Week 4

Exercise 1: Command line arguments and reference parameters

Write a simple program that gets an integer as a command line argument. Send the number to a function as a reference parameter and change the value of the number to its square inside the function. Print the value of the number before and after the function call.

Be sure to check that exactly one command line argument is given. If not, the program outputs: "You did not enter a number!".

Note also that the function should not return anything (`void`). You can use `atoi` function to convert string to int.

Add a newline to the end of each outputted line.

Example use case: ./program 12

```
The number is 12
The number is 144
```

Exercise 2: Memory allocation and freeing

Make a program that takes a positive integer as a command line parameter. Your program should first check that a positive integer is given.

Then create an integer array with size equal to the given command line parameter. Ask the user to give numbers to fill the array. After the numbers are stored in the array, the numbers in the array are printed. The used memory is freed at the end of the program.

Based on the above description the program needs 5 functions. In addition to these functions below, the program needs to check the command line argument. If it is not present, the execution of the program is terminated and message "you did not give an argument" is given.

- `sizeCheck`: The function checks whether the command line parameter is an integer greater than zero. If the conversion fails or the given integer was not positive, the function returns zero and the user is given the error message "The parameter was not a positive integer.". The program execution ends.

- `CreateArray`: The function allocates memory for an integer array of the desired size and returns the pointer to the array. If the memory allocation fails, the program will give an error message with the "Memory allocation failed." The program execution ends.

- `fillArray`: The function gets a pointer to the array and size of the array. Then it asks for the numbers and fills the array.

- `PrintArray`: The function gets a pointer to an array and size of the array. Then it prints the elements of the array according to example.

- `freeMemory`: The function gets the parameter pointer to the array and free it allocated memory.

At the end of each function, print out the success of the operation according to the examples below. Use the format "`%d`" to print the elements of the array

Add a newline to the end of each line

Example use case: ./program 3

```
Memory allocated successfully.
The array has 3 elements.
Enter number 1:
4
Enter number 2:
2
Enter number 3:
0
Array filled.
The array includes the numbers: 4 2 0
Array is printed.
Memory freed.
```

Exercise 3: Copy and compare records (structs)

Write a program with a PERSON struct and functions for copying and comparision.

The program must first create two PERSON structs, `first` and `second`. The program then compares these two records and tells the user if their contents are the same or not.

The program then copies the data in `first` to PERSON `third` with its own copy function `personCopy`. Finally, the program compares the contents of record 1 with the contents of record 3 and tells the user whether the information in them is the same and whether the memory addresses in records 1 and 3 are the same.

The PERSON struct must contain 2 variables, a char array of size 30 for `firstName` and an int for `age`. In `personCopy,` the fields are copied to the new record one field at a time, so that after the copying, both records have the same first name and age.

Comparing records means that the values of all the fields in the records are compared and if there are differences, the records are not the same.

Comparing the addresses of records means comparing the addresses of the first memory locations in the records. There are no ready-made solutions for copying a struct in C, so in this task you yourself must implement the struct copying and comparison functions. String functions in `string.h` can be used to handle strings.

Allocate records `first` and `second` statically and record `third` dynamically when needed. Be sure to free the dynamically allocated memory before the end of the program. The error message that is not visible in the example program is "Memory allocation failed."

Example use case:

```
Enter first name of the first person:
Ville
Enter age of first person:
19
Enter first name of the seocnd person:
Valo
Enter age of the second person:
21
Comparing two persons:
person 1 info: Ville, 19
person 2 info: Valo, 21
The data in the records are not the same.
Third person has been created.
Copied Info of the first person to third.
Comparing the first person and the third person:
person 1 info: Ville, 19
person 2 info: Ville, 19
The data in the records are the same.
Comparing the first person and the third person addresses:
The addresses of the records are not the same.
```

Exercise 4: Menu-based program for processing an integer array

Write a menu-based program that prints the numbers of an integer array and that can resize the current integer array. These operations should be repeated until the user wants to stop running the program.

At the beginning of the program, you need to create a pointer to an integer array and a variable that maintains the size *N* of the array. The size has a default value of zero. When the user resizes the array, dynamically allocate the required amount of memory and fill it with the numbers 0 to *N*-1. When printing, the program prints all the values in the array on the same line, separated by a space.

Memory allocation and printing should be performed in their own functions. Always make sure that the memory allocation is successful and free the allocated memory at the end of the program. Use "`%d`" format to print the integers.

The error messages that do not appear in the example run are:

- "Memory allocation failed.”
- "Array size cannot be negative."
- "Unknown selection, please try again."

Example use case:

```
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
4
Unknown selection, please try again.
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
1
the array is empty.
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
2
Enter a new size for the array:
4
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
1
The array includes the numbers: 0 1 2 3
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
2
Enter a new size for the array:
7
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
1
The array includes the numbers: 0 1 2 3 4 5 6
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
2
Enter a new size for the array:
0
1) Print the items in the array
2) Resize the array
0) Stop
```

```
Select an Item:
1
the array is empty.
1) Print the items in the array
2) Resize the array
0) Stop
Select an Item:
0
```