CS 101
Computer Programming and Utilization

Arrays

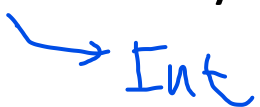
Suyash P. Awate

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

- An "array" is a collection 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
```

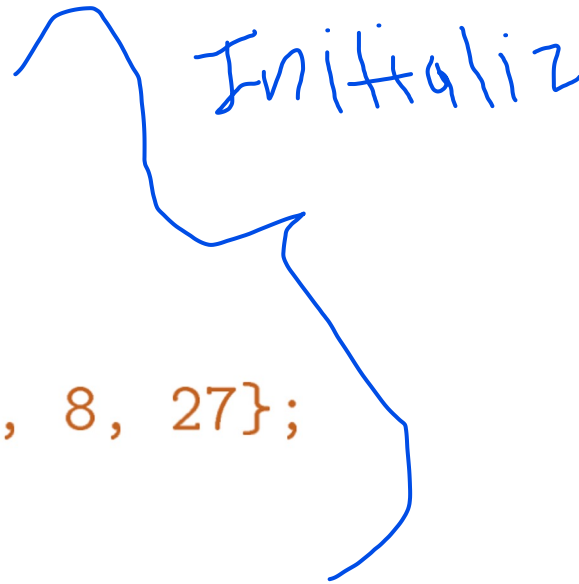
Arrays

- 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

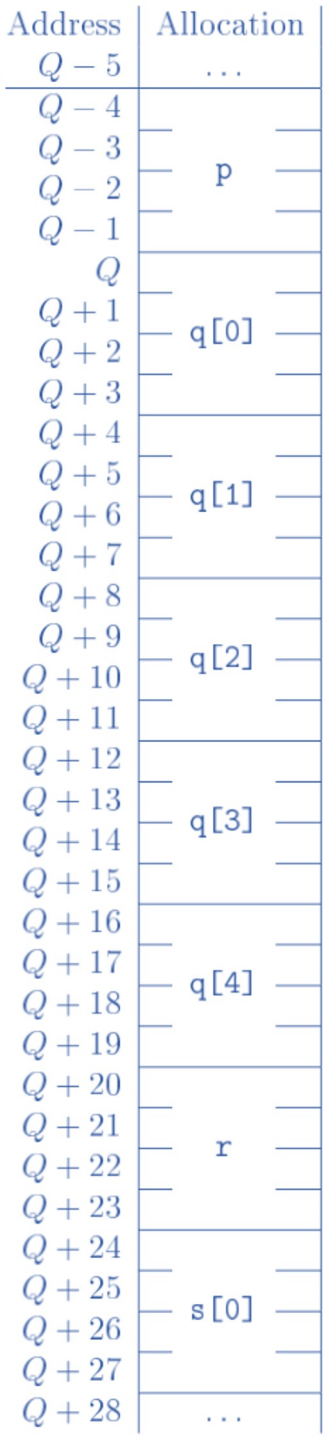
```
for(int i=0; i<10; i++)
    count[i]=0;
```

  - 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.
```

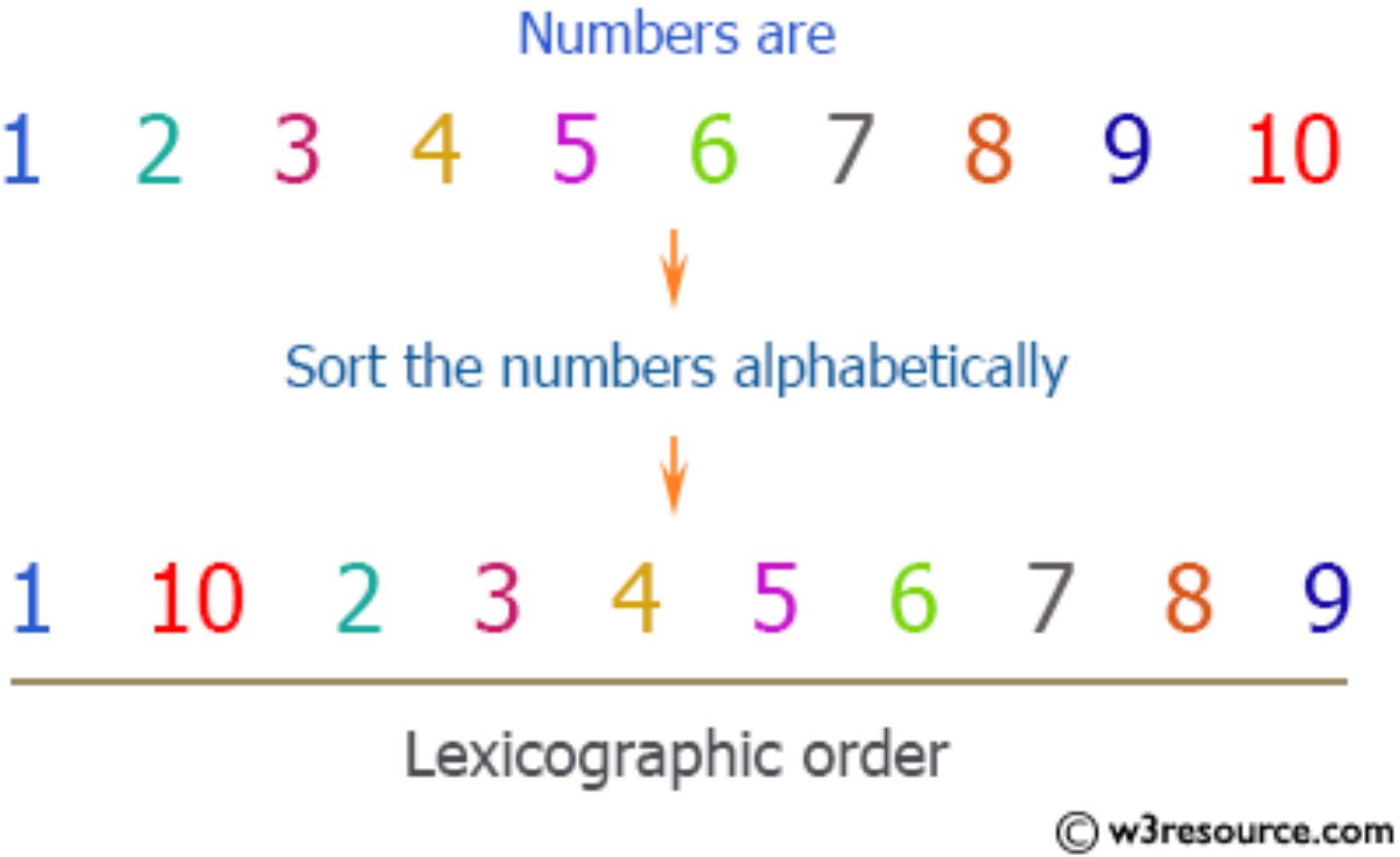
Arrays

- 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

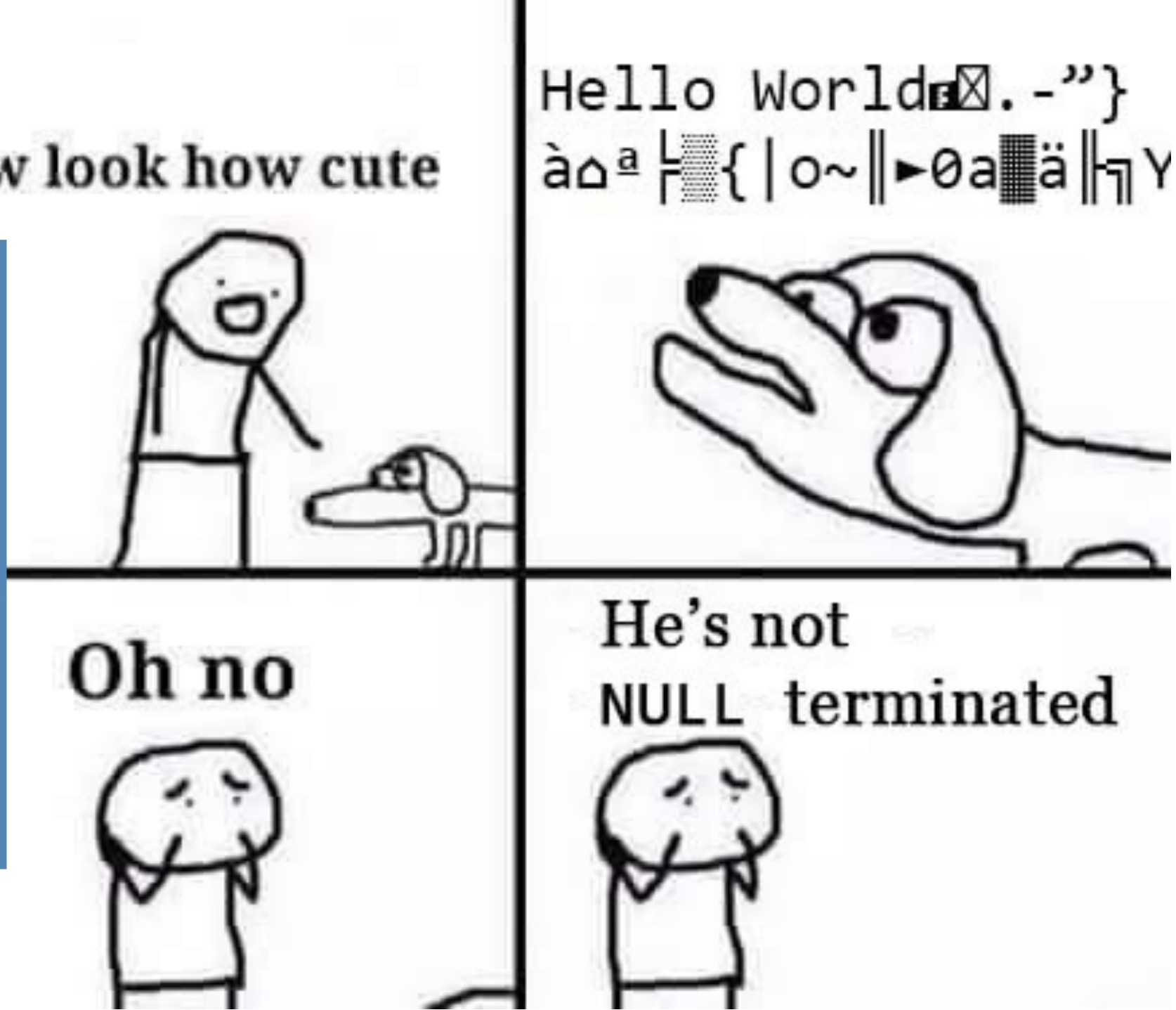
- Character arrays

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