SWENT 2024SS

Cross-Exercise 5

Tips:

- *Solve as many exercises as possible, at least 3.*
- *You can hand in the solutions until 23:55 the day before the next class.*
- *Only mark exercises that you can explan and show live. If you can't explain your solution sufficiently you don't get any points even for the correct solution.*
-

Exercise 1)

Create an integer array of length 50 and read numbers into the array until you read a 0 (also store the 0, but no further inputs!).
Then, use a while loop to output the array, but only output the entered numbers and do not access the unfilled positions of the array! Hint: You know that the last number you read in is a 0 or you have read in the maximum number of 50 numbers.
An Example:

```
Enter up to 50 numbers (Null terminates the input):
5
1234
-379
-1
0
The entered numbers are: 5, 1234, -379, -1,
```

Exercise 2)

Create a character array. Before that, define a constant to set the maximum length of the array. Now read a word into this array (make sure that the array is large enough to store all characters + null terminator).
Next, read a character. This character should be replaced in the word read at the beginning. Then read another character that should replace the first read one. Replace all occurrences of the first read character in the word with the second read character and print the changed word.

An Example:

```
Enter a word: Hallo
Which letter do you want to replace?? a
With which letter do want to replace it with? e
The resulting word is: Hello
```

Exercise 3)

Read floating-point numbers with a loop until either a 0 is entered or you have read 10 numbers. Count the number of numbers read. Now you want to know if the numbers read were only entered once or if a number was entered multiple times.
For example, in the sequence of numbers 3.14, 5.0, 1.01, -12.6, no number is contained more than once, but in the sequence of numbers 3.14, 5.0, 1.01, 5.0, 5.0, 5.0 is contained three times.
In the first example, your program should output that no numbers occur multiple times. In the second example, your program should output that there are duplicate numbers in the input.
You do not need to know how many or which numbers occur multiple times, only that the input numbers are either unique (example 1) or not unique (example 2).
To do this, write the test for the uniqueness of the numbers in a function with the signature _Bool allNumbersUnique(double numbers[], int usedFields) that takes a double array and the number of entries occupied in it as arguments and returns true (1) if the array contains unique numbers as previously described, otherwise the function returns false (0). Use your allNumbersUnique() function to test the numbers you entered for uniqueness and output the result of the test in a comprehensible way on the command line.

Exercise 4)

Create a struct called "aStruct" that has three entries with arbitrary names: a character, an integer, and a double entry.
Create a variable called "myStruct" that has the type of your defined struct and assign arbitrary values to the three entries (from the command line or directly in the code, as you wish).
Now print the memory address of myStruct. Also print the memory addresses of each entry in this struct, as well as the values of the entries (not in a loop, but simply with explicit code lines for each entry).
Now create a pointer to your myStruct variable named "myStructPointer".
Print the memory address of the pointer myStructPointer. Now also print the memory addresses of each entry of the struct that myStructPointer points to, as well as the values of the entries. For this, use only the pointer myStructPointer (not the myStruct variable anymore) in your code to print the memory addresses and values! Tip: -> operator
What do you notice about the memory addresses and what conclusion can you draw from this about the memory layout of the struct?

Exercise 5)

Create a struct that describes a product. It should contain a name, ID, description, and price. Create an array of this struct (e.g., with 10 entries), which is your shopping cart.

Pass the shopping cart array by reference (e.g., pointer or address) to a function that reads the data of products in the shopping cart and returns the number of products read as a return value.
That means, in this function, you read the data of several product entries for the shopping cart array from the command line.
Ask the user if another product should be read after reading a product, if not then break the reading and return to main().
Then pass the now filled shopping cart array to another function (again by reference), where you output the product entries from the shopping cart array one by one.
Note: Keep track of how many products you have read in order to know how many you need to output!

Exercise 6)

Create a struct named elementStruct with three entries: Id, Name, and a pointer named next_element of the same type as the struct in which it is located (elementStruct). The pointer (of type: struct elementStruct) should always contain the address of the next element (i.e., point to the next element).
Then, in main(), create 4 variables of the struct elementStruct and assign arbitrary values for Id and Name. Then, you set the pointer next_element in each struct to the address of the next struct variable (e.g., ErstesElement.next_element=&ZweitesElement) and the next_element pointer in the last struct should point to the first struct. Finally, in a while loop, print the entries of the structs one by one by starting with the first and then following the reference next_element to get the next element. Hint: Use a variable that always contains the currently output struct and overwrite it at the end of a loop iteration with the next element, but be careful of infinite loops!