Laboratory 06 – arrays

16.11.2021

1. [4points] Write a C program that will read the elements of the array from the keyboard and calculate their sum.

a) Declare an array of integers of size `10`. In your program, use the preprocessor `#define` directive to create an `SIZE` constant that has the value `10`.
b) Use a `for` loop and `scanf` functions to load values into an array.
c) Use a `for` loop to calculate the sum of all elements in an array.
d) Print all values from the array to the screen in the reverse order.
e) Print the sum of the elements from an array.

Test data:

```
Enter 10 elements: 1 23 4 56 67 89 12 45 56 78

Array in reverse order:
arr[9] = 78
arr[8] = 56
arr[7] = 45
arr[6] = 12
arr[5] = 89
arr[4] = 67
arr[3] = 56
arr[2] = 4
arr[1] = 23
arr[0] = 1

Sum of all elements of array = 431
```

2. [4points] Write a C program that will generate `10` pseudo-random numbers and put them in an array. Use the `rand` function from the `<stdlib.h>` header to generate pseudo-random numbers. Print the index and value of the minimum and maximum elements in an array.

a) Declare an array of integers of size `10`. In your program, use the preprocessor `#define` directive to create an `SIZE` constant that has the value `10`.
b) Use a `for` loop and the `rand` function to fill an array with pseudo-random numbers between `10` and `90`.
c) Assume that the first element in the array is both the maximum and the minimum.
d) Iterate through the array to find the maximum and minimum element in the array.
e) Print the index and value of the minimum and maximum elements in an array.

Test data:

```
Elements in array are:
arr[0] = 45
arr[1] = 33
arr[2] = 15
arr[3] = 24
arr[4] = 80
arr[5] = 21
arr[6] = 80
arr[7] = 45
arr[8] = 77
arr[9] = 82
Maximum element = arr[9] = 82
Minimum element = arr[2] = 15
```

3. [7points] Write a C program that implements selection sort.

The idea of the selection sort algorithm:

```
Find the smallest element. Swap it with the first element.

Find the second-smallest element. Swap it with the second element.

Find the third-smallest element. Swap it with the third element.

Repeat finding the next-smallest element, and swapping it into the
correct position until the array is sorted.
```

Example of the selection sort algorithm:

```
int arr[] = {64, 25, 12, 22, 11};

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

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

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

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

a) Declare an array of integers of size `10`. In your program, use the preprocessor `#define` directive to create an `SIZE` constant that has the value `10`.
b) Fill the array with pseudo-random values, from 0 to 100
c) Print the array before sorting.
d) Two nested loops are needed to implement sorting. In order for the outer loop to place the array elements in ascending order, the inner loop must look for the minimum value in the part of the array that has not yet been sorted. When the inner loop finds the minimum, the outer loop must put the smallest element at the beginning of the area that has not yet been sorted.

e) Analyze the example of the algorithm above to determine the starting and ending indexes for both loops. How does the number of iterations in the outer loop depend on the size of the array? How does the starting value of the inner loop index change in subsequent iterations?

f) Print the array after sorting.

Test data:

```
Elements in array are:
arr[0] = 97
arr[1] = 4
arr[2] = 33
arr[3] = 23
arr[4] = 41
arr[5] = 20
arr[6] = 8
arr[7] = 12
arr[8] = 51
arr[9] = 76

Elements in sorted array are:
arr[0] = 4
arr[1] = 8
arr[2] = 12
arr[3] = 20
arr[4] = 23
arr[5] = 33
arr[6] = 41
arr[7] = 51
arr[8] = 76
arr[9] = 97
```

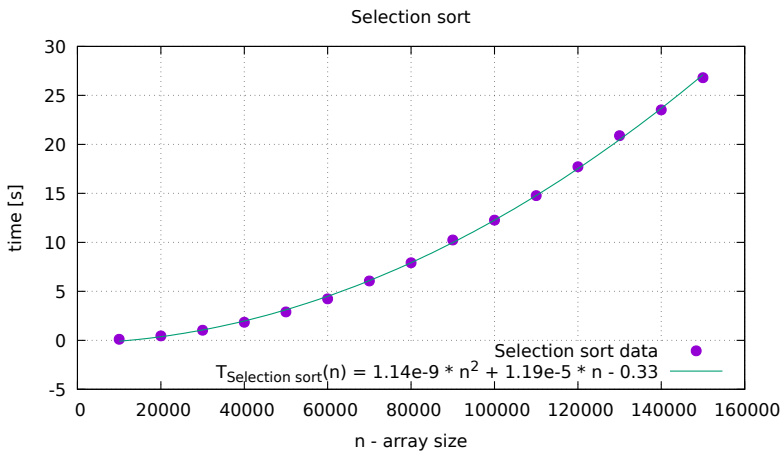
4. [5points] Let's see how the sort time depends on the size of the array. Use the `clock` function to measure how long the sort takes to run.
`clock_t clock (void); -` returns the processor time consumed by the program.
The value returned is expressed in clock ticks, which are units of time of a constant but system-specific length (with a relation of `CLOCKS_PER_SEC` clock ticks per second).

Scheme of the program:

```
//Use the clock function to save the time.
//Sorting
//Use the clock function again.
//Print the sort time.
//Repeat for arrays of increasing size.
```

Test data:

```
10000 0.105 s
20000 0.451 s
30000 1.031 s
40000 1.843 s
50000 2.900 s
60000 4.233 s
70000 6.053 s
80000 7.921 s
90000 10.237 s
100000 12.265 s
110000 14.766 s
120000 17.719 s
130000 20.890 s
140000 23.525 s
150000 26.806 s
```



Next time:
Laboratory 07 – Functions, Arrays,