Lecture 6 – Arrays

Based on slides designed by Shay Artsi, Aythan Avior, Gitit Rockstein and Saheir Ismir for the department of Computer Science, Technion, Israel Institute of Technology
Translated and updated by Anne Weill-Zrahia
Some statistical definitions: median and mean

- The median of a series $x_1 \ldots x_n$
- Is the middle element of the sorted series. If the number of elements is even we take the mean of the 2 middle elements.
- The median is the number for which there is an equal number of smaller and equal elements as for bigger and equal elements.
- Why use it – less sensitive to extremes

Example

Stage 1: Ten people are riding on a bus in Redmond, Washington.
Stage 2: Joe Blow gets off the bus. Bill Gates gets on.

<table>
<thead>
<tr>
<th>Average income</th>
<th>Median income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>50000 $</td>
</tr>
<tr>
<td>Stage 2</td>
<td>About 50 M $</td>
</tr>
</tbody>
</table>
Arrays -motivation

Input
- A series of 800 numbers representing salaries of 800 employees.

Goal
- Read data,
- Compute (1) the mean salary (2) how many employees are paid more than the average salary (3) the median salary

Attention
- It is not always feasible to perform computations while inputting the data, therefore they should be kept in memory.
- Using many variables is not possible since:

```c
double sal0, sal1, ..., sal799;
```

- We cannot go over all the variables
- in a loop
- What can be done when we have 80,000 instead of 8000 numbers?
Array

An **array** is a **data structure** in which holds a number of values of identical type

- **Arrays in C:**
  - Data is stored in **consecutive memory locations**.
  - The index of the first element in the array is 0
  - Size of the array is fixed at declaration time

**Definition:**

```
double salaries[800];
```

- **Accessing elements of array:**

  ```
salaries[0]
salaries[799]
salaries[i]
salaries[2*i+1]
```
The out of bounds case

- If an array is of size K, then the permitted indices are 0, K-1
- C does not automatically check whether the code accesses an out-of-bounds location in the array
- Such a problem can express itself in various ways:
  - Program crashes immediately
  - Program will continue to run and crash at some future point
  - Program will terminate with wrong answers
Array initialization

- **With a loop:**
  ```c
  for ( i = 0;  i < 800;  i++ )
      scanf("%lf", &salaries[i]);
  
  for ( i = 0;  i < 800;  i++ )
      salaries[i] = hours[i] * rate[i];
  ```

- **At definition:**
  ```c
  int grades[5] = {100, 97, 79, 0, 0};
  int grades[5] = {100, 97, 79};
  ```

- **Size is defined at initialization time:**
  ```c
  int grades[5] = {100, 97, 79, 0, 0};
  int grades[] = {100, 97, 79, 0, 0};
  ```
Example: finding mean salary

```c
#include <stdio.h>
#define EMPLOYEES_NUM 10
int main()
{
    double salaries[EMPLOYEES_NUM], sum = 0.0, average;
    int i, above_average = 0;
    for ( i = 0; i < EMPLOYEES_NUM; i++ ) {
        scanf("%lf", &salaries[i]);
        sum += salaries[i];
    }
    average = sum / EMPLOYEES_NUM;
    for ( i = 0; i < EMPLOYEES_NUM; i++ )
        above_average += (salaries[i] > average);
    printf("The average is: %.2f\n", average);
    printf("There are %d salaries above the average\n", above_average);
    return 0;
}
```

- Input: series of double numbers representing salaries
- Output: average salary, number of salaries above average
Finding the median of a series

1. First method – no information on the numbers
   • Sort the numbers (this action requests “work” from the computer)
   • Compute average of the 2 middle numbers

2. Second method – some info on the number
   For instance, we know that they are integers between [0..100]
   Can we find a better (more efficient) way?
   We will see method 2 in the next slide, and method 1 in the next lectures
Finding the median in an interval

- If the data are in an interval, we can use an histogram to find the median.
- We will go over the data till we find a value for which half of the values are smaller and half are higher.
- Example 11 grades between 0-10
```c
#include <stdio.h>
#define GRADES_NUM 5001
#define MAX_GRADE 100
#define MIN_GRADE 0
#define SLOTS_NUM (MAX_GRADE - MIN_GRADE + 1)

int main()
{
    int histogram[SLOTS_NUM] = {0};
    int i, grade, median, quantity;
    for ( i = 0; i < GRADES_NUM; i++ ) {
        scanf("%d", &grade);
        histogram[grade - MIN_GRADE] ++;
    }
    quantity = 0;
    for ( median = 0; median < SLOTS_NUM; median ++ ) {
        if ( (quantity += histogram[median]) > GRADES_NUM/2 )
            break;
    }
    printf("The median is: %d\n", median + MIN_GRADE);
    ...
}
```

- Input – series of numbers 0-100 – odd number of values
- Output : median grade
bucket sort

The program had a first step in which we “compressed” the data into a temporary array.

Advantage: if we have 101 different grades but 10000 students, we will be using only 101 memory locations instead of 10000 (space saving).

We will be using this concept to sort the grades – and ask ourselves if there is a faster way to sort them.
Example : bucket sort

```c
#include <stdio.h>
define GRADES_NUM 5001
define MAX GRADE 100
define MIN GRADE 0
define SLOTS_NUM MAX GRADE - MIN GRADE + 1
int main()
{
    int histogram[SLOTS_NUM] = {0};
    int i, j, grade, median, quantity;
    for ( i = 0; i < GRADES_NUM; i++ ) {
        scanf("%d", &grade);
        histogram[grade - MIN GRADE] ++;
    }

    for ( i = MIN GRADE; i <= MAX GRADE ; i++ )
        for ( j=0 ; j < histogram[i - MIN GRADE] ; j++)
            printf("%d ", i);
...
```

• Input : series 0-100
• Output : print sorted series
### Saving information in a table

Example: number of students of each age group who study in each department:

<table>
<thead>
<tr>
<th>Department</th>
<th>18 or less</th>
<th>18-21</th>
<th>21-25</th>
<th>25 or more</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Roman languages</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>History</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Geography</td>
<td>2</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Linguistics</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Archeology</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Law</td>
<td>9</td>
<td>8</td>
<td>172</td>
<td>7</td>
</tr>
<tr>
<td>Lawn sciences</td>
<td>125</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Implementation

- We can use 2 ways:
  - Array for each department, elements classified by age group
  - Array for each age group, elements classified by department
- In the first case, it is easy to get info on a given department, but difficult on an age group
- And vice-versa.

- Solution: 2 dimensional array “array of arrays”
**Two dimensional array**

```c
int data[8][4];
```

<table>
<thead>
<tr>
<th>Subject</th>
<th>18 or less</th>
<th>18-21</th>
<th>21-25</th>
<th>25 and more</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Roman languages</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>History</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Geography</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Linguistics</td>
<td>2</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Archeology</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Law</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Lawn sciences</td>
<td>125</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The table shows the distribution of students across different fields of study and age groups.
Access to 2 dimensional array

- Access to an element of a 2D array will be with 2 indices:
  - Row number
  - Column number

- Example `data[4][2]` 5th row, 3rd column.

- From now on we can get to the data of a certain department through checking a row , and to the data on a certain department through checking a column.

- Example: find total of students from lawn sciences (7th department)

```c
int j;
int num = 0;
for ( j = 0; j < 4; j++ )
    num += data[7][j];
```
Array of arrays

- Accessing a 2 D array (named a) is indicated in C as:
  - a[i][j]  I is the row number, j is the column number
- In fact a is an array of arrays:
  - a is an array of rows
  - a[i] is an array (representing the ith row)
  - a[i][j] is a single element of a (located at the ith row, jth column).
- The indices rules (valid indices, out of bounds) are similar for each dimension (and are as in a one-dimensional array)
Characters - reminder

- Characters are represented by a numeric code – ASCII
- ASCII code has either 128 characters (7 bits) or 256 characters (8 bits) in that case it is called ISO-8859-x
- Writing character constants is as follows:
  - Most characters will be written with a quote 'a' 'b' 'A' '0' '+'
  - Some special characters : '\\' ''
  - Control characters '\n' '\t' '\a'
  - Any characters can be written using its code: '\\ddd' '\xddd'
- A character constant is an integer (int) whose value is the character in code
- To the opposite, it is possible to store numbers (representing characters) in variables of type char
Character arrays

- Arrays are built from elements of identical types, they can also be `char`, `signed char` or `unsigned char`.
- Arrays can contain:
  - Small integers on which to perform calculations
  - Integers which are codes for characters
- In the second case, such an array is called a `string`.
- A string is terminated automatically by a special character whose code is 0.
- This means that the space occupied in memory will be:
  (letters in string + 1)
Initialization of strings

- First way

```c
char string[22] = {'C', 'o', 'u', 'r', 's', 'e', ' ', '2', '3', '4', '1', '1', '2', 'i', 's', ' ', 'f', 'u', 'n', '!', '\0'};
```

- No need to specify length of array

```c
char string[] = {'C', 'o', 'u', 'r', 's', 'e', ' ', '2', '3', '4', '1', '1', '2', 'i', 's', ' ', 'f', 'u', 'n', '!', '\0'};
```

- Printing a string `%s`:

```c
printf("%s", string);
```

- Initializing a string:

```c
char string[] = "Course 234112i is fun!";
```

- Differently from other arrays, strings allow for string constants

```c
printf("%s", "Course 234112i is much fun!");
```
String constants

- A string constant is a series of characters or codes inside double quotes ex. “Apple” and can contain:
  - Normal characters
    - a b A 3 +
  - Special characters preceded by backslash
    - \  
    - "
  - Control characters
    - \n \t \a
  - ASCII code \ddd \xDD
  - Always terminated by \0
Empty string

• " " is called the empty string
• The empty string does not contain characters except for \0 at the end
  - strlen("") == 0 , sizeof("") == 1.

Do not confuse between :

- 0 – an int number with value 0
- '\0' – an int constant with value 0
  - Used to check termination of the string
- (char)'\0' - a char constant with value 0.
  - Usually it is not necessary to do the casting, automatic casting is performed according to the context

" " empty string: memory location containing
Counting characters in a word

```c
int main() {
    int len = 0;
    char str[] = "Ubuntu";
    while (str[len] != 0)
        len++;
    printf("%d\n", len);
    return 0;
}
```
Counting words in a string

Assumptions:
- String does not start or end with a space
- Single space between words
- There is at least one work

Why is this necessary?
Enlarge the code so that we do not
Need those assumptions.

int main()
{
    int len = 0, count = 1;
    char str[] = "Ubuntu is an open OS";
    while (str[len] != 0)
    {
        if (str[len] == ' ')
            count++;
    
    printf("%d\n", count);
    return 0;
}
List of words

- We can use a 2D array, or an array of strings:

```c
char names[8][8] = {
    "Avraham",
    "Sara",
    "Izhak",
    "Hagar",
    "Ishmael"
};
```

- When some of the words are too short, there is more space allocated than is really needed.