Informatics II, Spring 2024, Exercise 1

Publication of exercise: February 19, 2024
Publication of solution: February 26, 2024
Exercise classes: February 26 - March 1, 2024

Learning Goal

- Gain an initial understanding of the C programming language. This should include initial experience with the syntax, getting to know some peculiarities of C and being able to program simple algorithms yourself.
- Understand and implement simple sorting algorithms.
- Gain some basic intuition for the runtime of an algorithm.

Task 1 [Easy/Medium]

In the following task you will find six code snippets, each of which should show you a characteristics of C or draw your attention to possible dangers in this language. Run all of the code snippets and answer the following questions for each:

- What do you observe?
- How do you explain what you observe?

You can also find all snippets on OLAT.

1.1: Danger when programming in C, possible source of bugs

```
1 #include "stdio.h"
2 int main () {
3     int a = 2147483647;
4     int b = 2147483648;
5     int c = 2147483649;
6     printf("%d, %d, %d", a, b, c);
7     return 0;
8 }
```

1.2: Danger when programming in C, possible source of bugs

```
1 #include "stdio.h"
2 int main() {
3     int myArray[20];
4     for(int i = 0; i < 20; i++) {
5         printf("%d \n", myArray[i]);
6     }
7     return 0;
8 }
```

1.3: Danger when programming in C, try to avoid this at all costs

```
1 #include "stdio.h"
2 int main() {
3     int myArray[1];
4     myArray[0] = 0;
5     myArray[1] = 1;
6     myArray[2] = 2;
7     printf("%d, %d, %d", myArray[0], myArray[1], myArray[2]);
8     return 0;
9 }
```

1.4: Danger when programming in C, possible source of bugs and leads to code that is difficult to understand

```
1 #include "stdio.h"
2 int main() {
3     int myArray[] = {72, 101, 108, 108, 111, 32,
4                      87, 111, 114, 108, 100, 33};
5     for(int i = 0; i < 12; i++) {
6         printf("%d", myArray[i]);
7     }
8     printf("\n");
9     for(int i = 0; i < 12; i++) {
10        printf("%c", myArray[i]);
11    }
12    return 0;
13 }
```

1.5: Special property of C

```
1 #include "stdio.h"
2 int main() {
3     int myArray[5];
4     int size1 = sizeof(myArray);
5     int size2 = sizeof(myArray[0]);
6     int size3 = size1 / size2;
7     printf("%d, %d, %d", size1, size2, size3);
8     return 0;
9 }
```

1.6: Special property of C

```
1  #include "stdio.h"
2  int main() {
3      char myString[] = "hello";
4      int stringSize = sizeof(myString) / sizeof(myString[0]);
5      printf("%d, ", stringSize);
6      for (int i = 0; i < stringSize; i++) {
7          printf("%c", myString[i]);
8      }
9      return 0;
10 }
```

Task 2 [Medium]

One of the simplest ways to compress a string is the so-called Run-Length-Encoding (RLE). In this method, consecutive identical characters in a string are replaced by the amount and the character itself.

An example of this would be "AAABBAAAA" is transformed by RLE to "3A2B4A", where "3A" stands for three consecutive "A"s, "2B" for two consecutive "B"s and "4A" for four consecutive "A"s.

Write the function `rleCompression(char string[], int length)` in C that takes a string and its length as input and outputs the compressed version on the terminal (the function doesn't need to return the result). To make it easier for you, you will find a code skeleton on OLAT, so that you only have to insert the relevant parts of the code.

Task 3 [Medium]

3.1: Write the function `bubbleSort(int array[], int length)` in C, which takes an array and its length as input and sorts it in ascending order using the bubble sort algorithm.

3.2: Also write the function `insertionSort(int array[], int length)` in C which takes an array and its length as input and sorts it using the insertion sort algorithm.

3.3: Modify the two previously programmed sorting algorithms so that they now count how often the innermost loop has been run through. After running the algorithm, output the number of runs on the terminal.

Execute the algorithms with the following array: [0, 1, 3, 4, 2, 8, 9, 5, 6, 7]

What do you notice? How do you explain what you observed?

3.4: Change the input array such that your algorithm outputs the highest possible counter. You are allowed change the order of the elements but you need to keep the same numbers.

3.5: On OLAT you will find a C program with an array of 100,000 elements. Sort this list with your bubble sort. How many seconds does the algorithm take? How often is the innermost loop run through for this list length? Try to calculate this number and then check with your counter.

Note: If this large list causes difficulties for your computer, you can also just do the calculation.

Task 4 [Medium/Hard]

In computer science, a subarray is a contiguous sequence of elements within an array. For example, for the array [1, 2, 3], the subarrays are [1], [2], [3], [1, 2], [2, 3], and [1, 2, 3].

Write a C function `zeroSubarray(int array[], int length)` that takes an array of integers and its length as parameters and returns 1 if it is possible to find a subarray where the sum of all integers adds up to zero. Otherwise, the function should return 0.

Examples:

- $[3, -2, 4, 2, 1, -5]$ should return 1, since [-2, 4, 2, 1, -5] is a contiguous subarray that adds up to zero.
- $[1, 2, 3, 4, 5, 0]$ should return 1, since [0] is a contiguous subarray that adds up to zero.
- $[-2, 4, -1, 5, 9, -12]$ should return 0, since there is no possibility to find such a subarray that adds up to zero.