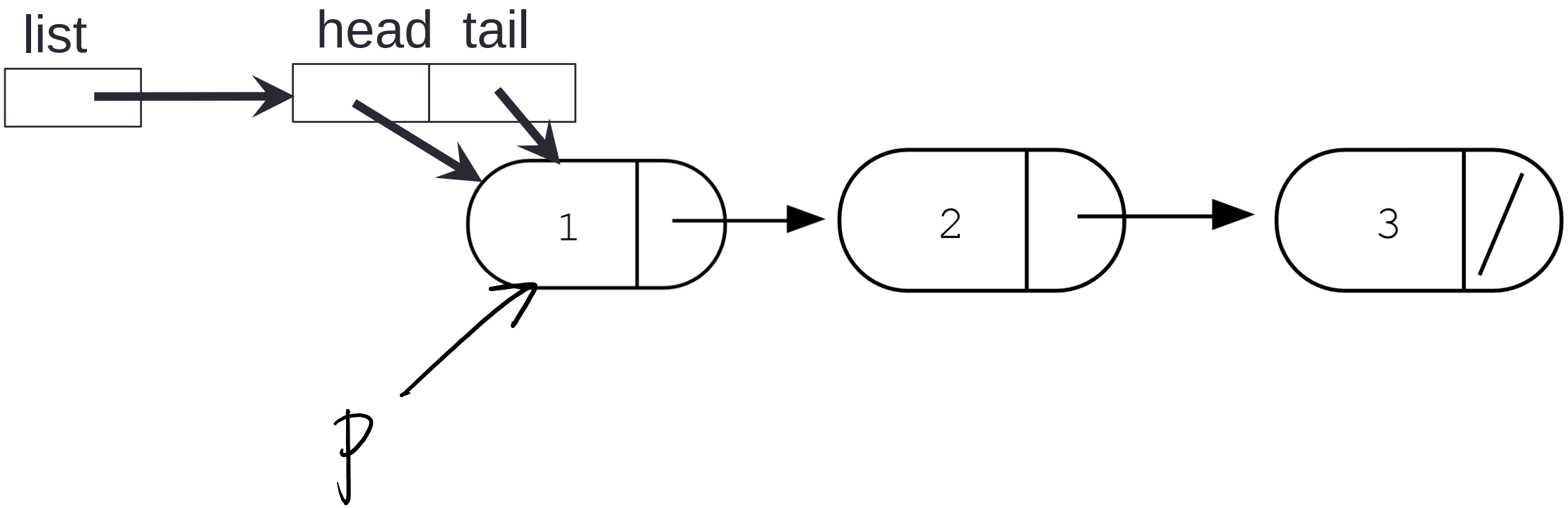


Case 3: Two node list(s)



1. Each pair: Write all possible test cases for list of size 2
2. Write the code for that passes each case
3. Exchange and review your code
4. Come up with the next logical test case

void deleteNodeRecursive(LinkedList *list, int value)



Next time

- Wrap up and review