Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

https://github.com/sze-info/ProgrammingBasics

November 15, 2020

Sorting numbers

Task:

- Improve the existing bubble sort program! Let the user enter the numbers to be sorted! Finish reading the input by entering a negative number.
- Entering more numbers than the size of the array must be prevented.

Problems:

- The count of numbers should be known at compile time
- Undersized array $\rightarrow$ there will be no space for the data
- Oversized array $\rightarrow$ wasting the memory
- The oversized array causes the smaller problem.

Sorting numbers

Output1

```
Enter non-negative numbers
Number #1: 2
Number #2: 4
Number #3: 1
Number #4: 3
Number #5: -1
After sorting:
1       2       3       4
```

Output2

```
Enter non-negative numbers
Number #1: 5
Number #2: 4
Number #3: 3
Number #4: 2
Number #5: 1
After sorting:
1       2       3       4       5
```

Sorting numbers

bubble5.c

```
3  #define MAX 5

37 int main(void) {
38   int used; // Number of used array elements
39   int numbers[MAX];
40   printf("Enter non-negative numbers\n");
41   used = read(numbers);
42   bubble(numbers, used);
43   printf("After sorting:\n");
44   printArray(numbers, used);
45   return 0;
46 }
```

Sorting numbers

bubble5.c

```
5  int read(int* numbers) {
6    int current, used = 0;
7    do {
8      printf("Number #%d: ", used + 1);
9      scanf("%d", &current);
10     if(current>=0 and used<MAX) {
11       *(numbers + used) = current;
12       used++;
13     }
14   } while(current>=0 and used<MAX);
15   return used;
16 }
```

Dynamic memory allocation

- The programmer decides the lifetime of dynamic variables
- `stdlib.h` must be included
- Memory allocation:
  - `void *malloc(size_t size);`
    Allocates `size` bytes of memory and returns its address. The allocated area is *uninitialized*.
  - `void *calloc(size_t nmemb, size_t size);`
    Allocates and returns the address of a continuous memory area for an array containing `nmemb` elements, each of which requires `size` bytes of memory. The area is *initialized to zeros*.
  - `void *realloc(void *ptr, size_t size);`
    Resizing the already allocated memory area without modifying the stored content.
- The return value is NULL in case of an error $\rightarrow$ should be checked
- Freeing memory: `void free(void *ptr);`
- The same meory area cannot be freed several times
- Freeing NULL does not cause problems

Sorting numbers

Tasks:

- Allocate memory dynamically for the array containing the numbers to be sorted
- Enter the count of numbers first, then allocate the required amount of memory and read the numbers
- Do not forget to free the allocated area as soon as possible

Sorting numbers

bubble6.c

```
34 int main(void) {
35   int total; // we have so many numbers in total
36   int* numbers = read(&total);
37   bubble(numbers, total);
38   printf("After sorting:\n");
39   printArray(numbers, total);
40   free(numbers);
41   return 0;
42 }
```

Sorting numbers

bubble6.c

```
4  int* read(int* total) {
5    printf("How many numbers do you want to sort? ");
6    scanf("%d", total);
7    int* numbers = (int*)malloc(*total * sizeof(int));
8    for(int i=0; i<*total; i++) {
9      printf("Number #%d: ", i + 1);
10     scanf("%d", numbers + i); // &numbers[i]
11   }
12   return numbers;
13 }
```

Student register

Tasks:

- Record the names and grades of students (one grade per student)
- Read the number of students first, then allocate the required amount of memory to store an array of name-grade structures
- Allocate memory dynamically even for the names!
- Sort the list according to the names and display it