Otto-von-Guericke-University Magdeburg
Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods in Systems and Control Theory

Dr. Jens Saak, Dipl.-Math. Martin Köhler
Website: http://www.mpi-magdeburg.mpg.de/csc/teaching/18ws/sc1/

Winterterm 2018/2019

Scientific Computing 1
4th Homework

Handout: 1st Nov. 2018 **Return:** 9th Nov. 2018

*"When reading the code in about six months and asking yourself: who wrote this crap?
The answer should not be: YOU!"*

Basically that means:

- Try to always use meaningful names for functions, variables, ...
- Write documentation wherever necessary.
- Use indentation to increase readability of the code.
- Add a short statement describing its purpose and basic behavior to each function.
- ...

Exercise 1: (8 Points)

The `sizeof` operator is used to determine how many bytes are required to store an element of a given data type.

a.) Write a short program which determines the size of a single character and a single integer value.

b.) We consider the following three structures:

```
struct s1 {
        char c1;
};
struct s2 {
        char c1;
        int i1;
        char c2;
};
struct s3 {
        char c1;
        char c2;
        int  i1;
};
```

Determine their size in bytes and relate them to the values obtained in a.). Which values do you expect and what does the compiler generate?

c.) Create an instance of each structure of b.). Use the address-of operator to determine address of the structure and of each of its components. Since these addresses can be interpreted as unsigned long integers, one can compute the memory distances between two components of a structure. How does `c1`, `c2`, and `i1` related to each other? What changes between structure `s2` and `s3`?

Exercise 2: (14 Points)

The stack is one of the basic data containers in computer science. It contains an ordered collection of data and defines the following operations on it:

push – pushes an element on top of stack,

pop – takes the most recently added element and removes it from the data collection,

top – returns the most recently added element of stack,

empty – indicates if the stack contains elements or not.

Access to other elements than the element on top is not possible. Since a computer has only a finite amount of memory a practical implementation has a maximum size.

Write an implementation of a stack containing double precision values. To this end, use the following structure as template to represent the stack:

```
struct stack_t {
        int maxsize;
        double *elements;
};
```

If you need more components in the structure feel free to add them.
Implement the following functions to manage the stack:

a.) The `stack_init` function takes the stack and the maximum size of the stack as arguments and initializes an empty stack with the given maximum size. It returns an indicator if the initialization was successful or not.

b.) The `stack_empty` function takes the stack as argument and returns *true* if there are no elements on the stack, *false* otherwise.

c.) The `stack_top` function takes the stack as argument and returns the last recently added value of the stack without removing it.

d.) The `stack_push` function takes the stack and a double value as arguments and places the double value on top of the stack. Its return-value should indicate whether this was successful or the stack already reached its maximum size.

e.) The `stack_pop` function takes the stack and a double value as arguments and returns the top of the stack via the given double value. The return-value of the function has to indicate whether an element could be popped from the stack or not.

f.) The `stack_clear` function takes the stack as input and frees all allocated memory locations.

In some cases the function arguments need to be passed as pointer rather than as value. Think of this issue during the implementation and use the proper solution.

Demonstrate the usage of your stack implementation by realizing the following procedure: Read a sequence of positive double precision numbers from the standard input and stop reading when the first negative number is entered. Print this sequence in reverse order to the screen.

Overall Points: 22