CS 101
Computer Programming and Utilization

Arrays

Suyash P. Awate

Arrays

- Motivation
  - Suppose we want to analyze heights and weights of kids in school
  - Creating distinct names of (so many) variables can be laborious
  - How to organize all these variables ?
- C++ has two such mechanisms
  - Arrays (older way; also present in C)
    - Simpler
  - Vectors (newer way; absent in C)
    - Recommended for C++

Arrays

- "Array": a collection (sequence) of variables (of the same data type)
- Consider a C++ statement "int A[100];"
  - Defines 100 variables with names as "A[0]", "A[1]", …, "A[99]"
  - Name of array is "A"
  - "A[0]", "A[1]", …, "A[99]" are "elements" of array
  - Number of elements in array = "size" of array
    - In our case, 100
  - Content within "[ ]" is the "index" of each element
    - Start from 0
    - Non-negative integers
    - Array index must always be within [0, size-1]
  - Memory needed for array = memory needed for data type * size of array
    - In our case, 4 * 100 bytes

Arrays

- Operations
  - Typical operations on a variable can be done on any element of an array

```
int a[1000];
cin >> a[0];  // reads from keyboard into a[0]

a[7] = 2;     // stores 2 in a[7].

int b = 5*a[7]; // b gets the value 10.

int d = gcd(a[0],a[7]); // gcd is a function as defined earlier

a[b*2] = 234;   // index: arithmetic expression OK
```

Arrays

- Definition and initialization together
  - Examples

```
float pqr[5] = {15.0, 30.0, 12.0, 40.0, 17.0};

float pqr[] = {15.0, 30.0, 12.0, 40.0, 17.0};

int x, squares[5] = {0, 1, 4, 9, 16}, cubes[]={0, 1, 8, 27};
```

- Notation (for theoretical purposes; not in C++)
  - A[i..j] is the sub-array { A[k] : k = i, …, j }

Arrays

- Analysis of marks of students in class
  - Creating storage for marks for all students
    - How many students ? Exactly 100. What if this isn't known at compile time ? [later]

```
float marks[100]; // marks[i] stores the marks of roll number i+1
```

  - Inputting marks of each student

```
for(int i=0; i<100; i++){
  cout << "Marks for roll number " << i+1 << ": ";
  cin >> marks[i];
}
```

    - Assume: Roll numbers from 1…100; Student with roll-number R has marks stored in marks[R-1]
  - Outputting marks of students based on roll number queried
    - Stop when input roll-number is -1

```
while(true){
  cout << "Roll number: ";
  int rollNo;
  cin >> rollNo;

  if(rollNo == -1) break;

  cout << "Marks: " << marks[rollNo-1] << endl;
}
```

Arrays

- Who (all) got the highest marks ?
  - Step 1: Searching in the array for the highest marks

```
float maxSoFar = marks[0];
for(int i=1; i<100; i++){   // i starts at 1 because we already took marks[0]
  if(maxSoFar < marks[i])
    maxSoFar = marks[i];
}
```

  - Step 2: Outputting roll numbers for all students who scored highest

```
for(int i=0; i<100; i++)
   if(marks[i] == maxSoFar)
     cout << ``Roll number `` << i << `` got maximum marks.'' << endl;
```

    - If you want to be extra safe about numerical errors in floating-point comparisons, then replace if condition by something like “abs (marks[i] – maxSoFar) < 1e-5”

Arrays

- What if roll numbers don't follow some specific ordered sequence ?
  - We need two arrays
    - "for" loop reads in roll numbers and marks
    - "while" loop output marks for a specified roll number

```
int rollno[100];
double marks[100];

for(int i=0; i<100; i++) cin << rollno[i] << marks[i];

while(true){
  int r; cin >> r;  // roll number whose marks are requested
  if(r == -1) break;
  for(int i=0; i<100; i++)
     if(rollno[i] == r) cout << marks[i] << endl;
}
```

  - What if input roll number r is absent in array ? We should inform accordingly

```
int i;
for(i = 0; i<100; i++){
   if(rollno[i] == r){ cout << marks[i] << endl; break;}
}
if(i >= 100) cout << "Invalid roll number.\n";
```

Arrays

- Histogram of marks of students in class (assume 0 <= marks < 100)
  - Define

```
int count[10]; // count[i] will store the number of marks in the range
               // i*10 through (i+1)*10 -1.
```

  - Initialize

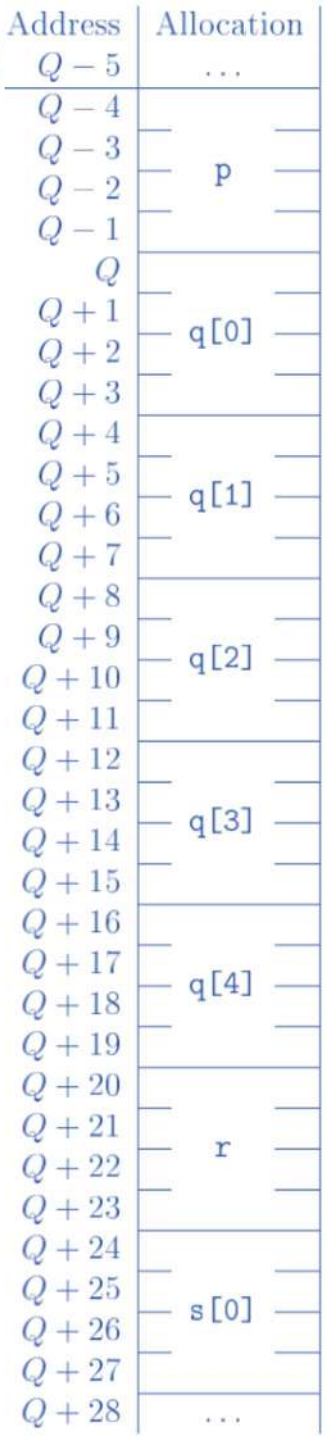
```
for(int i=0; i<10; i++)
    count[i]=0;
```

  - Populate

```
for(int i=0; i< 100; i++){
    float marks;
    cin >> marks;
    int index = marks/10;
    if(index >= 0 && index <= 9) count[index]++;
    else cout << "Marks are out of range." << endl;
}
```

Arrays

```
int p=5, q[5]={11,12,13,14,15}, r=9;
float s[10];
```



- Memory handling for arrays
- Location of “q[0]” starts at byte with address Q
- Location of “q[1]” starts at byte Q+1*(sizeof(int)) = byte Q+4
- Location of “q[2]” starts at byte Q+2*(sizeof(int)) = byte Q+8
- Location of “q[i]” starts at byte Q+i*(sizeof(int)) = byte Q+4i
- What if we execute a statement: “q[5] = 20”
  - This may end up changing value of r, if r was allocated right after q[4]
  - But this is dangerous, in general; can lead to seg fault
- Array name is a constant pointer
  - Statement “q = …;” leads to a compilation error

Arrays

```
int n;
cin >> n;
int a[n];  // Not allowed by the C++ standard.
```

- Array size cannot be variable [textbook]
- C++ requires “n” to be a compile-time constant value
  - Can be stored in a variable defined to be const
- Defining “int a[100];” hardcodes size to 100. If we want to change code later, we need to replace instances of 100 or 100-1=99 in source code
  - Cumbersome, risky

```
const int NMAX = 1000;
int a[NMAX], b[NMAX];
```

- Better ways
  - Define “n” as a const int, write all code in terms of “n”
  - Define array of a maximum size; allow user to use any sub-array of size <= maximum size

```
const int NMAX = 1000;
int a[NMAX], b[NMAX], nactual;
cin >> nactual;
assert(nactual <= NMAX);
```

Arrays

- Array size
  - Some compilers (e.g., current version of GNU C++) allow array size to be determined at run time
  - Others (e.g., Visual Studio) want array size to be fixed at compile time
    - Feature of standard C, but not included in C++
  - www.reddit.com/r/cpp_questions/comments/t2mkbb/one_compiler_allows_declaring_cstyle_arrays_with/
  - But once an array is defined, its size cannot be changed

```
int main ()
{
  int n = 10;
  int A[n];
}
```

```
 g++ -Wall -pedantic  array.cpp
array.cpp:4:9: warning: variable length arrays are a C99 feature [-Wvla-extension]
  int A[n];
        ^
array.cpp:4:9: note: read of non-const variable 'n' is not allowed in a constant expression
```

  - en.wikipedia.org/wiki/C99

Arrays

- Arrays and pointers
  - [] is an operator; the "subscript" operator
  - P[J] is an expression involving operator [] and arguments P and J, where
    P must be an address of some type T, and
    J must be an integer
  - [] operator is commutative !
    - en.wikipedia.org/wiki/Pointer_(computer_programming)

```
double speed[]={1.25, 3.75, 4.3, 9.2};
double *s;
s = speed;
cout << s[0] << endl;
s[1] = 3.9;
cout << speed[1] << endl;
```

```
int array[5];        /* Declares 5 contiguous integers */
int *ptr = array;    /* Arrays can be used as pointers */
ptr[0] = 1;          /* Pointers can be indexed with array syntax */
*(array + 1) = 2;    /* Arrays can be dereferenced with pointer syntax */
*(1 + array) = 2;    /* Pointer addition is commutative */
2[array] = 4;        /* Subscript operator is commutative */
```

Arrays

- Pointer arithmetic
  - Let '**x**' be name of **array of type T**
  - Then x is a pointer
    with value as starting address of array
  - Then, C++ expression "x+i"
    has a special interpretation,
    i.e., location of element of **type T**
    at starting location x + i * sizeof(T)
  - C++ expression "*(x+i)" means "x[i]"

```
#include <iostream>

using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int  *ptr;

   // let us have array address in pointer.
   ptr = var;

   for (int i = 0; i < MAX; i++) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;

      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;

      // point to the next location
      ptr++;
   }

   return 0;
}
```

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

www.tutorialspoint.com/cplusplus/cpp_pointer_arithmatic.htm

Arrays

- Pointer arithmetic

```
a = 0x3fffd1a
sizeof(short) = 2
          p = 0x3fffd1a    *p = 22          sum = 22
          p = 0x3fffd1c    *p = 33          sum = 55
          p = 0x3fffd1e    *p = 44          sum = 99
end = 0x3fffd20
```

```
int main()
{ const int SIZE = 3;
  short a[SIZE] = {22, 33, 44};
  cout << "a = " << a << endl;
  cout << "sizeof(short) = " << sizeof(short) << endl;
  short* end = a + SIZE;        // converts SIZE to offset 6
  short sum = 0;
  for (short* p = a; p < end; p++)
  { sum += *p;
    cout << "\t p = " << p;
    cout << "\t *p = " << *p;
    cout << "\t sum = " << sum << endl;
  }
  cout << "end = " << end << endl;
}
```

Arrays

- Array as argument to function call
- We want to write a function to compute sum of elements in an array
  - What arguments do we need to pass information about array ?
    - Array name as a pointer to a variable to some type T
    - Array size; can we use “sizeof(arrayName) / sizeof(arrayName[0])” ?

```
float sum(float* v, int n){  // function to sum the elements of an array
  float s = 0;
  for(int i=0; i<n; i++)
    s += v[i];

  return s;
}
```

```
int main(){
  float a[10] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0},  asum;
  asum = sum(a, 5);  // second argument should be array length
}
```

Arrays

- Array as argument to function call
  - Argument v can be defined as “float v[]” instead of “float * v”
  - Are all values in array passed to function ?
    - No
  - What happens if we call sum(a,3) ?
    - Sum of first 3 elements
    - Function takes size of array through the second argument
  - What happens if we call sum(a,10) ? Will function run ?
    - May not run (runtime error / crash). If runs, may get junk values.
  - If function has statement “a[0] = 0;” will that change be seen in main ?
    - Yes

```
int main(){
  float a[10] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0},  asum;
  asum = sum(a, 5);  // second argument should be array length
}
```

Arrays

- Array as argument to function call
  - Example code: array.cpp

```
#include <iostream>

void fun (int arrayArgument[])
{
  // use of sizeof isn't fine here
  for (int i = 0;
       i < sizeof (arrayArgument) / sizeof (arrayArgument[0]);
       i++)
    std::cout << arrayArgument[i] << "\t";
  std::cout << "\n";
}

int main()
{
  int array[4] = { 1, 2, 3, 4 };

  // use of sizeof is fine here
  for (int i = 0;
       i < sizeof (array) / sizeof (array[0]);
       i++)
    std::cout << array[i] << "\t";
  std::cout << "\n";

  fun (array);
}
```

Arrays

- Array as argument to function call

```
./a.out
1       2       3       4
1       2
```

```
 g++ array.cpp
array.cpp:7:19: warning: sizeof on array function parameter will
 return size of 'int *' instead of 'int[]' [-Wsizeof-array-argum
ent]
       i < sizeof (arrayArgument) / sizeof (arrayArgument[0]);
                  ^
array.cpp:3:15: note: declared here
void fun (int arrayArgument[])
             ^
array.cpp:7:35: warning: 'sizeof (arrayArgument)' will return th
e size of the pointer, not the array itself [-Wsizeof-pointer-di
v]
       i < sizeof (arrayArgument) / sizeof (arrayArgument[0]);
           ~~~~~~~~~~~~~~~~~~~~~~ ^
array.cpp:3:15: note: pointer 'arrayArgument' declared here
void fun (int arrayArgument[])
             ^
2 warnings generated.
```

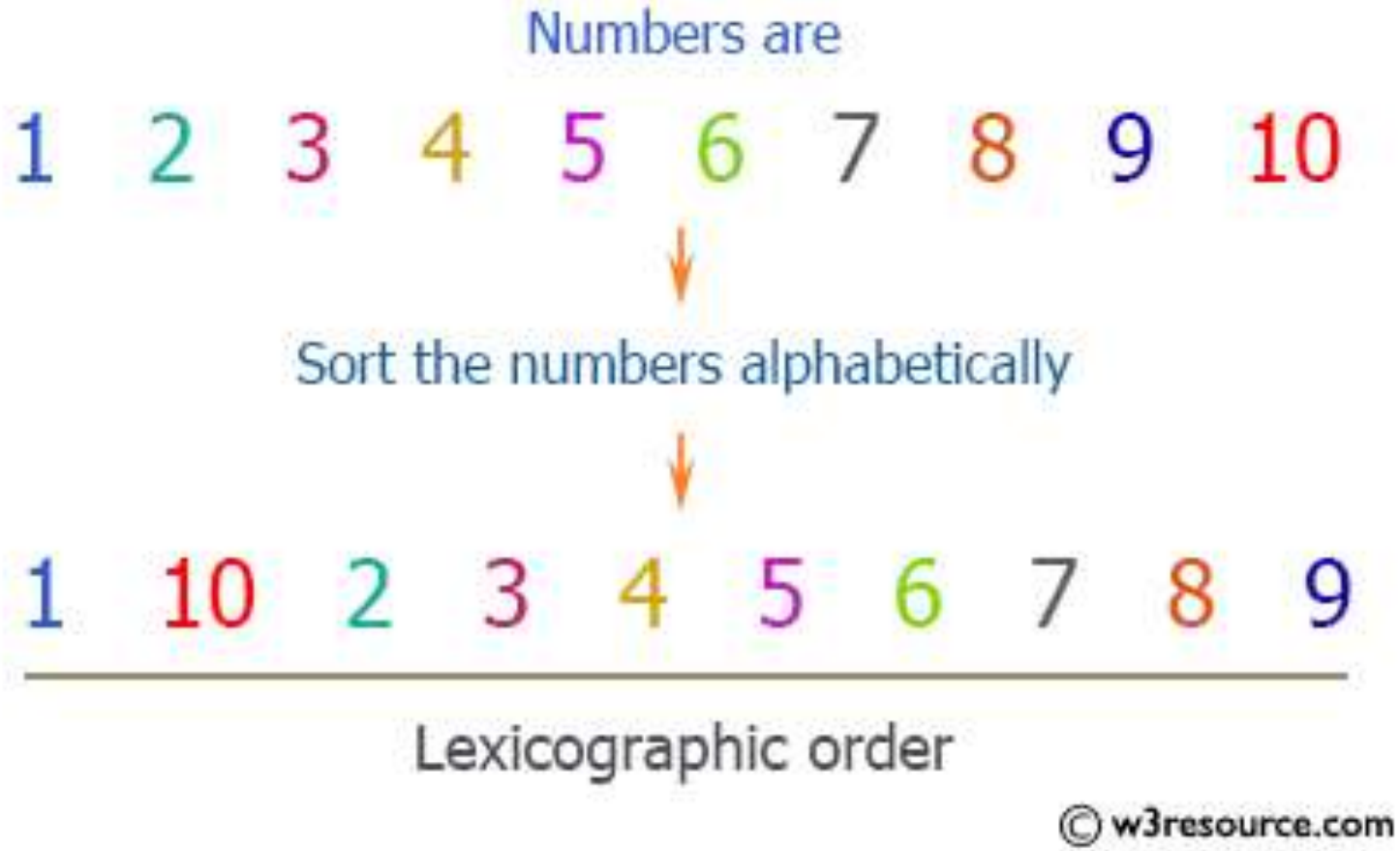
Arrays

- Find (some) roll number that had maximum marks

```
int argmax(float marks[], int L)
// marks = array containing the values
// L = length of marks array.  required > 0.
// returns maxIndex such that marks[maxIndex] is largest in marks[0..L-1]
{
  int maxIndex = 0;
  for(j = 1; j<L; j++)
    if( marks[maxIndex] > marks[j]) // bigger element found?
      maxIndex = j;                // update maxIndex.
  return maxIndex;
}
```

Arrays

- Sorting = process of (re)arranging elements of array/list in order (assuming there is way to order them)
  - e.g., numerical order, lexicographical (dictionary) order
- Many algorithms for sorting have been very well studied



Arrays

- A sorting algorithm: Selection Sort (sorting in non-decreasing order)
- Strategy 1
  - Input unsorted array
  - Find smallest element in unsorted array
  - Swap its position with element at first location in unsorted array
  - Redefine unsorted array as subarray with length reduced by 1
- Strategy 2
  - Replace smallest by largest
  - Replace first by last

Sorted sublist	Unsorted sublist	Least element in unsorted list
()	(11, 25, 12, 22, 64)	11
(11)	(25, 12, 22, 64)	12
(11, 12)	(25, 22, 64)	22
(11, 12, 22)	(25, 64)	25
(11, 12, 22, 25)	(64)	64
(11, 12, 22, 25, 64)	()	

Arrays

- Selection Sort
  - Strategy 1

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

Arrays

- Selection Sort
  - Strategy 2

```
void SelSort(float data[], int n)
// will sort in NON-DECREASING order.  different from above.
{
  for(int i=n; i>1; i--){
    int maxIndex = argmax(data,i);  // Find index of max in data[0..i-1]
    float maxVal = data[maxIndex];  // Exchange elements at
    data[maxIndex] = data[i-1];     //   index  maxindex
    data[i-1] = maxVal;             //   and index i-1.
  }
}
```

Arrays

- Selection Sort
  - Rough estimate of “time taken” / “work done” / “computational cost”
  - For array of size n, sorting algorithm calls argmax multiple times
  - In 1st call, argmax goes over subarray of size n
    - Work done proportional to n
  - In 2nd call, argmax goes over subarray of size n-1
  - …
  - Total work done = $n + (n-1) + (n-2) + \ldots + 2 = \sum_{i=2}^{i=n} i \approx n^2/2$

    = roughly proportional to n^2 (for large n)

Arrays

- Another application: handling polynomials $A(x) = \sum_{i=0}^{i=n-1} a_i x^i$
- Polynomials of (n-1)-degree are modeled by their coefficients, which can be stored in an array of size/"length" n
- Adding two (n-1)-degree polynomials A(x) and B(x)
  - Add their corresponding coefficients
  - Function takes as input arrays for coefficients of A and B
  - Where is result stored ?
    - Can we allocate array inside function and return that ?
    - We can pass, as argument, array storing coefficients of resulting polynomial, say, C

```
void addp(float a[], float b[], float c[], int n){
// addends, result, length of the arrays.
  for(int i=0; i<n; i++)  c[i] = a[i] + b[i];
}
```

Arrays

- Another application: handling polynomials
- Product of two (n-1)-degree polynomials A(x) and B(x)
  - Resulting polynomial will have degree 2(n-1), and will be stored in array of size 2(n-1) + 1 = 2n-1
  - Terms $a_j x^j$ get multiplied with terms $b_k x^k$ to give terms $a_j b_k x^{j+k}$

```
void prodp(float a[], float b[], float d[], int n){
// a,b must have n elements, product d must have 2n-1
  for(int i=0; i<2*n-1; i++) d[i] = 0;
  for(int j=0; j<n; j++)
    for(int k=0; k<n; k++) d[j+k] += a[j]*b[k];
}
```

Practice Examples for Lab: Set 11

- 1

You are to write a program which takes as input a sequence of positive integers. You are not given the length of the sequence before hand, but after all the numbers are given, a -1 is given, so you know the sequence has terminated. You are required to print the 10 largest numbers in the sequence. Hint: use an array of length 10 to keep track of the numbers that are candidates for being the top 10.

- 2

Suppose in the previous problem you are asked to report which are the 10 highest values in the sequence, and how frequently they appear. Write a program which does this.

- 3

Write a program which takes as input two vectors (as defined in mathematics/physics) – represent them using arrays – and prints their dot product. Make this into a function.

Practice Examples for Lab: Set 11

- 4

Suppose we are given the x, y coordinates of n points in the plane. We wish to know if any 3 of them are collinear. Write a program which determines this. Make sure that you consider every possible 3 points to test this, and that you test every triple only once. The coordinates should be represented as `floats`. When you calculate slopes of line segments, because of the floating point format, there will be round-off errors. So instead of asking whether two slopes are equal, using the operator `==`, you should check if they are approximately equal, i.e. whether their absolute difference is small, say 10^{-5}. This is a precaution you need to take when comparing floating point numbers. In fact, you should also ask yourself whether the slope is a good measure to check collinearity, or whether you should instead consider the angle, i.e. the arctangent of the slope.

Practice Examples for Lab: Set 11

- 5

Write a function which given polynomials $P(x), Q(x)$ returns their *composition* $R(x) = P(Q(x))$. Say $P(x) = x^2 + 3x + 5$ and $Q(x) = 3x^2 + 5x + 9$. Then $R(x) = (3x^2 + 5x + 9)^2 + 3(3x^2 + 5x + 9) + 5$.

- 6

Suppose we are given an array `marks` where `marks[i]` gives the marks of student with roll number `i`. We are required to print out the marks in non-increasing order, along with the roll number of the student who obtained the marks. Modify the sorting algorithm developed in the chapter to do this. Hint: Use an additional array `rollNo` such that `rollNo[i]` equals `i` initially. As you exchange marks during the course of the selection sort algorithm, move the roll number along with the marks.

Arrays

- Character arrays
  - e.g., “char name[20], residence[100];”
  - If name is 7 letters long, those letters can be stored in name[0] to name[6]
  - When we print name, we don’t want to print contents name[7] onwards
  - Instead of storing length of name explicitly,
    C++ expects a special character in array after the valid contents
    - In our case, name[6] will contain ‘\0’, or NUL or ASCII value 0
    - This is inherited from C
    - Thus, a 7-letter long name needs 8 characters of storage

```
char name[20] = "Shivaji";
char residence[50] = "Main Palace, Raigad";

char name[] = "Shivaji";
char residence[] = "Main Palace, Raigad";
```

Arrays

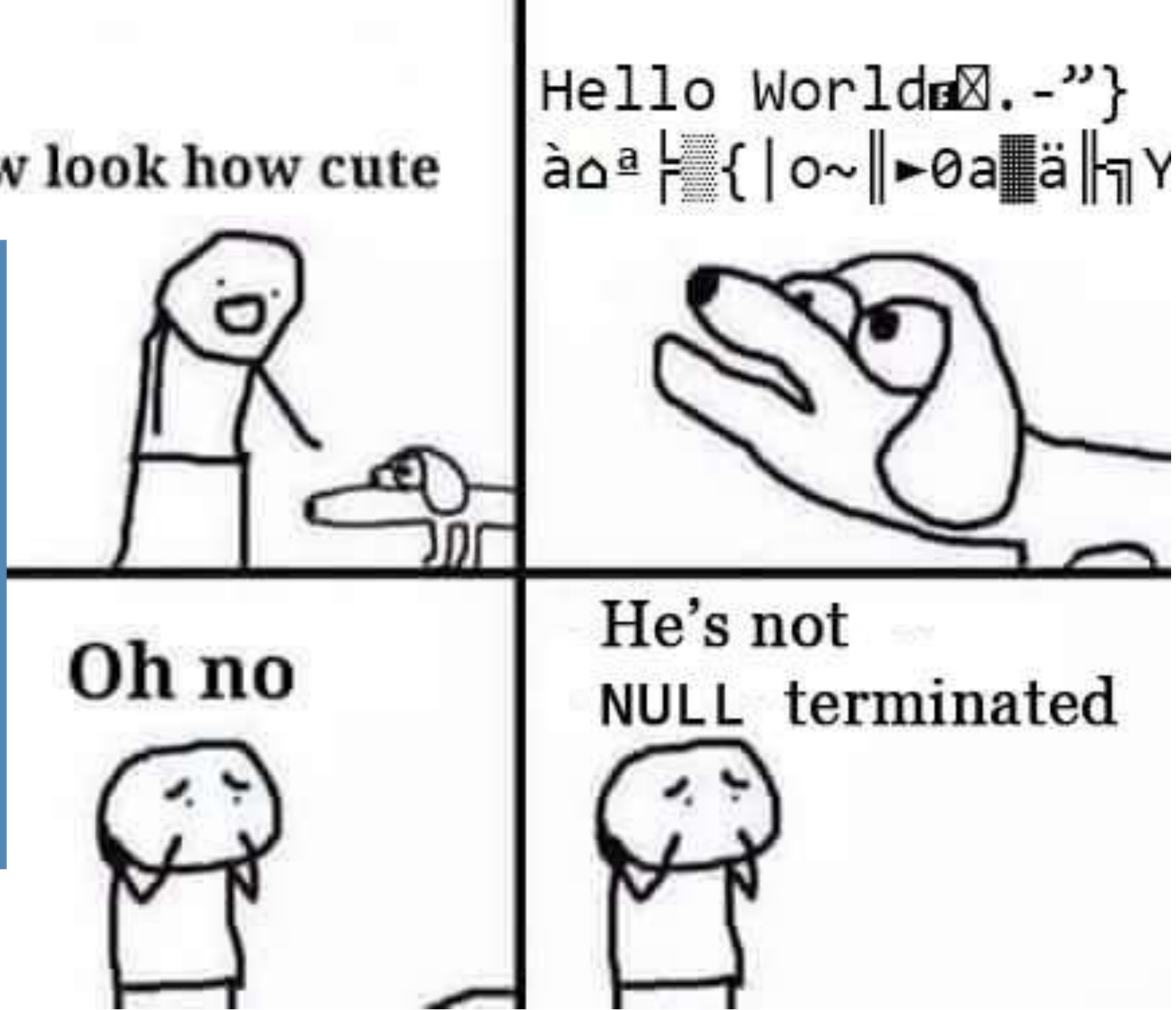
- Character arrays

Roses are red

This program might halt

I forgot a null terminator

Segmentation fault

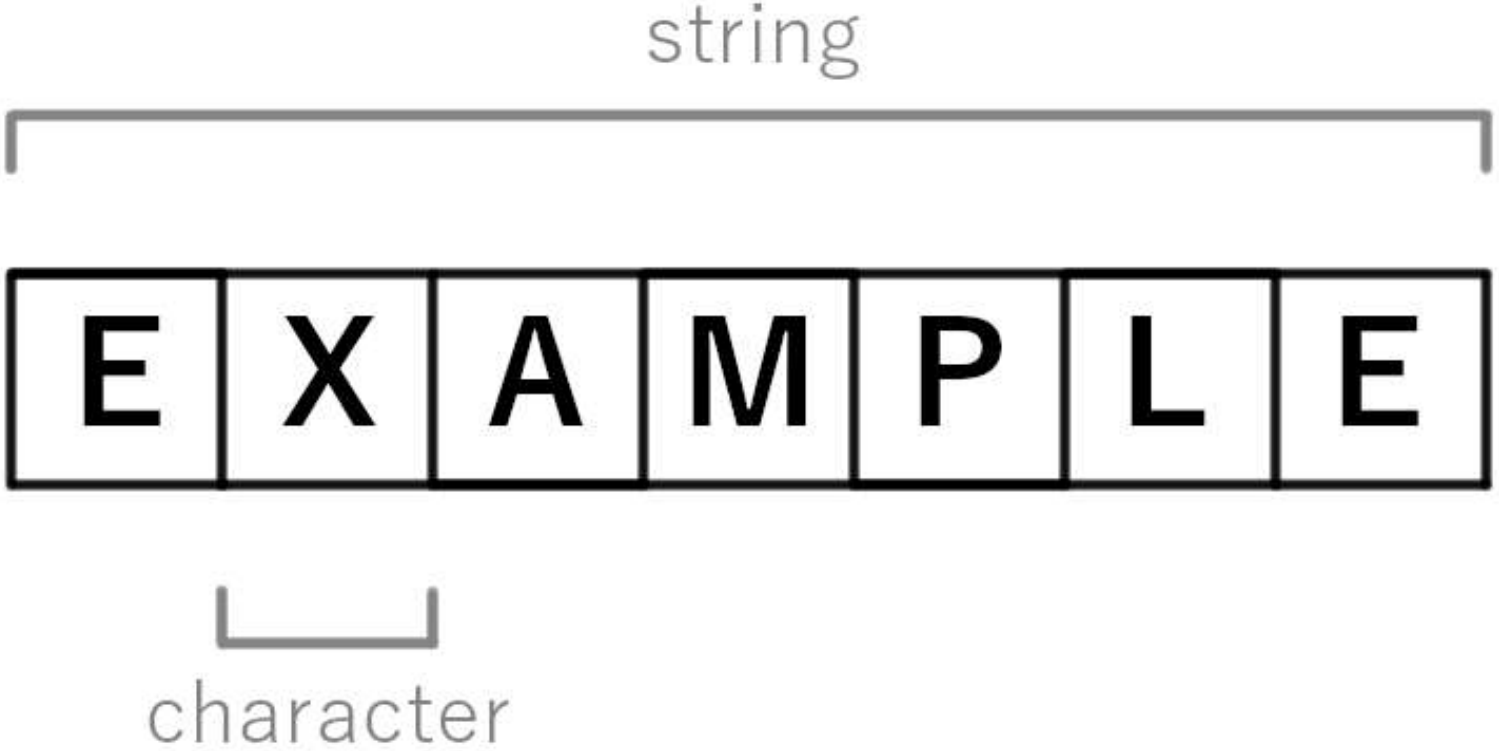


Arrays

- Character arrays
  - Actually,
    C++ calls it NUL (= '\0')
    instead of NULL (= 0)
    that is used with pointers

Arrays

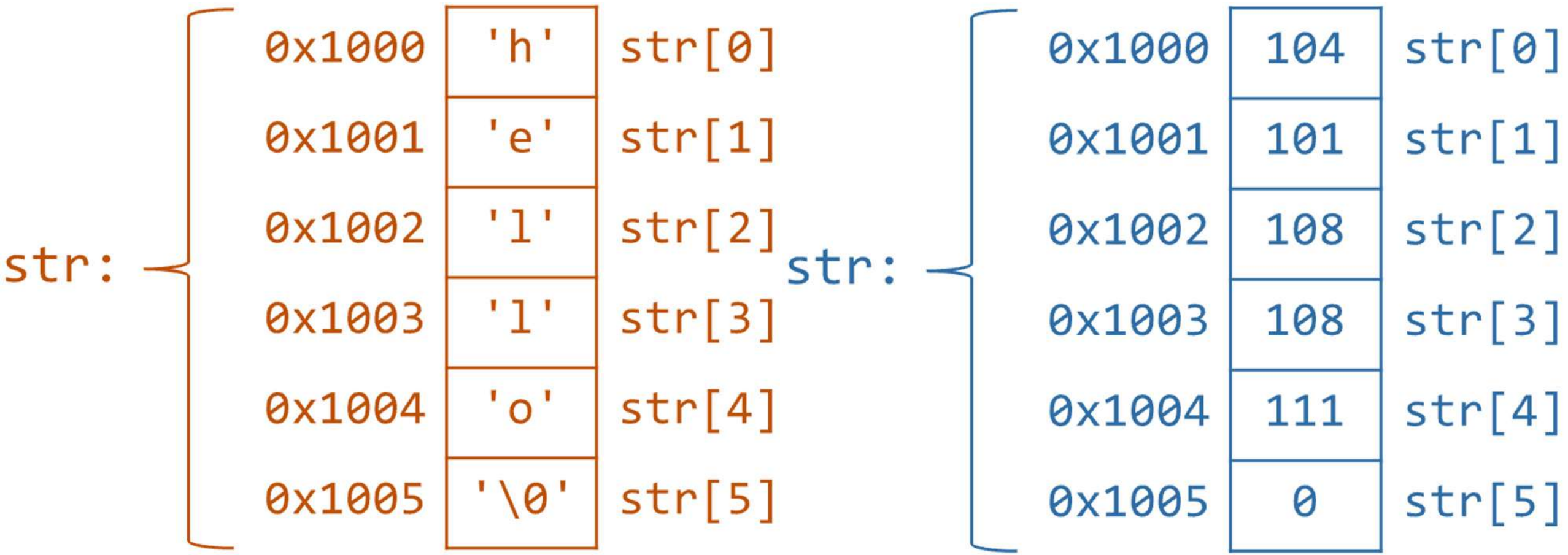
- Concept of a "string"
  - en.wikipedia.org/wiki/String_(computer_science)
  - "In computer programming,
    a string is traditionally a sequence of characters,
    either as a literal constant or as some kind of variable."
  - "A primary purpose of strings is to store human-readable text,
    like words and sentences."



Arrays

- C-style data structure for storing a string
  - eecs280staff.github.io/notes/05_Strings_Streams_IO.html

```
char str[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```



Arrays

- Character arrays

```
char str1[6] = "hello";
char str2[6] = "hello";
char str3[6] = "apple";
char *ptr = str1;


// Test for equality?
str1 == str2;

// Copy strings?
str1 = str3;

// Copy through pointer?
ptr = str3;
```

Arrays

- Character arrays
  - To output:
    cout << name;
  - To input:
    cin >> name;

```
char name[20];
cout << "Please type your name: ";
cin >> name;
```

    - From the input, initial whitespace characters will be ignored
    - Characters starting from first non-whitespace character to subsequent whitespace character will be read into name
      - e.g., if you type “james bond”, then only “james” would be stored as the name
  - To get a name including spaces, use the getline(…, …) function
  - What if the user typed in a name that is longer than 19 letters ?
    - Overflow; risky; may overwrite memory allocated to another variable;
      or may try to access memory reserved for something else leading to a system crash
  - Safer way: cin.getline (nameOfCharArray, sizeOfCharArray)
    - Here, input after sizeOfCharArray-1 characters will be ignored

Arrays

- Character array processing
- Find length of text stored in char array

```
int length(char *txt){
  //precondition: txt points to sequence of '\0' terminated characters
  int L=0;
  while(txt[L] != '\0') L++;
  return L;
}
```

- Copy string in char array 'source' to a char array 'destination'

```
void scopy(char destination[], char source[])
// precondition: '\0' must occur in source.  destination must be long
// enough to hold the entire source string + '\0'.
{
  int i;
  for(i=0; source[i] != '\0'; i++)
    destination[i]=source[i];
  destination[i]=source[i];    // copy the '\0' itself
}
```

Arrays

- Character array processing
- Find lexicographic ordering of 2 strings (ordering in dictionary)
  1. We compare corresponding characters one by one
  2. We continue until
     we find a difference,
     or each end of one string
  - At iteration within this algorithm, we have one of following cases:
    1. We reach \0 in both strings → both strings equal
    2. We reach \0 in one string but not the other → one string is prefix of other
       - Prefix string occurs earlier in dictionary
    3. Char in strings aren't \0 and are unequal → one string occurs earlier in dictionary

Arrays

- Character array processing
- Lexicographic ordering of 2 strings

```
main(){
  char a[40], b[40];
  cin.getline(a,40);
  cin.getline(b,40);
  cout << a << " " << compare(a,b) << " " << b << endl;
```

```
char compare(char a[], char b[])
// returns '<' if a is smaller, '=' if equal, '>' if b is smaller.
{
  int i = 0;
  while(true){                    // Invariant: a[0..i-1] == b[0..i-1]
    if(a[i] == '\0' && b[i] == '\0') return '=';
    if(a[i] == '\0') return '<';
    if(b[i] == '\0') return '>';
    if(a[i]<b[i]) return '<';
    if(a[i]>b[i]) return '>';
    i++;
  }
}
```

Arrays

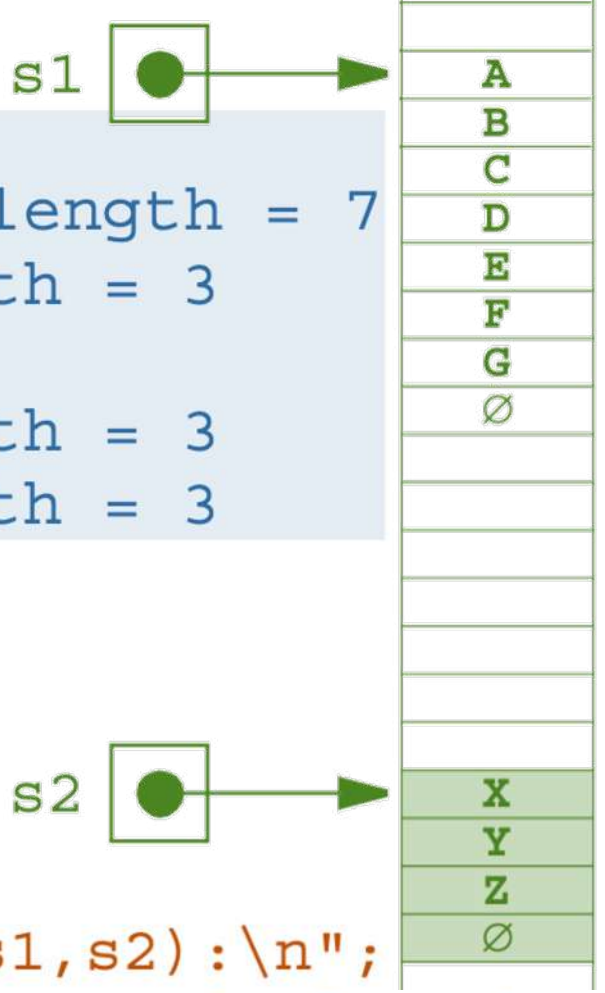
- Character-array operations in <cstring>: #include <cstring>
  - cplusplus.com/reference/cstring/
  - Function: strlen(…) returns length of char array passed to it

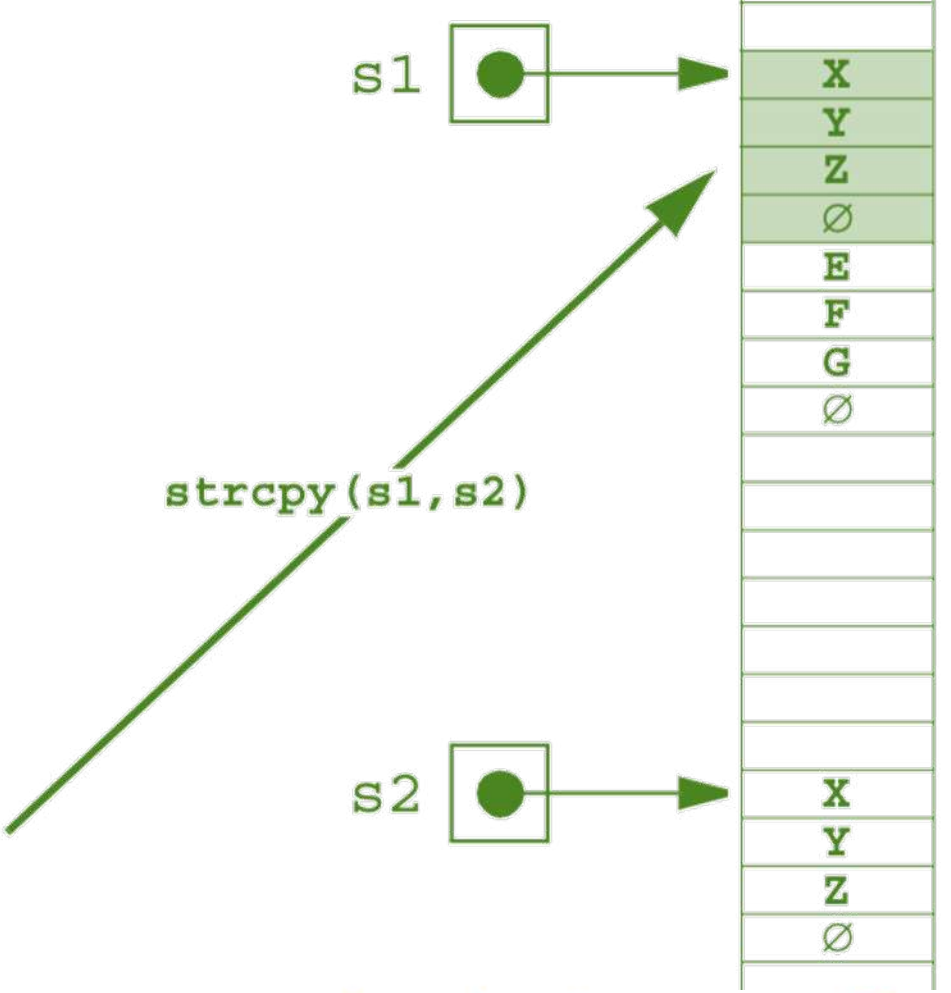
  - Functions to locate characters and sub-strings within a string
  - If not found, each such function returns NULL
  - strchr(s, c): returns pointer to leftmost instance of character c within string s
  - strrchr(s, c): returns pointer to rightmost instance of character c within string s
  - strstr(s, ss): returns pointer to leftmost instance of string ss within in string s

```
s = "The Mississippi is a long river."
strchr(s, ' ') points to s[3].
strchr(s, 's') points to s[6].
strrchr(s, 's') points to s[17].
strstr(s, "is") points to s[5].
strstr(s, "isi") returns NULL
```

Arravs

```
Before strcpy(s1,s2):
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strcpy(s1,s2):
        s1 = [XYZ], length = 3
        s2 = [XYZ], length = 3
```






```
#include <cstring>
#include <iostream>
int main()
{ char s1[] = "ABCDEFG";
  char s2[] = "XYZ";
  cout << "Before strcpy(s1,s2):\n";
  cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
  cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
  strcpy(s1,s2);
  cout << "After strcpy(s1,s2):\n";
  cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
  cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
```

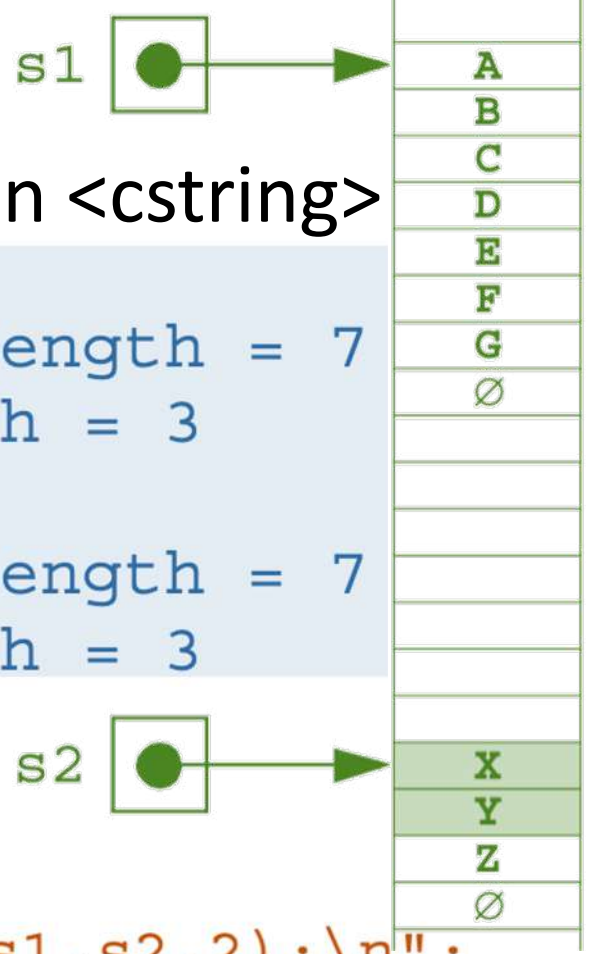
Arrays

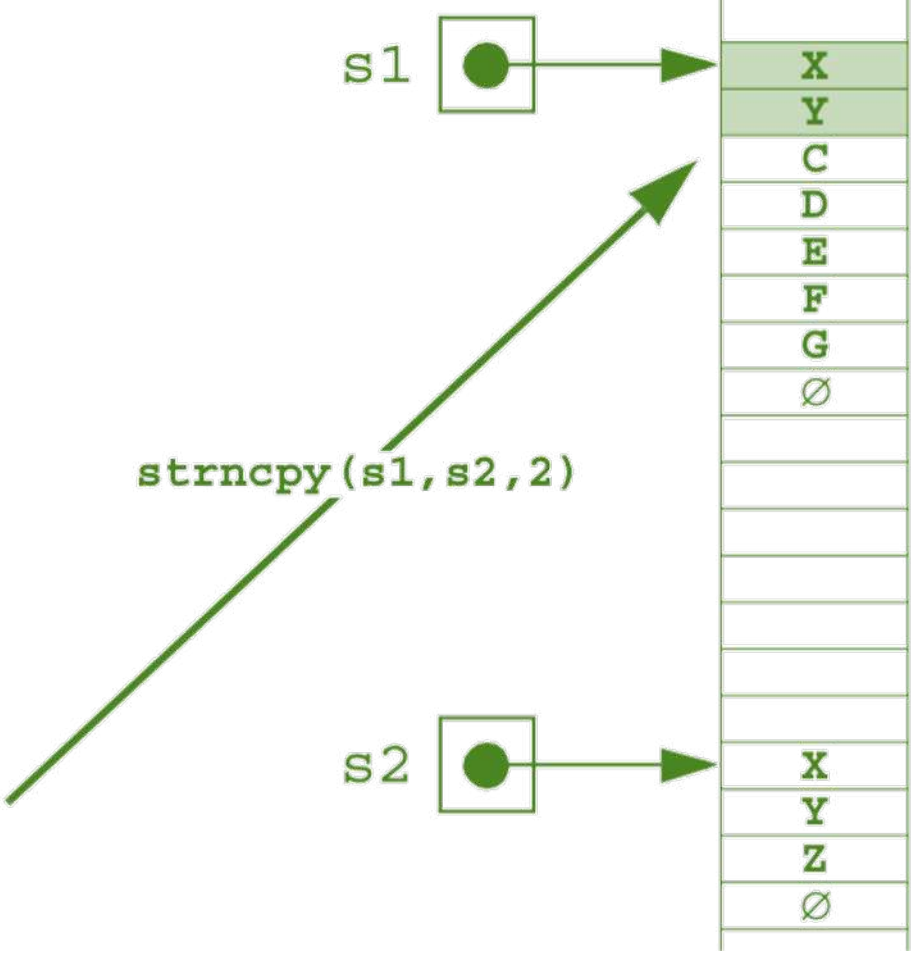
- Character-array operations in <cstring>
  - Function: strcpy(s1, s2) copies char array s2 into char-array s1
  - Returns updated s1
  - Beware:
    Length of s1 shall be long enough to contain s2 (including terminating NUL)
  - Beware:
    Char arrays pointed to by s1 and s2 shall not overlap

Arrays

- Character-array operations in <cstring>

```
Before strncpy(s1,s2,2):
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strncpy(s1,s2,2):
        s1 = [XYCDEFG], length = 7
        s2 = [XYZ], length = 3
```






```
int main()
{ char s1[] = "ABCDEFG";
  char s2[] = "XYZ";
  cout << "Before strncpy(s1,s2,2):\n";
  cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
  cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
  strncpy(s1,s2,2);
  cout << "After strncpy(s1,s2,2):\n";
  cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
  cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
```

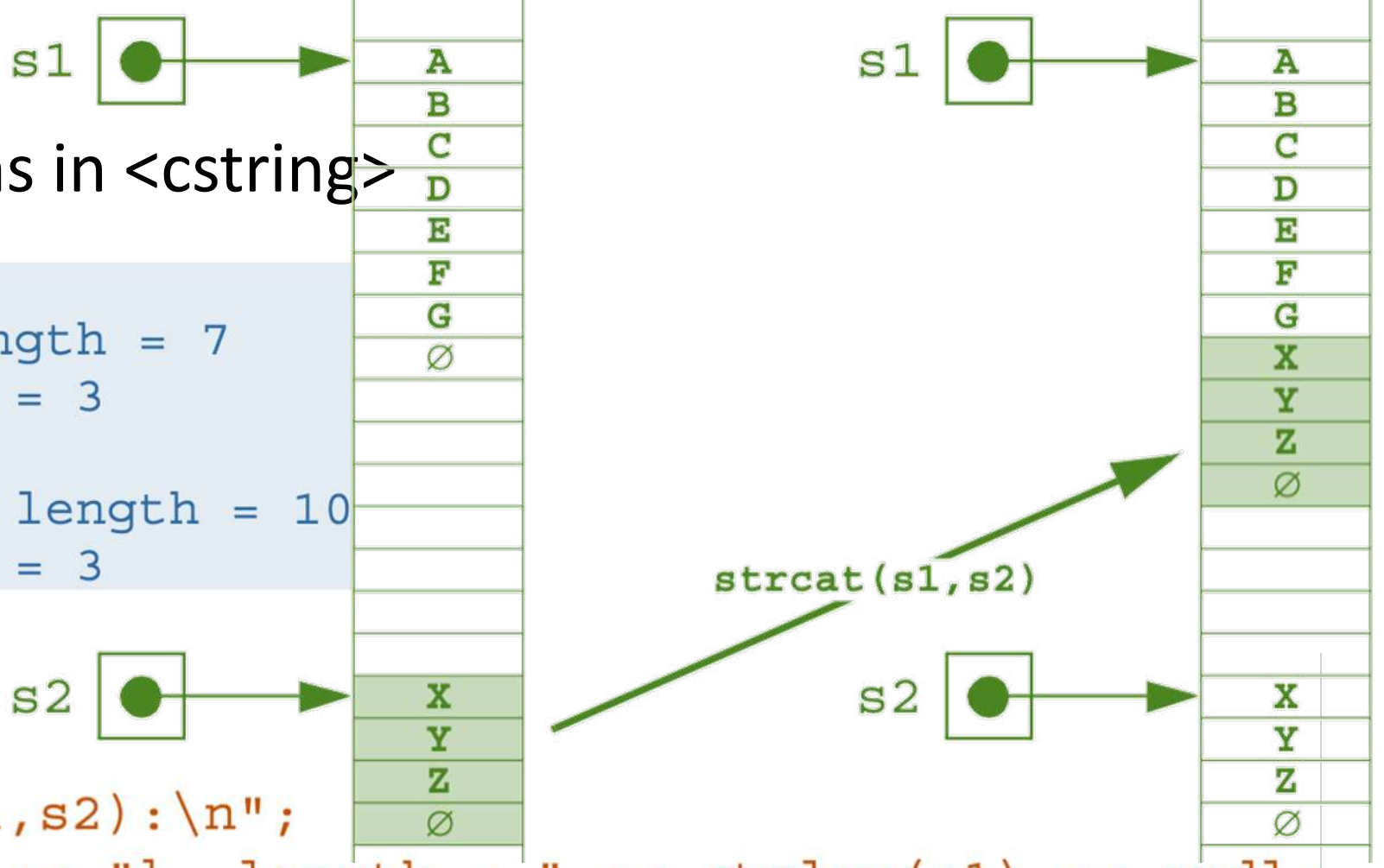
Arrays

- Character-array operations in <cstring>
  - Function:
    strncpy(s1, s2, n) copies first n characters from char-array s2 into char-array s1
  - Returns updated s1
  - Beware:
    If end of s2 is found before n characters have been copied,
    s1 is padded with NUL chars until a total of n chars have been written to it
  - Beware:
    No NUL character is implicitly appended at end of s1 if n <= length(s2).
    This might lead to s1 might not being NUL terminated (e.g., if s1 was uninitialized) !
  - Beware:
    Char arrays pointed to by s1 and s2 shall not overlap

Arrays

- Character-array operations in <cstring>

```
Before strcat(s1,s2):
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strcat(s1,s2):
        s1 = [ABCDEFGXYZ], length = 10
        s2 = [XYZ], length = 3
```




```
char s1[] = "ABCDEFG";
char s2[] = "XYZ";
cout << "Before strcat(s1,s2):\n";
cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
strcat(s1,s2);
cout << "After strcat(s1,s2):\n";
cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
```

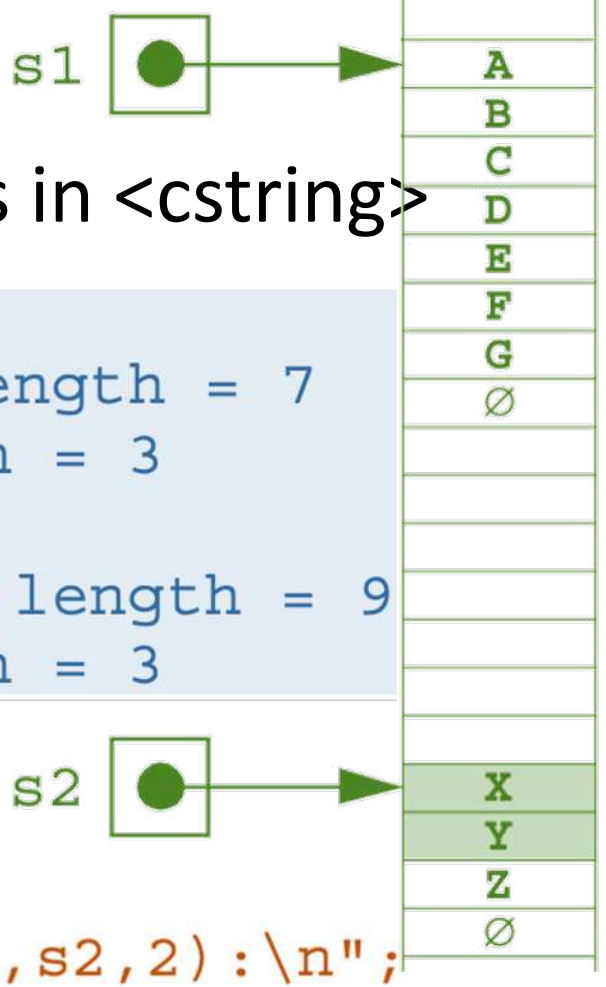
Arrays

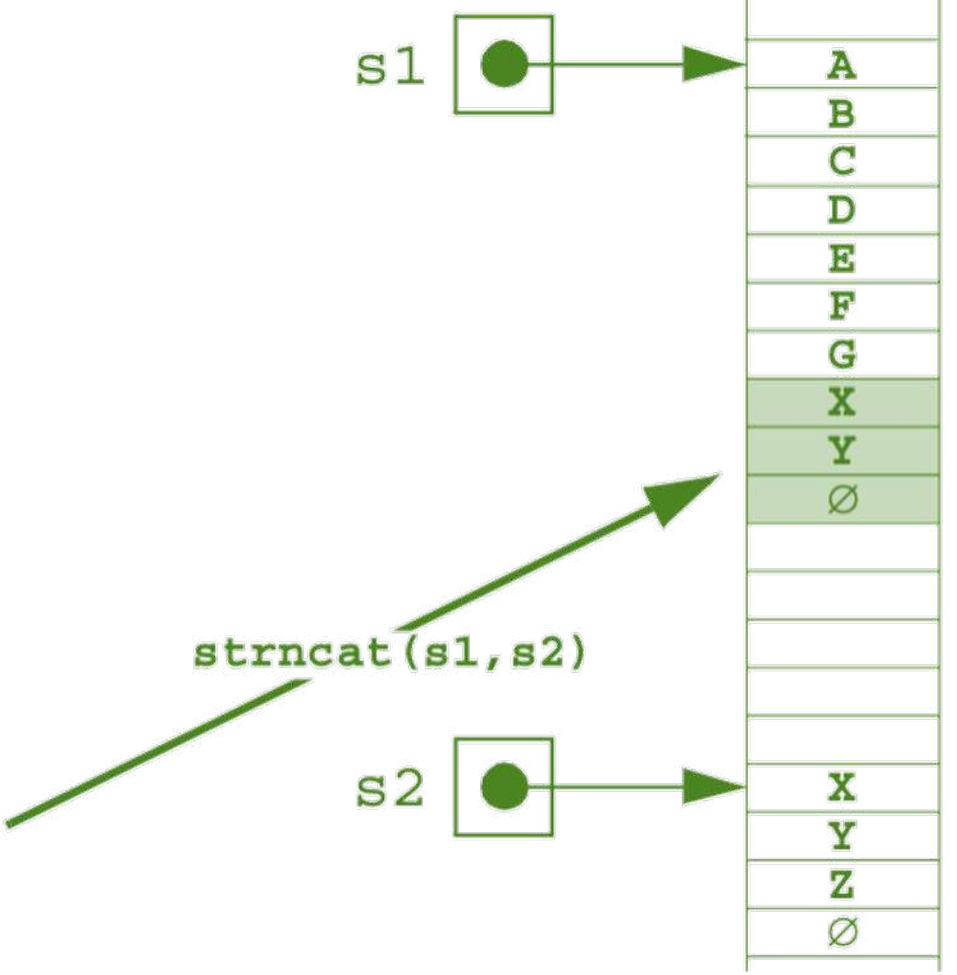
- Character-array operations in <cstring>
  - Function: strcat(s1,s2) appends a copy of char-array s2 to char-array s1
  - Returns pointer to joined string
  - Beware:
    Array s1 should be large enough
    (i.e., there should be enough space in memory after s1)
    to contain concatenated resulting string, including additional NUL character
  - Beware:
    If any of extra bytes following s1 that are needed to copy s2 are in use by any other object,
    then all of s1 and its appended s2 may be copied to some other free section of memory
  - Beware:
    Char arrays pointed to by s1 and s2 shall not overlap

Arrays

- Character-array operations in <cstring>

```
Before strncat(s1,s2,2):
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strncat(s1,s2,2):
        s1 = [ABCDEFGXY], length = 9
        s2 = [XYZ], length = 3
```






```
char s1[] = "ABCDEFG";
char s2[] = "XYZ";
cout << "Before strncat(s1,s2,2):\n";
cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
strncat(s1,s2,2);
cout << "After strncat(s1,s2,2):\n";
cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
```

Arrays

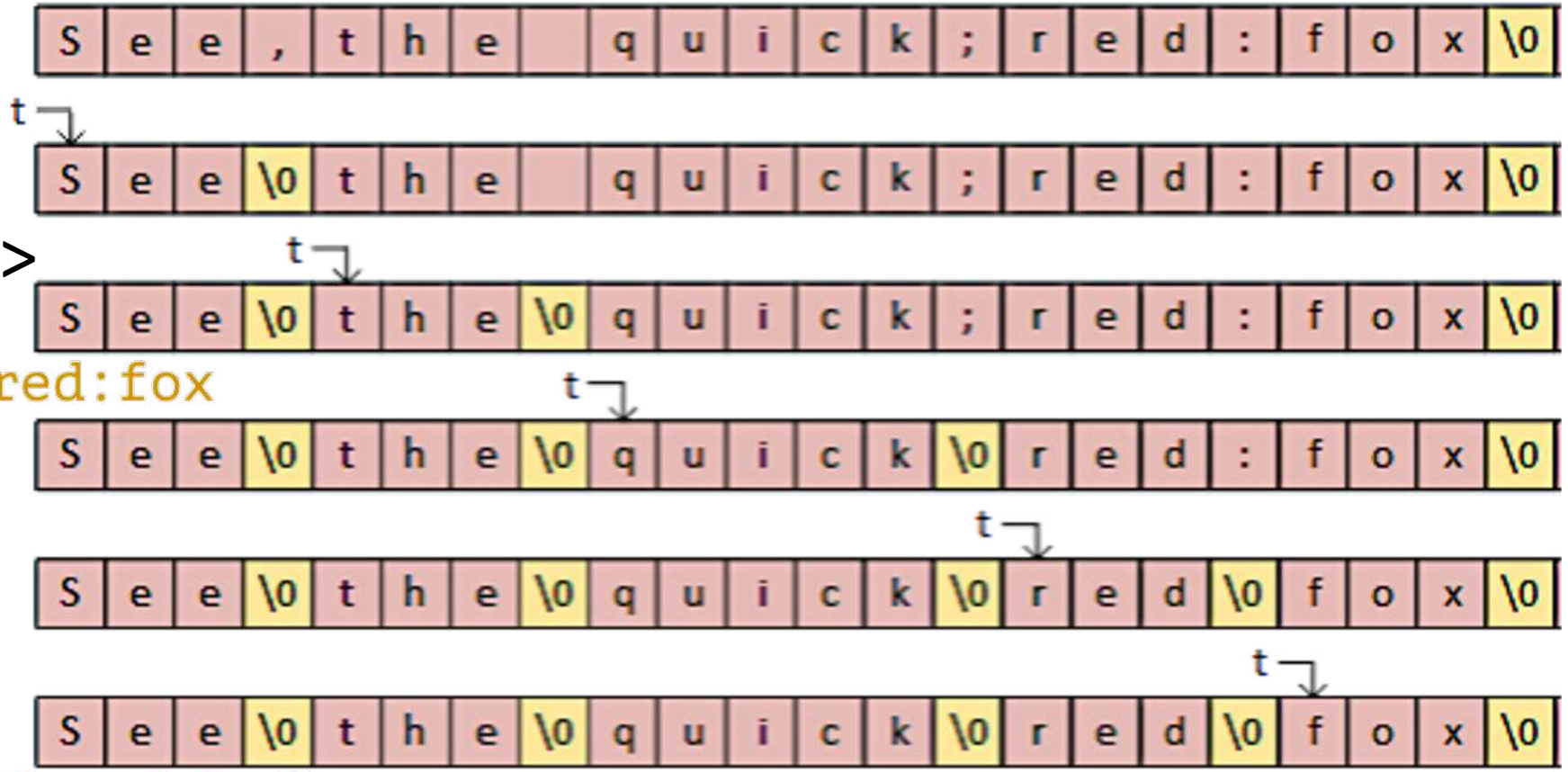
- Character-array operations in <cstring>
  - Function: strncat(s1,s2,n) appends a copy of first n chars from char-array s2 to char-array s1
  - If L=length(s2) < n, then only L chars appended
  - Returns pointer to joined string
  - Beware:
    Array s1 should be large enough
    (i.e., there should be enough space in memory after s1)
    to contain concatenated resulting string, including additional NUL character
  - Beware:
    If any of extra bytes following s1 that are needed to copy s2 are in use by any other object,
    then all of s1 and its appended s2 may be copied to some other free section of memory

Arrays

- Character-array operations in <cstring>
  - "Tokenize" (e.g., words) in a string: a sequence of calls to strtok()
    splits str into tokens, which are sequences of contiguous characters
    separated by any of characters that belong to specified list of delimiters
  - cplusplus.com/reference/cstring/strtok/
  - char * strtok (char * str, const char * delimiters)
  - On a first call, function expects a char array as argument for str,
    whose first character is used as starting location to scan for tokens
  - In subsequent calls, function expects a null pointer (for str) and
    uses position right after end of last token as new starting location for scanning
  - After each call,
    end of found token is automatically replaced by a null-character,
    and beginning of found token is returned by function

Arrays

- Character-array operations in <cstring>




```
String: See,the quick;red:fox
Tokens:
See
the
quick
red
fox
```

```
char str[] = "See,the quick;red:fox";
cout << "String: " << str << endl << "Tokens: " << endl;
char *  t = strtok (str, ",") ; cout << t << endl;
        t = strtok (NULL, " "); cout << t << endl;
        t = strtok (NULL, ";"); cout << t << endl;
        t = strtok (NULL, ":"); cout << t << endl;
        t = strtok (NULL, " "); cout << t << endl;
        t = strtok (NULL, " "); if (t != NULL) cout << "NOT NULL";
```

```
Splitting string "--- CS101 IITB: string theory ---" into tokens:
CS101
IITB:
string
theory
```

- Character-array operations in <cstring>
  - String tokenize

```
char str[] ="--- CS101 IITB: string theory ---";
std::cout << "Splitting string \""
          << str
          << "\" into tokens:"
          << std::endl;
char * ptrChar = strtok (str," ,.-");
while (ptrChar != NULL)
{
  std::cout << ptrChar << std::endl;
  ptrChar = strtok (NULL, " ,.-");
}
```

Practice Examples for Lab: Set 12

- 1

Write a program that reads in an integer from the keyboard and prints it out in words. For example, on reading in 368, the program should print "three hundred and sixty eight".

- 2

For this exercise it is important to know that the codes for the digits are consecutive, starting at 0. Further '8' - '0' is valid expression and evaluates to the difference in the code used to represent the characters, and is thus 8. To clarify, if we execute

```
char text[10] = "1729";
int d = text[1] - '0';
```

Then `d` will have the value 7. Use this to write a function that takes a char array containing a number and return an integer of the corresponding magnitude.

Practice Examples for Lab: Set 12

- 3

Suppose `destination` and `source` are of type `char*`. What do you think the following statement does?

```
while(*destination++ = *source++);
```

Note: it uses several programming idioms you have been warned not to use. The point of this exercise is not to encourage the use of these idioms but to warn you how dense C++ code can be.

- 4

Write a "calculator" program that takes 3 command line arguments in addition to the name of the executable: the first and third being `double` values and the second being a single `char`. The second argument must be specified as an arithmetic operator, i.e. `+`, `-`, `*` or `/`. The program must perform the required operation on the two numbers and print the result.

Practice Examples for Lab: Set 12

- 5

Write and test a function that returns the *plural* form of the singular English word that is passed to it.

- 6

Write and test a function to reverse a C-string in place, without any duplication of characters.

- 7

Write and test the `strrchr()` function.
Write and test the `strstr()` function.
Write and test the `strncpy()` function.
Write and test the `strcat()` function.
Write and test the `strcmp()` function.
Write and test the `strncmp()` function.
Write and test the `strspn()` function.
Write and test the `strcspn()` function.
Write and test the `strpbrk()` function.

Arrays

INTERNAL - II

Computer programming

1. Define Array.

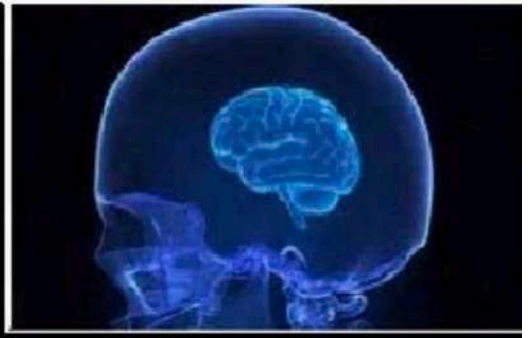
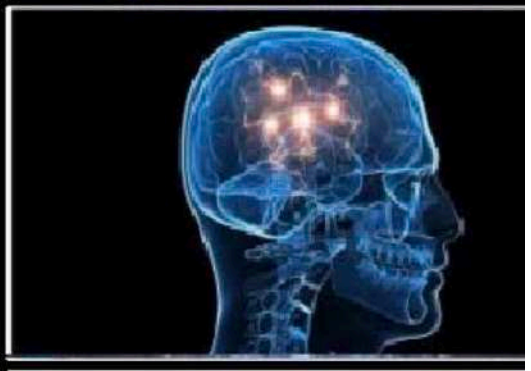
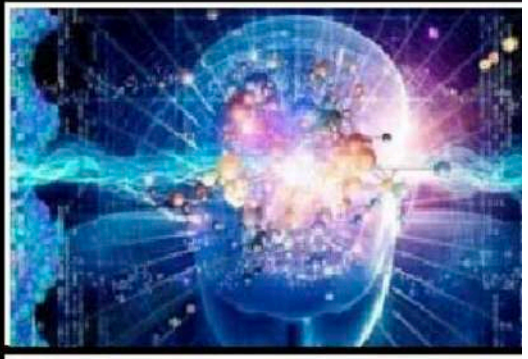
Ans: * Array:

An Array is used to call a boy or a person who is at a distance far away from us who are visible to our naked eye.

Example: Array Rupesh.....

Meet me

Arrays

C, C#, Java, JS, C++, Python, Kotlin	Arrays start at 0	
Lua, MATLAB, Scratch	Arrays start at 1	
Perl	Arrays can start wherever ¯_(ツ)_/¯	
STAR WARS	Arrays start at 4, stop at 6, restart at 1, stop again at 3, restart at 7 then continue on	

Arrays

Arrays

- 2-dimensional arrays: useful for storing matrices
- Example: "double a[3][2];" defines a 2D array of 3*2 double variables
  - Each variable is accessed as a[i][j] where i=0,1,2 and j=0,1
  - Variables are stored in memory in "row-major" order, i.e., row by row
    - In some languages, they would be stored column by column
    - Number of rows = 3
    - Number of columns = 2

Arrays

- Matrix multiplication
  - If A is a m×n matrix and B is a n×p matrix, then C=A*B is a m×p matrix

```
double a[3][2]={{1,2},{3,4},{5,6}}, b[2][4]={{1,2,3,4},{5,6,7,8}}, c[3][4];

for(int i=0; i<3; i++)                          // compute c = a * b.
   for(int j=0; j<4; j++){
      c[i][j] = 0;
      for(int k=0; k<2; k++)
         c[i][j] += a[i][k]*b[k][j];
   }

for(int i=0; i<3; i++){                         // print out c.
   for(int j=0; j<4; j++) cout << c[i][j] << " ";
   cout << endl;
}
```

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Arrays

$$\begin{aligned} 3x_2 + 5x_3 &= 10 \\ 2x_1 + 6x_2 + 8x_3 &= 38 \\ 7x_1 + 4x_2 + 9x_3 &= 22 \end{aligned}$$

$$\begin{bmatrix} 0 & 3 & 5 \\ 2 & 6 & 8 \\ 7 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 38 \\ 22 \end{bmatrix}$$

- Linear simultaneous equations that can be represented in matrix form Ax=b where A is square
- Strategy to solve Ax=b
  - Make modifications to A and b such that:
    - Each modification leads to new equations with same solution as the first
    - Sequence of modifications changes A into Identity matrix, and b into b', leading to trivial solution x = b'
  - What kind of modifications ?
    1. Scaling LHS and RHS of an equation
       - Multiply a row of LHS matrix, and corresponding element on RHS vector, by (non-zero finite) scalar
    2. Adding one equation to another, and replacing any of those by the new equation
       - Add rows of LHS matrix, and replace a row by resulting sum
       - Add elements of RHS vector, and replace an element by resulting sum

Arrays

- Linear simultaneous equations
  1. From equations i=1 to last, find one with coefficient of x_1 as non-zero; Swap 1st equation with this new-found equation
     - If such equation doesn't exist, then report failure (solution doesn't exist or multiple solutions exist)
  2. Divide 1st equation to make coeff of x_1 as 1
  3. Add suitably-scaled 1st equation to all other equations to make their coefficients for x_1 as 0
     - Now 1st column same as that of identity matrix
- Repeat steps 1-2-3 with respect to variable x_i and i-th equation for i←i+1

$$\begin{bmatrix} 0 & 3 & 5 \\ 2 & 6 & 8 \\ 7 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 38 \\ 22 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 6 & 8 \\ 0 & 3 & 5 \\ 7 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 38 \\ 10 \\ 22 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 4 \\ 0 & 3 & 5 \\ 7 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ 10 \\ 22 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 4 \\ 0 & 3 & 5 \\ 0 & -17 & -19 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ 10 \\ -111 \end{bmatrix}$$

Arrays

- Linear simultaneous equations
  - en.wikipedia.org/wiki/Gaussian_elimination
  - Instead of making A as identity, alternate strategy makes A as upper-triangular

System of equations	Row operations	Augmented matrix
$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$		$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ 2y + z &= 5 \end{aligned}$	$\begin{aligned} L_2 + \tfrac{3}{2}L_1 &\to L_2 \\ L_3 + L_1 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ -z &= 1 \end{aligned}$	$L_3 + -4L_2 \to L_3$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$

Arrays

$$\begin{aligned} 2x + \ \ y - \ \ z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ -z &= 1 \end{aligned} \qquad \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

- Linear simultaneous equations
  - en.wikipedia.org/wiki/Gaussian_elimination
  - Given triangular matrix/form, use "back-substitution" to solve equations, or take few more steps to convert lower-triangular matrix to identity matrix

$\begin{aligned} 2x + \ \ y &= 7 \\ \tfrac{1}{2}y &= \tfrac{3}{2} \\ -z &= 1 \end{aligned}$	$\begin{aligned} L_2 + \tfrac{1}{2}L_3 &\to L_2 \\ L_1 - L_3 &\to L_1 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ 0 & 0 & -1 & 1 \end{array}\right]$
$\begin{aligned} 2x + y &= 7 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} 2L_2 &\to L_2 \\ -L_3 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$
$\begin{aligned} x &= 2 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} L_1 - L_2 &\to L_1 \\ \tfrac{1}{2}L_1 &\to L_1 \end{aligned}$	$\left[\begin{array}{ccc\|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$

Arrays

- **Two-dimensional array of characters** = a 1D array of strings

```
char countries[6][20] = {"India","China","Sri Lanka","Nepal",
                         "Bangladesh","Pakistan"};
```

  - “countries[0]” refers to first string “India”
    - Pointer referring to address of first character of first string

Arrays

- **2D arrays as arguments to functions**
  - In called-function's definition,
    second dimension of array must be a compile-time constant

```
void print(char countries[][20], int noOfCountries){
  for(int i=0; i<noOfCountries; i++) cout << countries[i] << endl;
}
```

  - This format limits generalizability,
    because it can only be used for arrays of strings of size 20

- An array is sequence of variables of the same type stored in memory
- So, we can have arrays of any data type, including pointers
  - e.g., "int * a[10];" defines an array, of size 10, of pointers to integers

Arrays

- Arguments to main program
  - C++ allows you to create a program that can be invoked, from the operating system's command line, using arguments typed on the command line
    - e.g., ./a.out cs101 cs102
  - To receive and use these arguments in our program, we need to define our "main" function as follows (you can have your own argument names):

```
int main(int argc, char *argv[]);
```

  - argv (argument vector) is an array of pointers to char* (equivalently, strings)
    - Contains strings typed on command prompt, i.e., "./a.out", "cs101", "cs102"
  - argc (argument count) = number of strings typed on command line
    - Automatically set and passed by the operating system to ./a.out
  - The program can use these arguments, do the data processing, and return an int code back to the operating system's command line

Arrays

```
int main(int argc, char *argv[]){
  for(int i=0; i<argc; i++) cout << argv[i] << endl;
}
```

- This program, when invoked as "a.out Mathematics Biology" would print out →

```
a.out
Mathematics
Biology
```

- Utility
  - Replace input within program to input on command-line itself
  - e.g., imagine a program to draw regular polygons where number of sides and side length are command-line parameters
  - e.g., when we invoke "s++ -c filename", then s++ program receives argc=3 with the 2 arguments as argv[1] and argv[2]

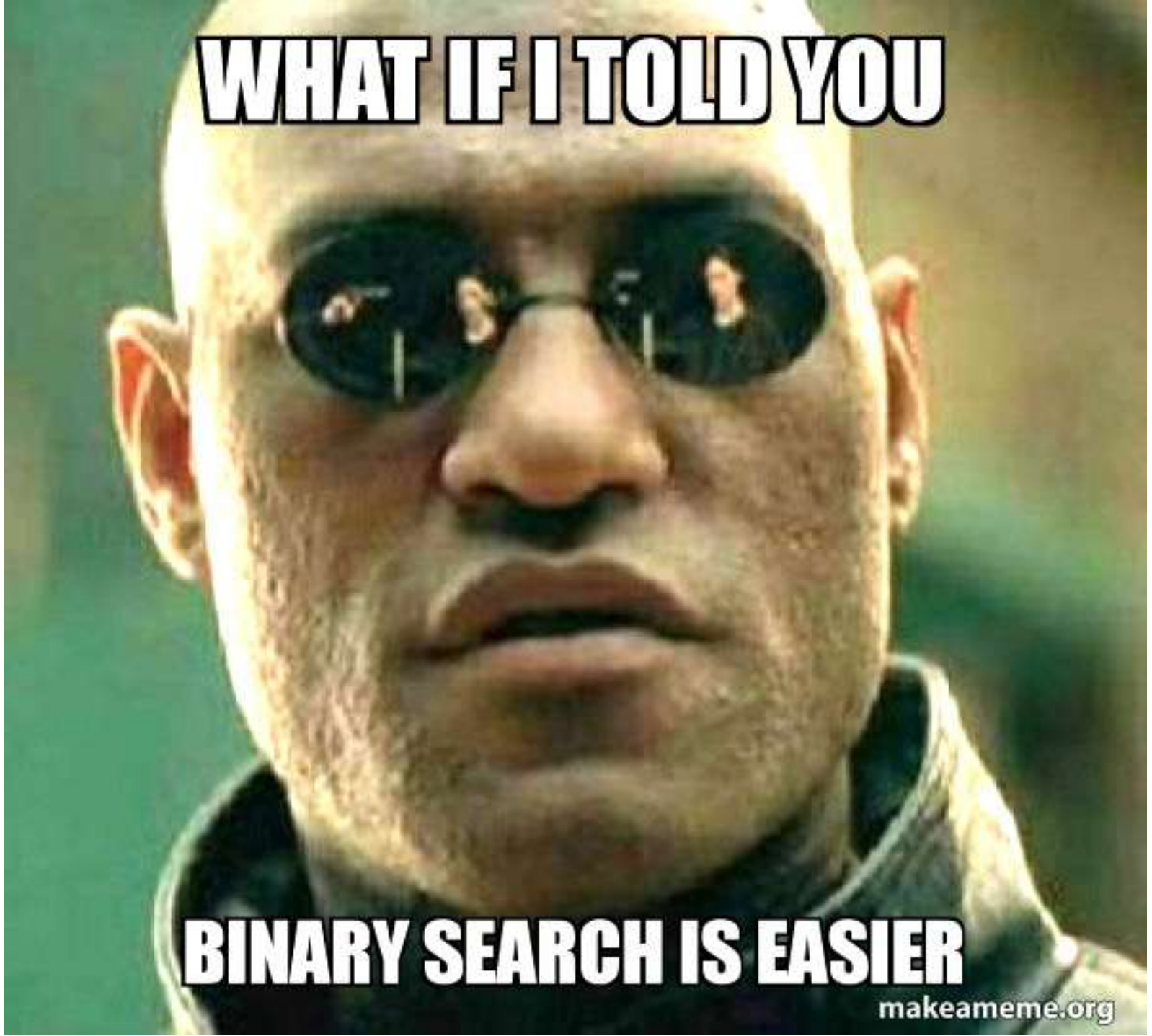
Arrays

- Arguments to main program
  - All arguments are received as strings

`atof()`	`double atof(const char* s);` Returns the number represented literally in the string `s`.	`<cstdlib>`
`atoi()`	`int atoi(const char* s);` Returns the integer represented literally in the string `s`.	`<cstdlib>`
`atol()`	`long atol(const char* s);` Returns the integer represented literally in the string `s`.	`<cstdlib>`

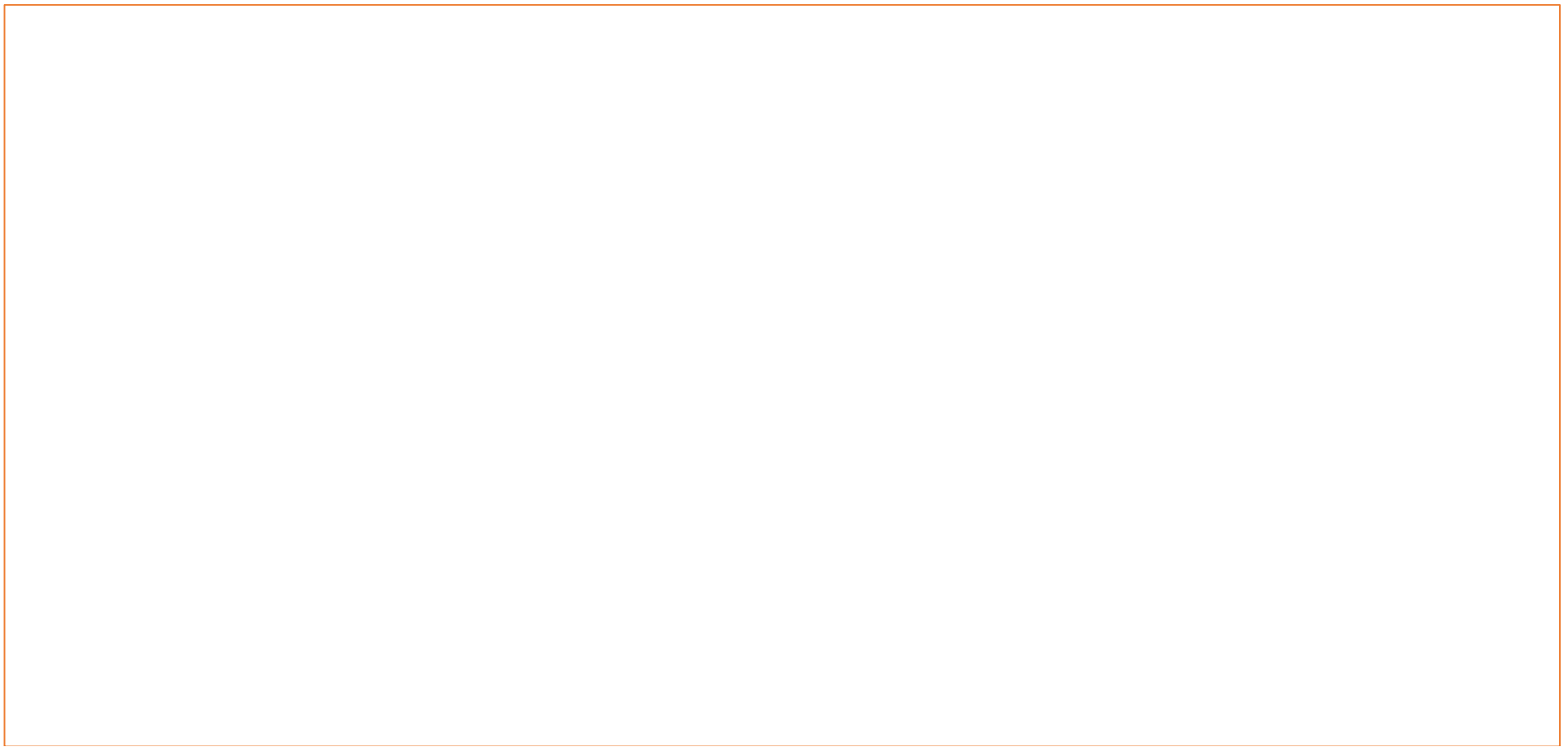
Arrays

- Searching through a list
  - In general, we have to check every element in the list



Arrays

- Searching through a list
  - Searching can be done much faster on long lists when the lists are sorted, as compared to when they aren't sorted
  - "Binary Search" algorithm on a sorted list (stored as an array)



Arrays

- Binary Search (version 1)
  - Algorithm
    - Compare element in “middle” of array with targetValue
    - If element = targetValue, then return its position in array
    - If element > targetValue, then continue search in lower half of array
    - If element < targetValue, then continue search in upper half of array

```
function binary_search(A, n, T) is
    L := 0
    R := n - 1
    while L ⩽ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```

1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- Binary Search (recursive): call from main: 7, A, start = 0, size = 17
  - half = 8, A[start+half] = A[8] = 14 → call Bsearch (7, A, start = 0, size = 8)
  - half = 4, A[start+half] = A[4] = 7 → call Bsearch (7, A, start = 4, size = 4)
  - half = 2, A[start+half] = A[6] = 10 → call Bsearch (7, A, start = 4, size = 2)
  - half = 1, A[start+half] = A[5] = 8 → call Bsearch (7, A, start = 4, size = 1)
  - return (A[4] == 7)

```
bool Bsearch(int x, int A[], int start, int size){
// x : target value to search
// range to search: A[start..start+size-1]
// precondition: size > 0;
//
  if(size == 1) return (A[start] == x);
  int half = size/2;                    // 0 < half < size, because size>1.
  if(x < A[start+half])
    return Bsearch(x, A, start, half);          // recurse on first half
  else
    return Bsearch(x, A, start+half, size-half); // recurse on second half
}
```

Arrays

- Binary search: estimating computational load
  - Consider array of size 1024
  - If array isn't sorted,
    search needs 1024 comparisons
  - If array is sorted,
    we need 10 (= $\log_2 1024$) comparisons
    - Each call (if made) reduces sub-array size by factor of 2
    - Only in the last call will size==1 and we can return the answer

Arrays

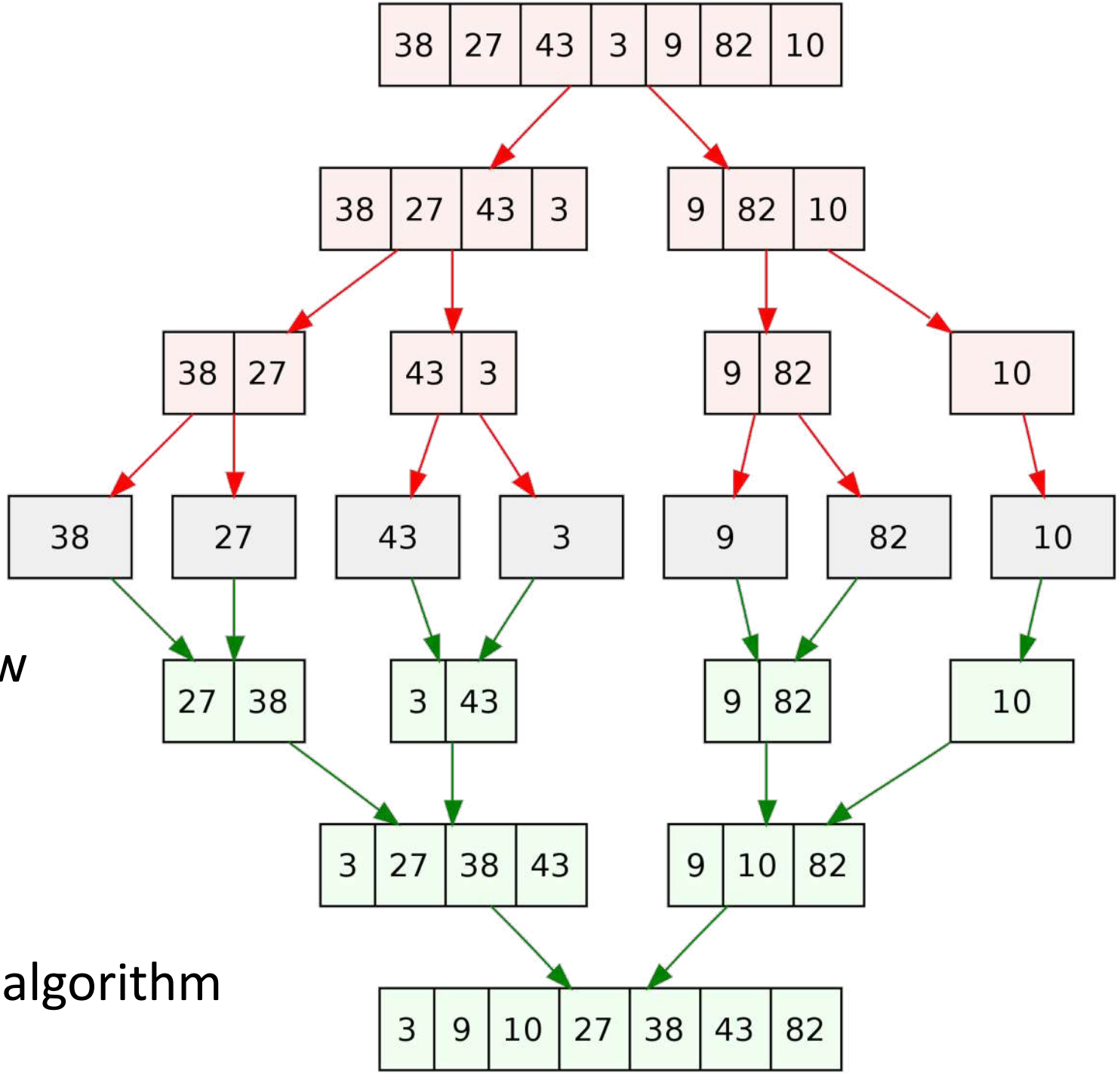
- Can we sort much faster than selection sort ?
  - For large sized arrays
- Merge Sort

Arrays

- Merge Sort insights
  - Merging 2 sorted lists of sizes m and n takes proportional to m+n operations

1. Divide input array into 2 sub-arrays of size ~half
2. Sort each subarray somehow
   - Recursive call to mergesort !
   - Base case: list size = 1, already sorted, do nothing
3. Merge the sub-arrays

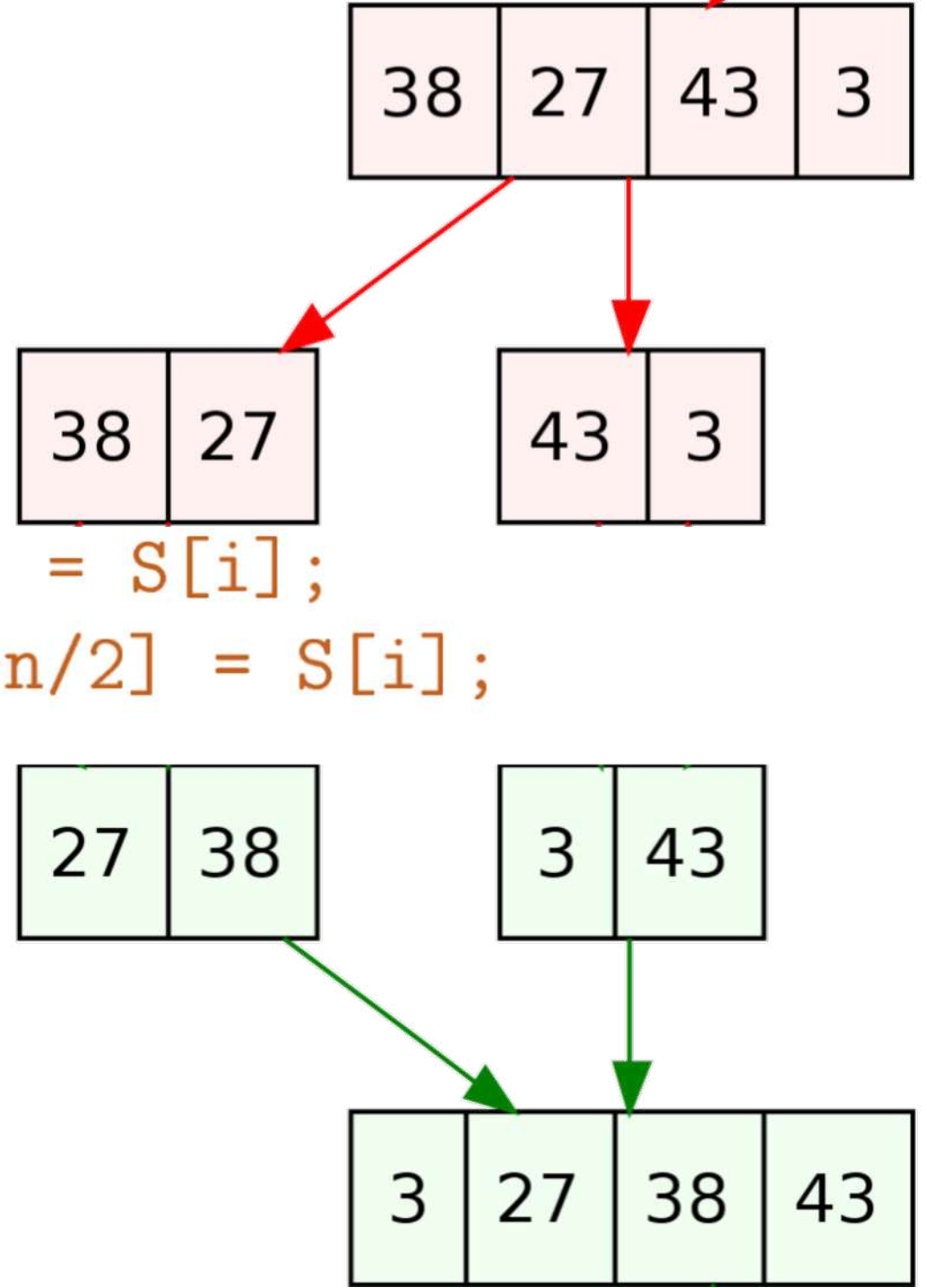
- "Divide and conquer" type of algorithm



Arrays

- Merge Sort

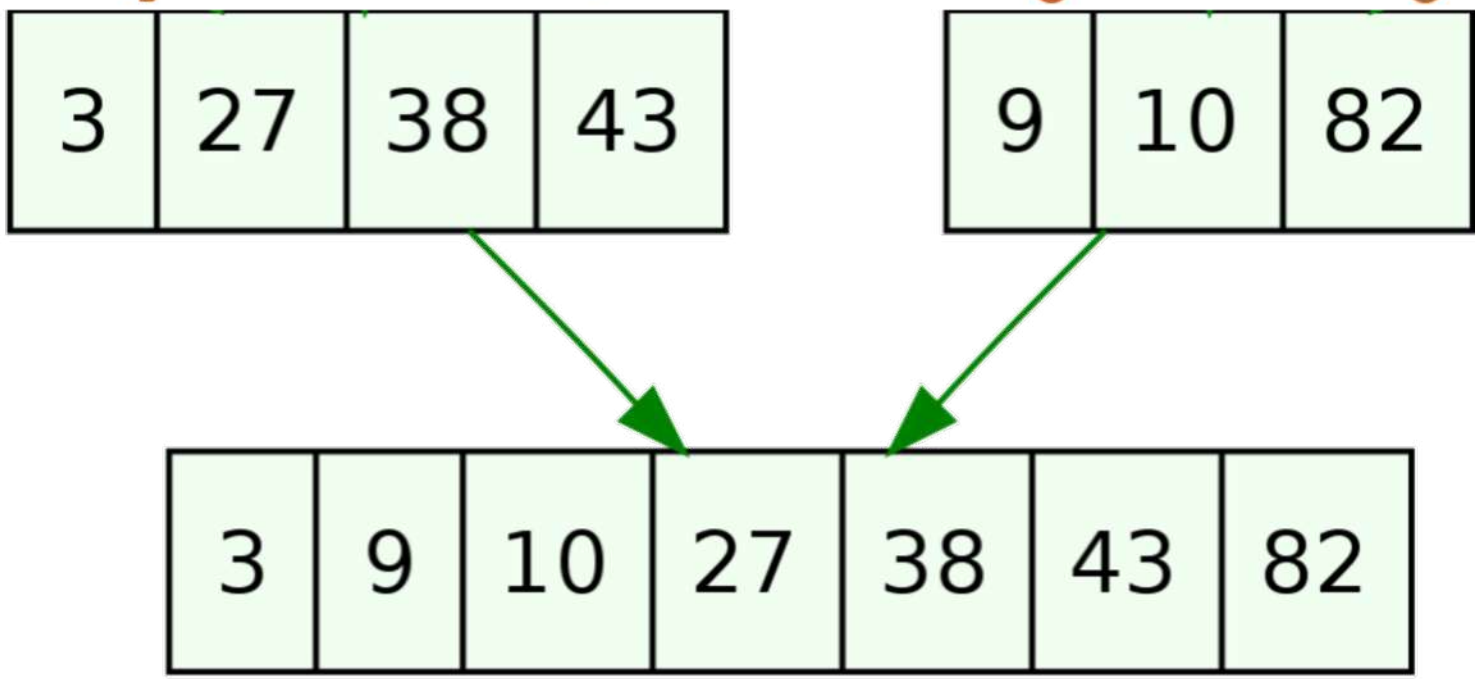
```
void mergesort(int S[], int n){
   if(n>1){
      int U[n/2], V[n-n/2];
      for(int i=0; i<n/2; i++) U[i] = S[i];
      for(int i=n/2; i<n; i++) V[i-n/2] = S[i];


      mergesort(U, n/2);
      mergesort(V, n - n/2);
      merge(U, n/2, V, n-n/2, S);
   }
}
```



Arrays

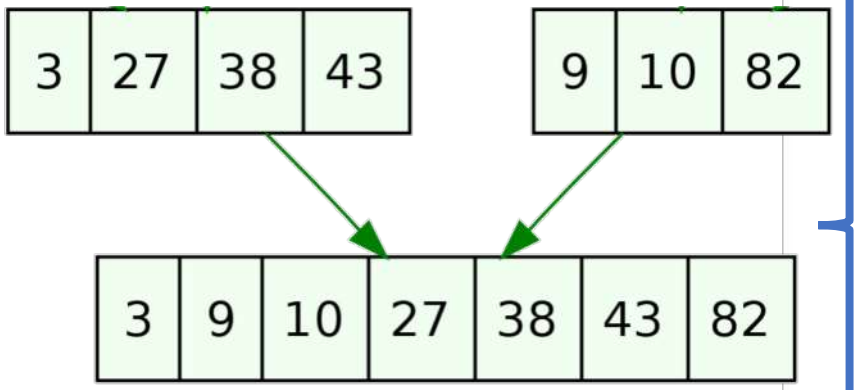
- Merge Sort

```
void merge(int U[], int uLength, int V[], int vLength, int S[]){
// arrays U,V of length uLength and vLength respectively contain the
// sequences that are sorted.  The result of merging is to be placed
// in the array S.  The length of S is not specified explicitly, but
// it is assumed (precondition) to be uLength + vLength.
```



Arrays

- Merge Sort
  - How to merge ?



- uF 0, vF 0, sB 0
- uF 1, vF 0, sB 1
- uF 1, vF 1, sB 2
- uF 1, vF 2, sB 3
- uF 2, vF 2, sB 4
- uF 3, vF 2, sB 5
- uF 4, vF 2, sB 6
- uF 4, vF 3, sB 7

```
for(int uFront=0, vFront=0, sBack=0; sBack<uLength+vLength; sBack++){
// INVARIANT: sBack = uFront + vFront.  Keys U[0..uFront-1] will
// have been moved to S, and and also keys V[0..vFront-1].  S will
// contain these keys in S[0..sBack-1], in non-decreasing order.

  if(uFront<uLength && vFront<vLength){  // if both queues non-empty
    if(U[uFront] < V[vFront]){          // if U has smaller
      S[sBack] = U[uFront];             // move to S
      uFront++;                         // advance U
    }
    else{                               // if V has smaller
      S[sBack] = V[vFront];             // move to S
      vFront++;                         // advance V
    }
  }
  else if(uFront < uLength) {           // else if only U is not empty
    S[sBack] = U[uFront];               // move to S
    uFront++;                           // advance U
  }
  else {                                // else if only V is not empty
    S[sBack] = V[vFront];               // move to S
    vFront++;                           // advance V
  }
  }
}
```

Arrays

```
mergesort(U, n/2);
mergesort(V, n - n/2);
merge(U, n/2, V, n-n/2, S);
```

- Merge Sort: computational cost
  - Let T(n) be number of operations to sort list of n elements
  - Merging two sorted lists of size a and b takes number of operations proportional to a+b
    - In our case, a+b = n
  - So, recursively analyzing: $T(n) \leq (\text{Time proportional to } n) + T(n/2) + T(n - n/2)$
  - Assuming, for simplicity, n is a power of 2 ($n = 2^k$): $T(n) \leq cn + 2T(n/2)$
  - But, doing analysis for 2 sub-arrays of size n/2: $T(n/2) \leq c(n/2) + 2T(n/4)$
  - So, $T(n) \leq cn + 2T(n/2) \leq cn + 2(c(n/2) + 2T(n/4) = 2cn + 4T(n/4)$
  - But, T(n/4) = c(n/4) + 2T(n/8)
  - So, T(n) = 2cn + 4c(n/4) + 8T(n/8) = 3cn + 8T(n/8)
  - Continuing for k steps, we get: $T(n) \leq kcn + 2^k T(n/2^k)$
  - When k = $\log_2 n$ steps, we get: $T(n) \leq cn \log_2 n + nT(1)$

mathew ✅

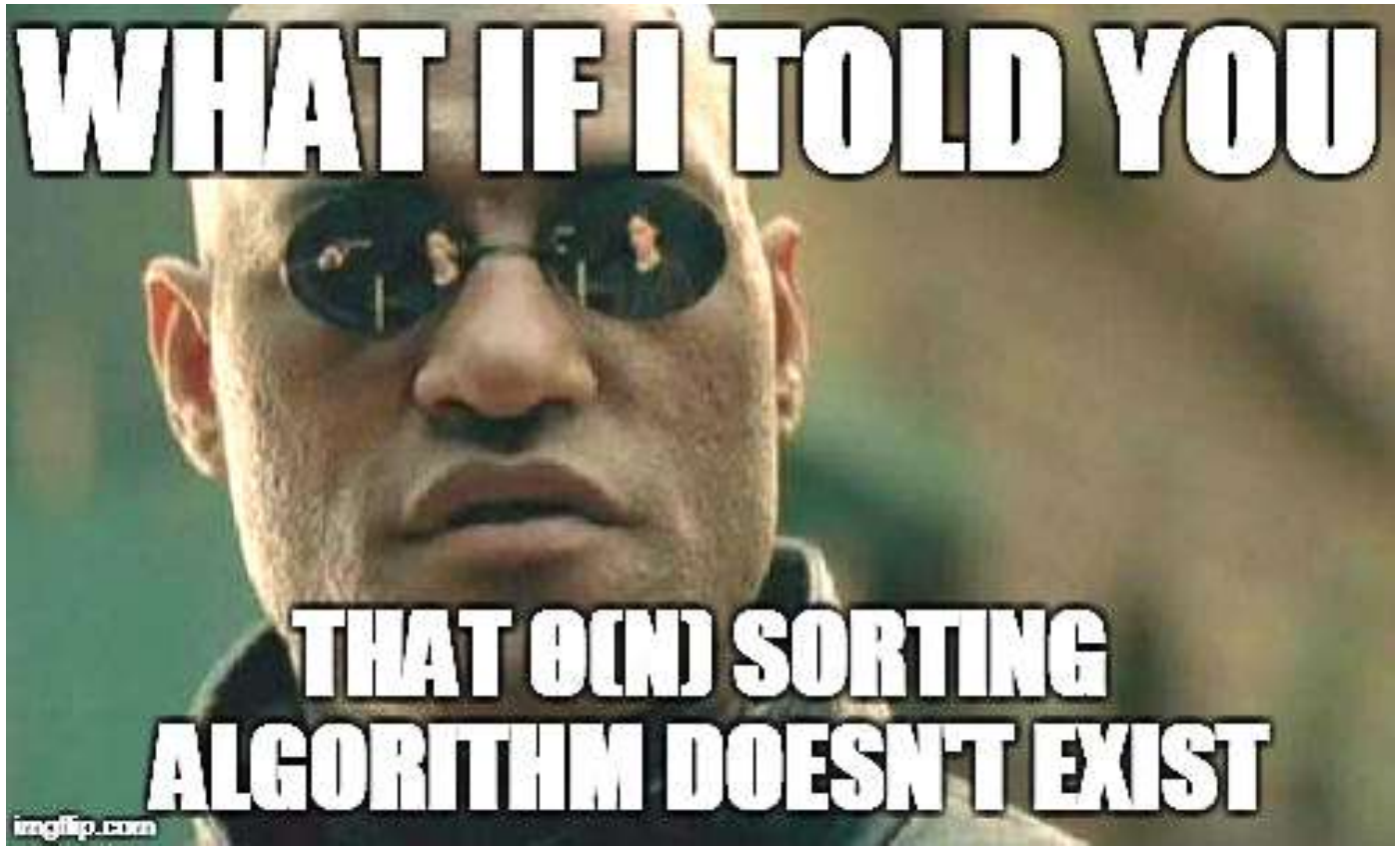
@mathew@mastodon.social

I came up with a single pass O(n) sort algorithm I call StalinSort. You iterate down the list of elements checking if they're in order. Any element which is out of order is eliminated. At the end you have a sorted list.

2018/10/26 04:20:16

Arrays

- “Comparison sort”
  - en.wikipedia.org/wiki/Comparison_sort



- “Non-comparison sorting”: a different story
  - en.wikipedia.org/wiki/Sorting_algorithm#Non-comparison_s

- Quicksort
- Heapsort
- Shellsort
- Merge sort
- Introsort
- Insertion sort
- Selection sort
- Bubble sort
- Odd–even sort
- Cocktail shaker sort
- Cycle sort
- Merge-insertion sort
- Smoothsort
- Timsort
- Block sort

Practice Examples for Lab: Set 13

- 1

Suppose you are given a sequence of numbers, preceded by the length of the sequence. You are required to sort them. In this exercise you will do this using the so called *Insertion sort* algorithm. The idea of the algorithm is to read the numbers into an array, but keep the array sorted as you read. In other words, after you read the first i numbers, you must make sure that they appear in the first i elements of the array in sorted (say non-increasing) order. So when you read the $i+1$th number, you must find where it should be inserted. Suppose you discover that it needs to be placed between the numbers that are currently at the jth and $j+1$th position, then you should move the numbers in positions $j+1$ through $i-1$ (note that the indices or positions start at 0) forward in the array by 1 step. Then the newly read number can be placed in the $j+1$th position. Write the program that does this.

- 2

Write a program that reads a sequence of names, one per line, and then sorts and prints them.

Practice Examples for Lab: Set 13

- 3

Write and test the following function that attempts to remove an item from an array:

```
bool removeFirst(float a[], int& n, float x);
```

The function searches the first `n` elements of the array `a` for the item `x`. If `x` is found, its first occurrence is removed, all the elements above that position are shifted down, `n` is decremented, and **`true`** is returned to indicate a successful removal. If `x` is not found, the array is left unchanged and **`false`** is returned.

- 4

Write and test the following function:

```
void rotate(int a[], int n, int k);
```

The function "rotates" the first `n` elements of the array `a`, `k` positions to the right (or –`k` positions to the left if `k` is negative). The last `k` elements are "wrapped" around to the beginning of the array. For example, the call `rotate(a,8,3)` would transform the array {22,33,44,55,66,77,88,99} into {77,88,99,22,33,44,55,66}. The call `rotate(a,8,-5)` would have the same effect.

Practice Examples for Lab: Set 13

- 5

  Rewrite bubble-sort as an indirect sort, i.e., instead of moving the actual elements of the array, sort an "index" array.

- 6

Write and test the following function:

```
void reverse(int a[], int n);
```

The function reverses the first `n` elements of the array. For example, the call `reverse(a,5)` would transform the array {22,33,44,55,66,77,88,99} into {66,55,44,33,22,77,88,99}.

- 7

Write and test a function that implements the *Perfect Shuffle* of a one-dimensional array with an even number of elements. For example, it would replace the array `{11,22,33,44,55,66,77,88}` with the array `{11,55,22,66,33,77,44,88}`.

Find the Bug

- courses.cs.vt.edu/~cs2204/summer2004/readings/bugexamples.htm

TYPE: Accidental

```
for (i=0; i<numrows; i++)
  for (j=0; j<numcols; j++);
    pixels++;
```

TYPE: Missing or improper initialization

```
int minval(int *A, int n) {
  int currmin;

  for (int i=0; i<n; i++)
    if (A[i] < currmin)
      currmin = A[i];
  return currmin;
}
```

TYPE: Accidental

```
if (foo = 5)
  foo == 7;
```

TYPE: Mis-copy bug

```
switch (i) {
  case 1:
    do_something(1); break;
  case 2:
    do_something(2); break;
case 3:
  do_something(1); break;
case 4:
  do_something(4); break;
default:
  break;
```

Find the Bug

TYPE: Dyslexic

204/summer

```
int minval(int *A, int n) {
  int currmin = MAXINT;

  for (int i=0; i<n; i++)
    if (A[i] > currmin)
      currmin = A[i];
  return currmin;
}
```

TYPE: Model error

```
char* string1 = "Hello";
char* string2 = "World";

if (string1 == string2)
  do_something();
```

TYPE: Abused global

```
int i = 5;
int j;

int foo(int j) {
  for (i=0; i<j; i++) do_nothing();
  return j;
}

void ineedj(void) {
  cout << "j is " << j << "\n";
}

main() {
  int j;
  j = foo(i);
  ineedj();
}
```

Find the Bug

- courses.cs.vt.edu/~cs2204/summer2004/readings/bugexamples.htm

TYPE: Parameter mismatch

```
char string1[10] = "Hello";
char string2[10];

strcpy(string1, string2);
```

TYPE: Memory error

```
char* ptr;

cin >> ptr;
```

TYPE: Missing string terminator

```
char string[4];

for (i=0; i<4; i++)
  string[i]=getchar();
cout << string;
```

TYPE: Off-by-one error

```
int i;
int array[5];
int j;

for (i=0; i<=5; i++)
  cin >> array[i];
```

Find the Bug

- courses.cs.vt.edu/~cs2204/summer2004/readings/bugexamples.htm

TYPE: Stack frame problem

```
char *initialize() {
   char string[80];
   char* ptr = string;
   return ptr;
}

main() {
   char *myval = initialize();
   do_something_with(myval);
}
```

TYPE: Stack frame problem

```
char* assign() {
   return "hello world!";
}

main() {
   char *ptr = assign();
}
```

Arrays

6 5 3 1 8 7 2 4

- Bubble sort
  - Make a pass through entire input list element by element
    - Compare current element with next one: If not in order, then swap them
  - Repeat passes until a pass leads to NO swaps

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            { if this pair is out of order }
            if A[i-1] > A[i] then
                { swap them and remember something changed }
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

Arrays

6 5 3 1 8 7 2 4

- Bubble sort
  - Observations (sorting in ascending order)
    - 1st pass places largest element at last place (its final place) [largest element bubbles up]
    - 2nd pass places 2nd-largest element at last-but-1 place (its final place)
    - N-th pass places N-th largest element at its final place
  - Slightly improved algorithm
    - During n-th pass, inner loop avoids looking at last n−1 items

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i])
                swapped := true
            end if
        end for
        n := n - 1
    until not swapped
end procedure
```

Arrays

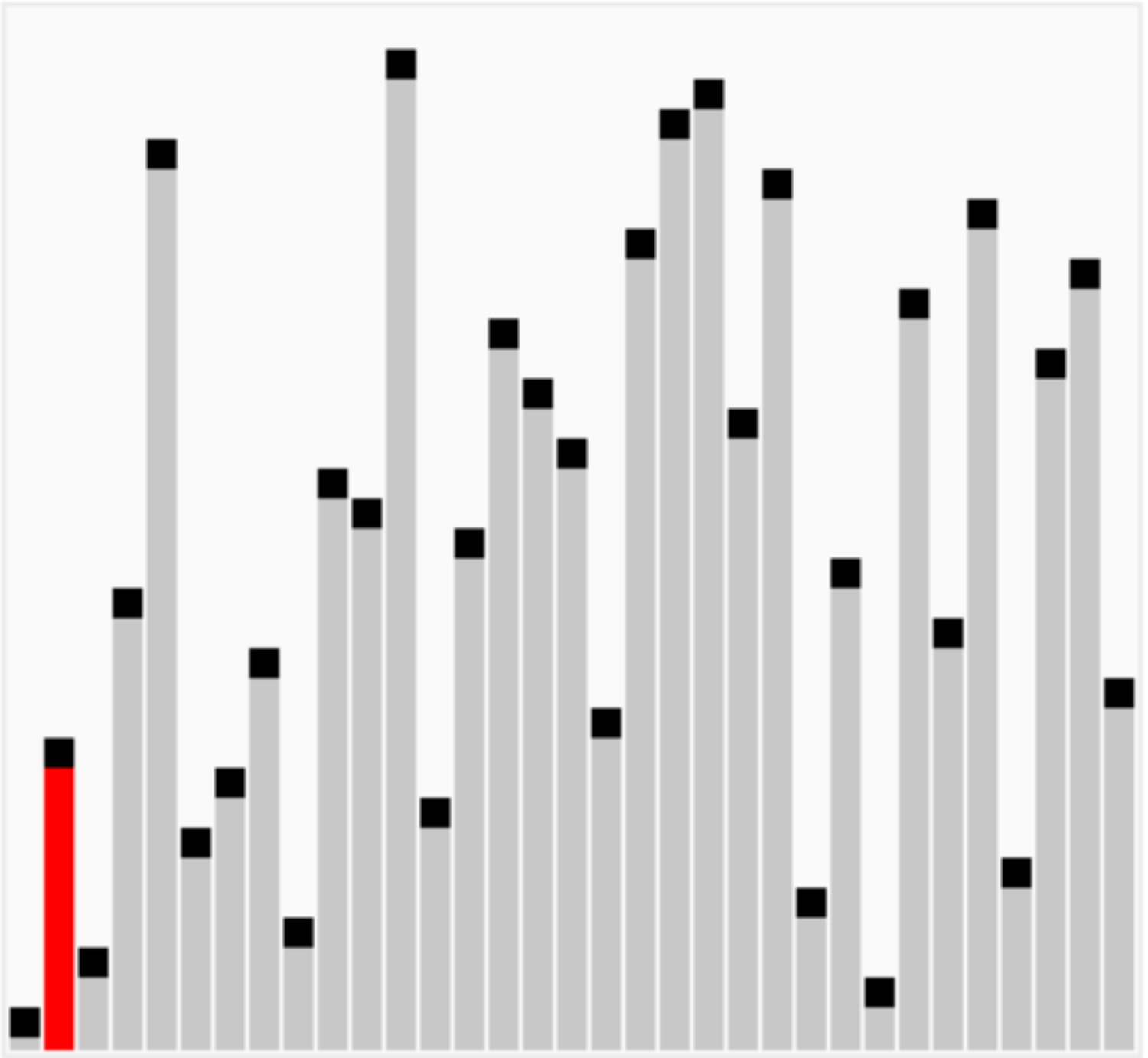
6 5 3 1 8 7 2 4

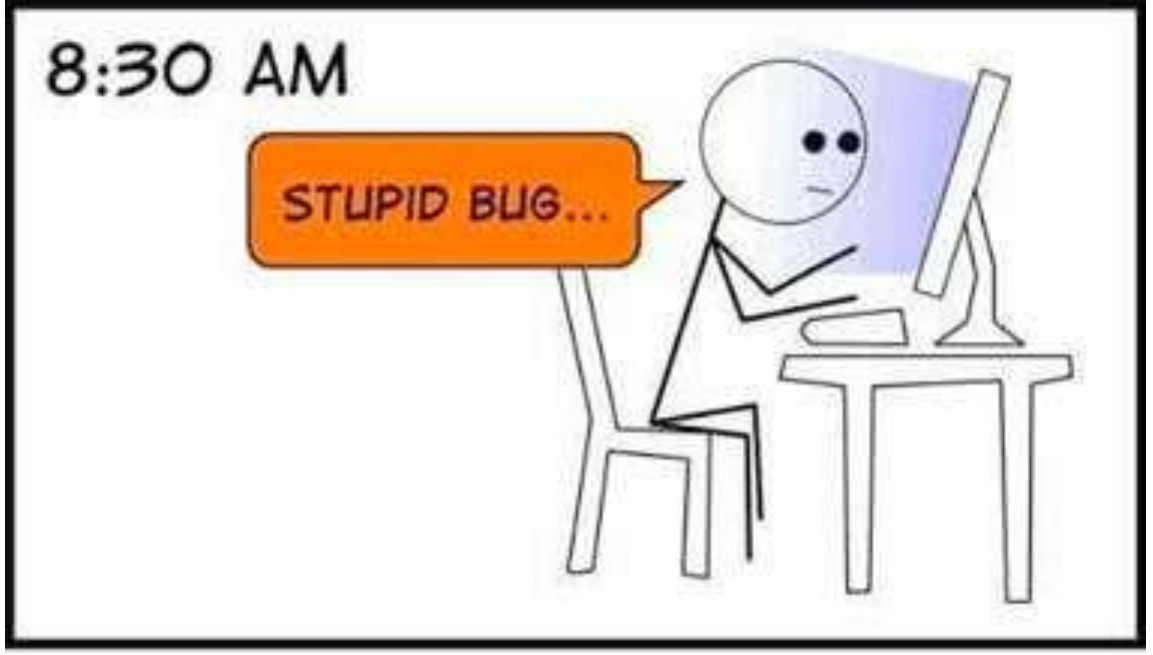
- Bubble sort
  - Observations (sorting in ascending order)
  - After a pass:
    - Possible that multiple elements placed in final position
    - All elements after last swap are sorted !
  - Improved algo.
    - Save position of last swap
    - Shorten size of sub-array

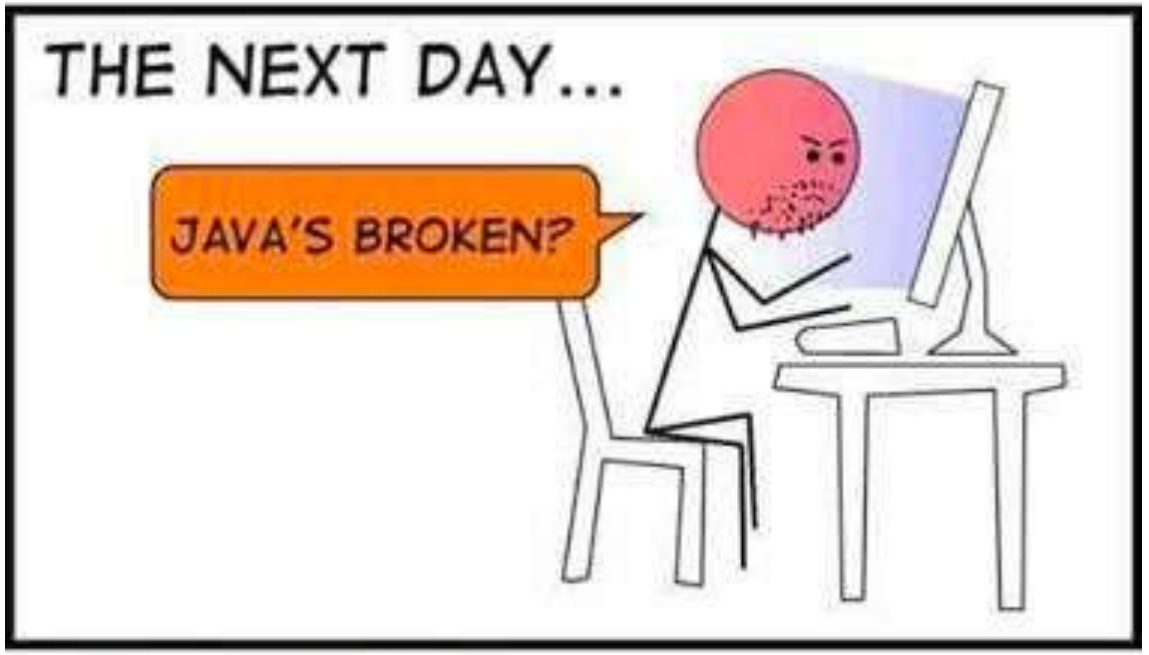
```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        newn := 0
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i])
                newn := i
            end if
        end for
        n := newn
    until n ⩽ 1
end procedure
```

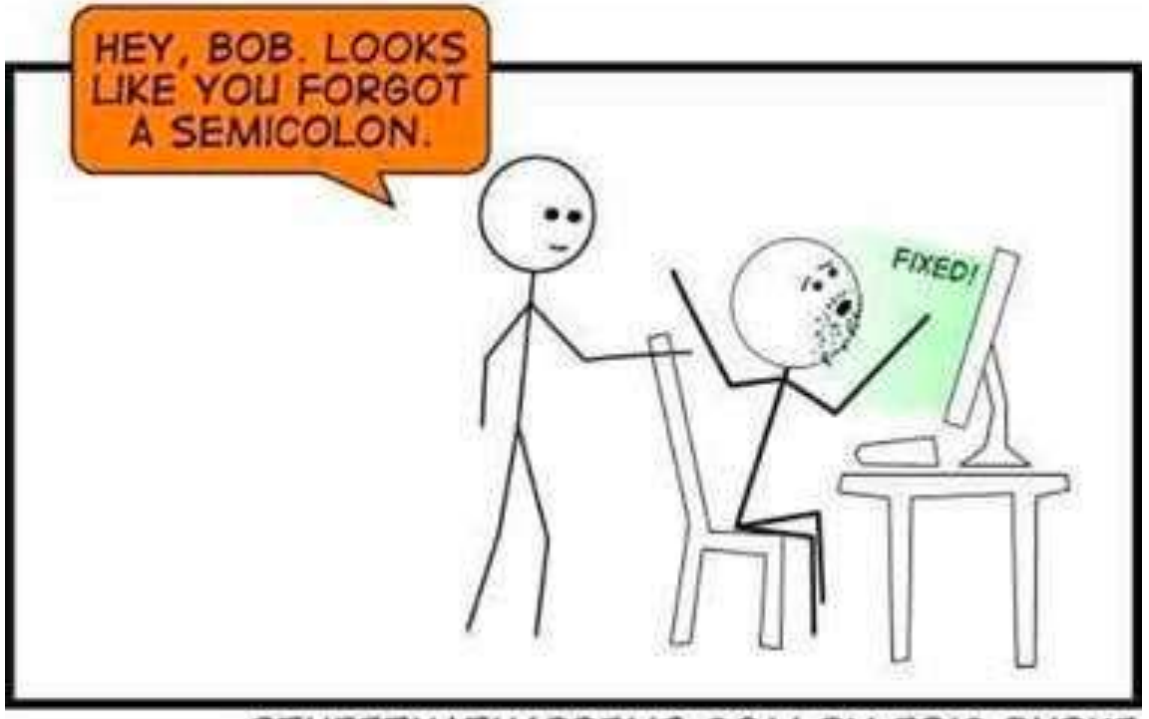
Arrays

- Insights into bubble sort
  - e.g., [2 3 4 5 1] needs 4 passes of "ascending" bubble sort
  - Bubble sort can move elements backwards only one location within a pass
- How can we improve ?
  - Bubble sort alternatingly in BOTH directions (left→right ascending, then right→left descending)
  - Bidirectional Bubble Sort



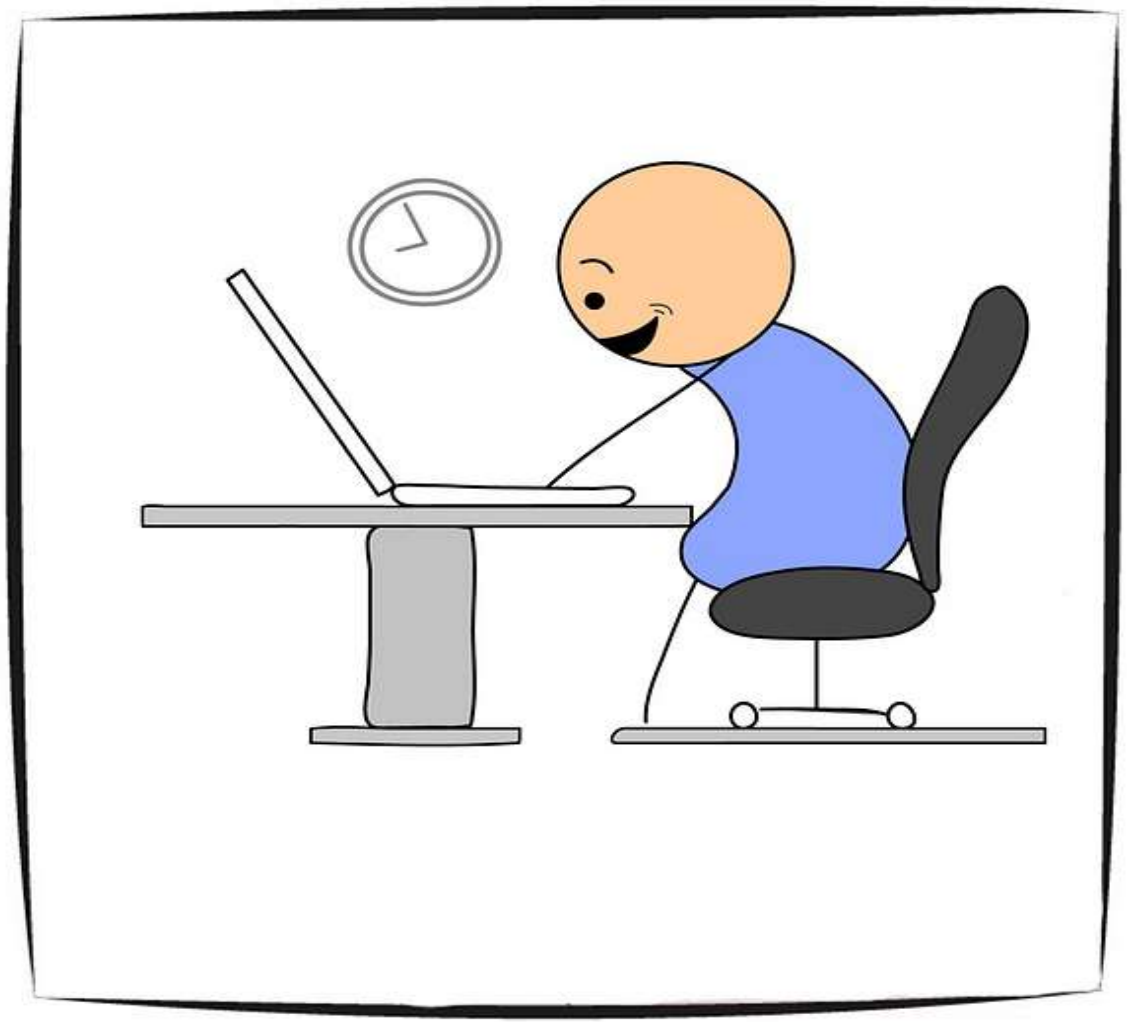


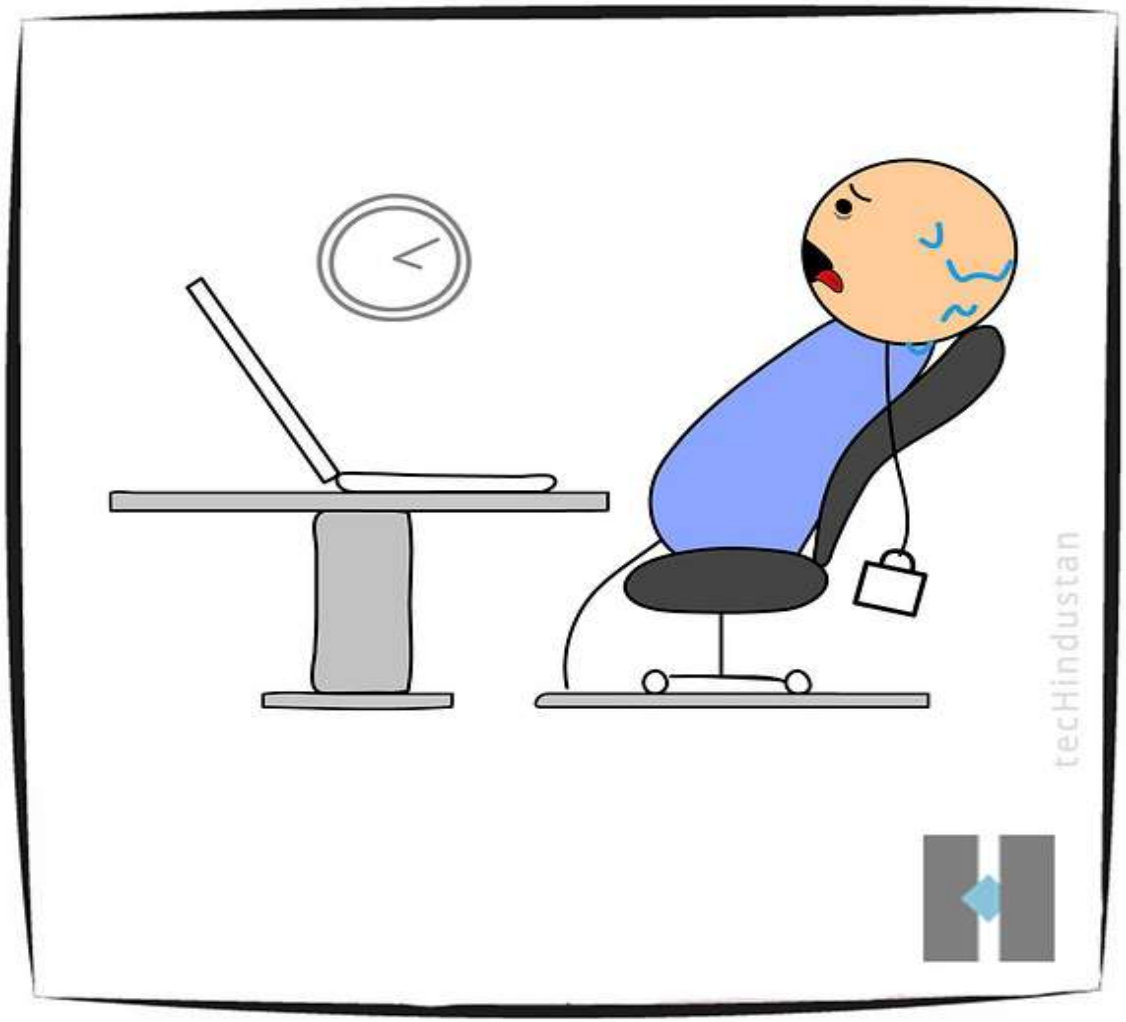




STUFFTHATHAPPENS.COM BY ERIC BURKE

Programming

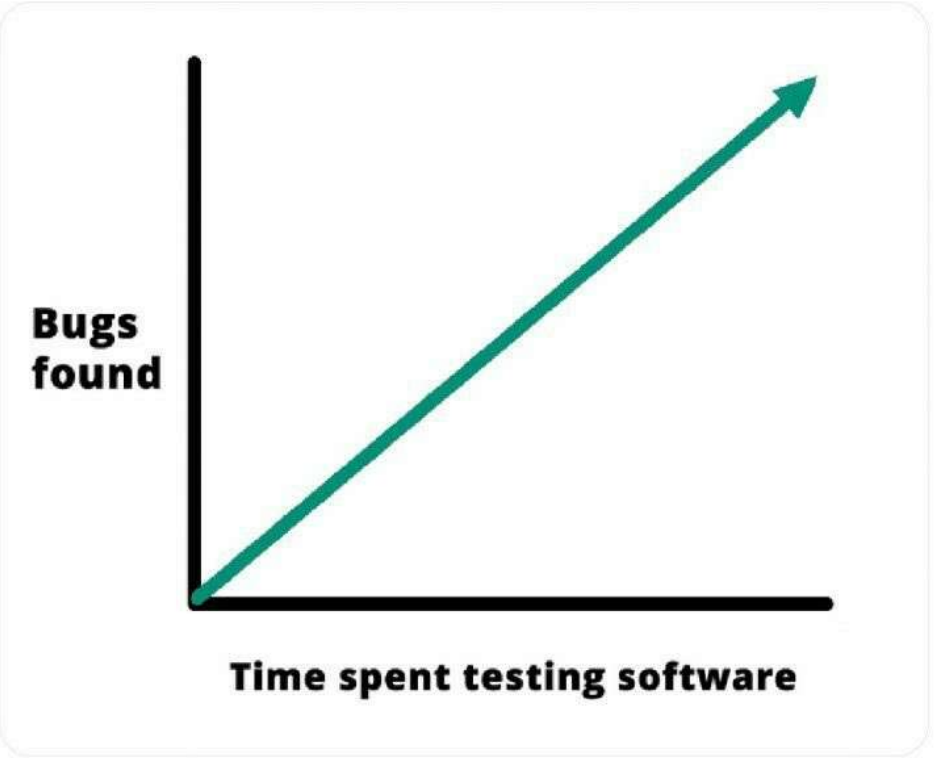




If you're not tired
you're not doing it right

Kat Maddox
@ctrlshifti

Is it a coincidence? I don't think so. Stop testing your software



14:06 · 20 Sep 20 · Twitter Web App

1,717 Retweets **141** Quote Tweets