Einführung in die Programmierung für Physiker

WS 2017/2018 – Marc Wagner

Francesca Cuteri: `cuteri@th.physik.uni-frankfurt.de`
Alessandro Sciarra: `sciarra@th.physik.uni-frankfurt.de`

Exercise sheet 11

To be corrected in tutorials in the week from from 22/01/2018 to 26/01/2018

Exercise 1 [*Arrays and dynamic memory allocation*]

In this exercise we will play around with arrays, trying to implement a clever handling of memory. We would like to have a resizeable array, which offers the possibility to add and remove elements at the end. We will call it `vector`[1]. The basic idea is to dynamically allocate some memory for the array and to allocate some more whenever is needed. To do it efficiently, we will distinguish between the concept of *size* and *capacity* of the vector. The capacity of a vector refers to the maximum number of elements for which memory has been allocated, while the size tells us how many elements are stored in the vector. When the user decides to add an element to the vector, it could happen that there is no free space for it (i.e. the vector size is equal to its capacity) and a new allocation is needed.
In this exercise, we will use some global variables to write our code more easily and to stay focused on our smart array[2].

(i) Declare a couple of global variables to store the `capacity` and the `size` of our vector. Add a global pointer to a type of your wish (e.g. `int`, `float`, `double`, etc.), called `vector`. Initialize them using appropriate values.

(ii) Define `void createVector(const unsigned int numberOfElements, const /*type*/value)`, which reserves some memory and initializes some elements of the `vector`. In order to do so,

  (a) set capacity to the smallest power of 2 larger than the provided `numberOfElements`[3];

  (b) make a first memory allocation of `capacity*sizeof(/*type*/)` bytes, using the `malloc` function;

  (c) set the size of our vector to `numberOfElements` and initialize the first elements of `vector` to `value`.

(iii) Implement a function `void destroyVector(void)`, which deallocates all the memory using the `free` function and resets the global memory to appropriate values.

(iv) Add the following three functions to your program, in which you will need to take care of any logic and/or of possible memory reallocation. Feel free to take your design decisions, but be sure to have valid arguments to justify them.

  (a) `/*type*/ at(const unsigned int index); //retrieve content with boundary check`

  (b) `void pushBack(const /*type*/ value);    //to add an element at the end`

  (c) `void popBack(void);                     //to remove an element at the end`

  Whenever you need to change the `capacity` variable, make sure of using only powers of 2, halving or doubling it. A boundary check prevents access of memory outside the array.

Time to Test! Try to use the `vector` in the `main`. At first, start initializing it with 11 elements set to some value. Print the capacity of the `vector` as well as its content using both the access operator `[]` and the `at` function. Then, free the `vector` and create it again with zero elements. Make a loop with 40 iterations and put the loop index into `vector`, printing its size and capacity at every iteration.

[1] If you are familiar with `C++`, we will get inspired from the `vector` implementation of the STL.
[2] At a later point, a new data-type should be created in order to be comfortably used in more complex programs.
[3] This strategy represents a good compromise taking into account memory usage and frequency of reallocations.

Exercise 2 [*Exam-like questions*]

(i) What is a pointer and what is the type in its definition indicating?

(ii) To what can a pointer be initialized and how can you reference the value of some variable through a pointer to that variable?

(iii) List the four possible ways of passing a pointer to a function taking into account the usage of the `const` qualifier and explain what can/cannot be modified in each case.

(iv) Which arithmetic operations can be performed on pointers, i.e. what arithmetic operators can have pointers as operands? What are the results of these operations?

(v) Under which conditions a pointer can be assigned to another pointer? What is a pointer to `void`? Can it be dereferenced? Make a use example.

(vi) What does a pointer to function store? What is it dereferenced to?

(vii) Write the header of a void function `someFunction` that expects as parameter a pointer to a function `*someOtherFunction` that receives two integer parameters and return an integer result.

(viii) Assume the following code in the main function of a program where all needed header files have been included

```
int *zPtr; // zPtr will reference array z
int *aPtr = NULL;
void *sPtr = NULL;
int number, i;
int z[5] = {1, 2, 3, 4, 5};
sPtr = z;
zPtr = z;
```

Find the error in each of the following program segments and give a brief explanation.

(a) `printf("%d\n", *aPtr);`

(b) `number = zPtr;`

(c) `number = *zPtr[2];`

(d) `for (i = 0; i <= 5; i++)   printf("%d\n", zPtr[i]);`

(e) `number = *sPtr;`

(f) `++z;`