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 12

To be corrected in tutorials in the week from 29/01/2018 to 02/02/2018

Exercise 1 [*New data types*]

Having the possibility to create customised data types is a common feature of many program languages and it helps in many situations. In this exercise we will consider again our previous implementation of the *smart array* and we will encapsulate it in a new data type. Doing so, it is possible in principle to declare multiple variables of that custom type in the same program.

- Use a pre-processor macro `TYPE` to create an alias for a chosen built-in type (e.g. `int`, `float`, etc.).
- Instead of having global variables `unsigned int capacity`, `size` and `TYPE* vector`, create a `struct` containing them. Use the name `Vector` as tag of the structure.
- Adapting your previous code, implement the following functions, which now work with the new data type.

```
struct Vector createVector(const unsigned int numberOfElements,
                           const TYPE value);
void destroyVector(struct Vector* toBeDestroyed);
TYPE at(struct Vector myVector, const unsigned int index);
void pushBack(struct Vector* myVector, const TYPE value);
void popBack(struct Vector* myVector);
```

- Implement three new functions

```
unsigned int sizeOf(const struct Vector myVector);
unsigned int capacityOf(const struct Vector myVector);
TYPE* getRawPointer(const struct Vector myVector);
```

  in order to obtain each member of any `struct Vector` variable.

- Which is the difference in passing a `const struct Vector` to a function rather than a `const struct Vector*`? Which could be an argument in favour of the first or of the second? Did you test the passed pointer for `NULL` value in your functions?
- Set `TYPE` to be `double`. In your `main`, declare a `Vector` with one only element set to 1. Add elements to the `Vector` in a `do-while` loop, in a way such that the element with index $i > 0$ is the previous one plus $(i+1)^{-2}$. Stop the loop when the difference of the last two elements of the `Vector` is smaller then $\varepsilon = 10^{-6}$. At each loop iteration, print the capacity, the size and the last element of the `Vector`.

Optional: Use `gnuplot` to understand what is going on and in particular to which number the content of the last element of your `Vector` converges to. Plot your data `using 2:(sqrt($3*6)) with lines`. It could be interesting to plot using the right y-axis the capacity of the `Vector` to see how it grows. You can use `set ytics nomirror` and `set y2tics` to activate the right y-axis and specify which axis to use at plotting time via `axes x1y2`.

Time to Test! If you did not do it during the exercise, you should test now the functions you wrote. Use the `main` to do so. What about writing a function which takes a `double*` variable and call it from the main using your `Vector`?

Exercise 2 [*Exam like questions*]

(i) What is an array? What have the elements of a given array in common? How are they distributed in memory?

(ii) How much space in memory is occupied by an array declared as `/*type*/ arr[N];`? What can you say about the type and initialization of the variable `N` which stores the size of `arr`?

(iii) List the different ways to initialize a character array with the string `"Hello"`. How many elements has the array? Given an array, how can you print its size to the screen? Make an example for the usage of `scanf` to read some given string.

(iv) When is the memory, which was reserved for an array, automatically freed throughout program execution? Can it be manually freed? Through which functions can you achieve dynamic memory allocation and deallocation?

(v) State whether the following are *true* or *false* and, in the latter case, explain why.

(a) "Array elements" outside the bounds of an array cannot be referenced with the access operator.

(b) Variables of different types can be stored in the same array.

(c) An array index can be of data type `float`.

(d) An individual array element that is passed to a function as an argument of the form `a[i]` and modified in the called function will not retain the modified value in the calling scope.

(e) If an initializers list contains more initializers than there are elements in the array, those in excess are not considered by the compiler.

(f) If the number of initializers in an initializer list is smaller than the number of elements in the array, the remaining elements are initialized to the last value in the list of initializers.

(vi) Define a function which takes an array of integers and its size as parameters and return the sum of the elements contained in the array. Use the `const` qualifier properly.