A+ will be down for a version upgrade on Tuesday 03.01.2023 at 9-12.

This course has already ended.

« Exam (/elec-a7100/2021-summer/exam... December 2021 Retake Exam » (/elec-a71...

ELEC-A7100 (/elec-a7100/2021-summer/)
/ Exam (/elec-a7100/2021-summer/exams_summer2021/) / 1 Assigned Questions

Assigned Questions¶

Arrays in C

Implement function `print_diagram`, that gets an array of integers, and prints a diagram according the table contents in the following format: For each number, the same number of `=` is printed on a line followed by a newline. So, an array containing numbers `3`, `1` and `2` would produce the following output:

```
===
=
==
```

The parameter `n` tells how many numbers there are in the array.

Implement function `sum_of_largest_and_smallest`, that finds the smallest and the largest value in an array and returns their sum. The parameter `n` tells how many numbers there are in the array.

Note: a function for comparing two elements in an array is provided with the template, if you wish to use it.

Implement function `pair_array_max`, that gets two integer arrays, `arr1` and `arr2`, as parameters, both of which have `n` elements. The function goes through all the members and compares them to each other. If `arr2` has a bigger number than `arr1`, the function swaps the members between the two arrays. So, after running the function, `arr1` contains the bigger numbers in each pair and `arr2` contains the smaller numbers in each pair.

Note: a function for swapping two values with pointers is provided with the template.

Implement function `print_table`, that prints an `x` times `y` table, that contains the numbers starting from the given number in parameter `start`. Each table element takes four characters and they are separated by a space. So, given input `x = 3`, `y = 4` and `start = 5`, the output would be:

```
5    6    7
8    9   10
11   12   13
14   15   16
```

Implement function `copy_sorted`, that copies the given array as sorted to a new array with dynamically allocated memory (reserve the memory yourself). A function for comparing two elements in an array is provided in the template. The parameter `n` tells how many elements there are in array.

Note: you don't need to free the memory for `array`.

Implement function doublecat, that concatenates two array with floating point values, `arr1` with `n1` items, and `arr2` with `n2` items. The needed memory for the resulting array needs to be dynamically allocated.

Note: you don't need to free the memory for `arr1` or `arr2`.

Implement function `swap_max`, that swaps the largest numbers of two arrays. The amount elements in `arr1` is `n1`, and the amount of elements in `arr2` is `n2`. The function should find the largest elements in the arrays and swap them.

Note: a function for swapping two values with pointers is provided with the template.

Implement function `median`, that calculates the median value of the integers in the array and returns it. The parameter `n` tells how many numbers there are in the array.

Note: median value is the middle value of an array when it has been ordered. There will always be an odd number of elements in the array. A function for comparing two elements in an array is provided with the template.

Strings in C

Implement function `convert_non_alphabetical`, that converts all non-alphabetical characters in a string to a lowercase character a, `'a'`. The modifications are done in-place, so no memory allocations are needed.

Hint: `ctype.h` functions

Implement function `convert_non_numbers`, that converts all non-numerical characters in a string to zeroes, `'0'`. The modifications are done in-place, so no memory allocations are needed.

Hint: `ctype.h` functions

Implement function `count_non_printable`, that calculates all characters in a string that are NOT printable and returns the amount as an `unsigned int`.

Hint: `ctype.h` functions

Implement function `convert_non_printable`, that converts all non-printable characters in a string to spaces. The modifications are done in-place, so no memory allocations are needed.

Hint: `ctype.h` functions

Implement function `copy_reversed_as_lower`, that copies the string str as reversed to the buffer `copy`. The copied string should be a valid string in C, i.e. it ends with `'\0'`. During the copying, all characters are converted to lowercase characters with the `tolower` function.
Implement function `copy_reversed_as_upper`, that copies the string str as reversed to the buffer `copy`. The copied string should be a valid string in C, i.e. it ends with `'\0'`. During the copying, all characters are converted to uppercase characters with the `toupper` function.
Implement function `count_non_numbers`, that calculates all characters in a string that are not numbers and returns the amount as an `unsigned int`.

Hint: `ctype.h` functions

Implement function `count_non_alphabetical`, that calculates all characters in a string that are non-alphabetical and returns the amount as an `unsigned int`.

Hint: `ctype.h` functions

Binary operations in C

Implement function `bit_strings_and_to_number`, that gets two binary numbers as strings and returns the result of an AND operation between the two numbers as an integer. So, with parameters `str1 = "1101"` and `str2 = "0101"`, the function would perform the `AND` operation resulting in the binary number `0101`, `0x5` in hexadecimal format. The binary number strings have an arbitrary length, at maximum 16 bits.

Hint: the `strtol` function.

Implement function `fetch_bits_and_combine`, that takes the leftmost four bits from `byte` and creates a binary number from them, and also takes the rightmost four bits and creates another number from them. Then, the function returns the result of `AND` operation between these two binary numbers as the four leftmost bits of a new number and the result of `OR` operation between the two as the four rightmost bits of the new number.

So, for example, given `byte = 10101101` (`0xAD` in hexadecimal form), the function would separate two binary numbers, `1010` and `1101`, and would then perform the `AND` operation, resulting in `1000`, `0x8` in hexadecimal form, and the `OR` operation, which would give the result `1111`, `0xf` in hexadecimal form. And the function would return the value `10001111`, `0x8f`.

Implement function `count_set_bits_in_array`, that calculates the set bits in the array containing binary numbers. Parameter `n` tells how many binary numbers there are in the array.

So, for example, given `array = { 0xA3, 0x58 }`, the function would return `7` as `0xA3 = 10100011` in binary, and `0x58 = 01011000` in binary.

Implement function `combine_with_operations`, that gets two 4-bit binary numbers as parameters, called `a` and `b`. The function should return a number that has the following information in it:

- the four leftmost bits hold the result of the `AND` operation between the two binary numbers
- the next four bits hold the result of the `OR` operation between the two binary numbers
- the next four bits hold the result of the `XOR` operation between the two binary numbers
- the last four bits hold the result of the `NOT` operation on the binary number `a`.

So, for example, given `a = 1111` and `b = 1010`, then result would be `1010111101010000`, `0xAF50` in hexadecimal and `44880` in decimal format.

Hint: remember to limit `a`'s `NOT` operation to four bits with bit mask: `& 0xf`.

Implement function `fetch_bits_to_binary`, that fetches the values of given bits in `bit_idxs` of an `array`, which has `n` elements in it. The function creates a new binary number with these bits.

So, for example, given `array = { 0xA3, 0x58 }`, and `bit_idxs = { 0, 2, 10 }`, the function would return `0x6` (`0000 0110` in binary), as `0xA3 = 10100011` in binary, and `0x58 = 01011000` in binary, and bits 0, 2 and 10 are 1, 1 and 0.

Implement function `fetch_bits`, that fetches the values of given bits in `bit_idxs` from binary a number `array`, which has `n` elements in it. The function calculates the sum of the bits and returns it.

So, for example, given `array = { 0xA3, 0x58 }`, and `bit_idxs = { 0, 2, 10 }`, the function would return `2`, as `0xA3 = 10100011` in binary, and `0x58 = 01011000` in binary, and bits 0, 2, and 10 are 1, 1 and 0.

Implement function `get_set_bit_in_combination`, that gets two binary numbers as parameters, `a` that has 5 bits and `b` that has 3 bits. The function should combine the two numbers in a way, that the leftmost five bits are the bits from `a`, and the last three bits are the bits from `b`. Then, the function should return the value of the bit in the index idx.

For example, given `a = 00001000`, `b = 00000111`, and `idx = 5`, the function would return 1, as the combination would be `01000111`, and the bit in index 5 is 1: `01000(1)11`.

Implement function `combine_to_string`, that gets two binary numbers as parameters, `a` that has `2` bits and `b` that has `6` bits. The function should combine the two numbers in a way, that the leftmost six bits are the bits from `b`, and the last two bits are

the bits from `a`. Then, then function should write the binary number to the given string, `str`.

So, for example, given `a = 11` and `b = 101100`, after the function has been called, `str` should contain the string `"10110011"`.

Fix code

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. You can also assume that there are no mistakes with the road points and their handling in any way. The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically roads. The program saves them in a `Map` structure that holds a location name, a roads array and the amount of roads in the array. The `Road` structure holds the data for a single road, and stores the name of the road and a point (latitude-longitude pair) of the road.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There is one mistake in the `createRoads` function.
- There are three mistakes in `the createMap` function.
- There is one mistake in the `printRoadInfo` function.
- There is one mistake in the `freeMemory` function.

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. The main function has no problems, and it cannot be changed. The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically points of interest. The program saves them in a `Map` structure, that holds a location name, a `POI` array and the amount of points of interest in the array. The POI structure holds the data for a single point of interest, and stores the name and type of the POI, as well as the location (a latitude-longitude pair) of the POI.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There are two mistakes in the `createPOIs` function.
- There are two mistakes in the `createMap` function.
- There is one mistake in the `printPoiInfo` function.
- There is one mistake in the `freeMemory` function.

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. The main function has no problems, and it cannot be changed. You can also assume that there are no mistakes with the road points and their handling in any way. The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically map tiles. The program saves them in a `Map` structure, that holds a location name, a `MapTile` linked list and the amount of tiles in the array. The MapTile structure holds the data for a single map tile, and stores the id of the map tile, the center coordinate (a latitude-longitude pair) of the tile and the size and the zoom level for the tile.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There are two mistakes in the `createMapTile` function.
- There is one mistake in the `createMapTiles` function.
- There is one mistakes in the `createMap` function.
- There is one mistake in the `printTileInfo` function.
- There is one mistake in the `freeMemory` function.

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. The main function has no problems, and it cannot be changed. You can also assume that there are no mistakes with the road points and their handling in any way. The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically roads. The program saves them in a `Map` structure, that holds a location name and linked list containing the roads. The `Road` structure holds the data for a single road, and stores the name of the road and a point (latitude-longitude pair) of the road.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There are two mistakes in the `createRoad` function.
- There is one mistake in the `createRoads` function.
- There is one mistake in the `createMap` function.
- There is one mistake in the `printRoadInfo` function.
- There is one mistake in the `freeMemory` function.

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. The main function has no problems, and it cannot be changed.

The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically map tiles. The program saves them in a `Map` structure, that holds a location name, a MapTile array and the amount of tiles in the array. The `MapTile` structure holds the data for a single map tile, and stores the id of the map tile, the center coordinate (a latitude-longitude pair) of the tile and the size and the zoom level for the tile.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There are two mistakes in the `createMapTiles` function.
- There are two mistakes in the `createMap` function.
- There is one mistake in the `printTileInfo` function.
- There is one mistake in the `freeMemory` function.

The code below doesn't work as it's supposed to. Fix the functions so that they work as the comments describe and don't produce compiler errors or warnings, nor valgrind errors or memory leaks. The main function has no problems, and it cannot be changed. The structures are also defined correctly, and no includes are missing. In general, the logic of the code is correct.

The program handles map data, more specifically points of interest. The program saves them in a `Map` structure, that holds a location name, a POI linked list. The `POI` structure holds the data for a single point of interest, and stores the name and type of the POI, as well as the location (a latitude-longitude pair) of the POI.

There are six mistakes in the program, each of them clearly visible either in the compiler errors/warnings or valgrind output. Note that there might be more than one error/warning per mistake as a single mistake can create many problems at once.

- There are two mistakes in the `createPOI` function.
- There is one mistake in the `createPOIs` function.
- There is one mistake in the `createMap` function.
- There is one mistake in the `printPoiInfo` function.
- There is one mistake in the `freeMemory` function.

Program

Implement program that maintains a database for library. For each book following information should be stored:

- Name of book (arbitrarily long string)
- Author(s) (arbitrarily long string)
- Publication year

You cannot make any assumptions about the maximum length of book name or authors.

Implement following functions:

- **add_book** that adds one book to the registry
- **print_books** that prints all books in the registry, one per line. For each book the program should output name, authors and publication year.

Implement also a `main` function that calls the aforementioned functions. The main function should add at least three books and print the register after that.

The program should allocate the needed memory dynamically, but it should use only as much memory as is really needed.

Scoring:

- appropriate data structures: **2p**
- working add_book: **2p**
- working print_books: **2p**
- working main function: **2p**

Rampe and Naukkis are musicians who perform at gas stations throughout Finland.

 Rampe and Naukkis (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_images/rampenaukkis.jpg)
They need a program that stores a song list for their gigs, along with the lenghts of the songs. The song list contais the following information for each song:

- Name of the song
- Length of the song

Some of the songs have very long names, so you cannot assume a maximum length for the name.

Implement the following functions:

- **add_song**, that adds a new song to the song list. The function arguments should contain (at least) the name of the new song and its length, along with any other arguments you may need.
- **print_songs**, that outputs all songs in the song list, along with their lenghts, each on a separate line. The lenghts of the songs should be output as `MM:SS`, where MM indicates minutes and SS indicates seconds.

In addition, implement a main function that calls the above two functions. In main function, add at least three songs to the list, and output its content afterwards.

The proram allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_song: **2p**
- working print_songs: **2p**
- working main function: **2p**

Implement a program that maintains a shop inventory that contains numbers of items in storage, along with their prices. For each item, the following information needs to be stored:

- title of product
- quantity of items in storage
- price (including cents)

The name of the product can be long, and you cannot assume maximum length for the title.

You will need to implement the following functions:

- **add_product** that adds new item to the inventory. The title, quantity, and price will be given as parameters.
- **print_products** that will print all products in the inventory along with the price and quantity information.

In addition, implement a main function that calls the aforementioned functions. The main function should add at least three products to the inventory, and outputs its content after that.

The proram allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_product: **2p**
- working print_products: **2p**
- working main function: **2p**

Implement program that maintains a vehicle registry. For each vehicle the following information will be stored:

- register number
- model of the vehicle

The register number is at most 7 characters long, but the vehicle model may be arbitrary long.

Implement the following functions:

- **add_vehicle** that adds a new vehicle to the registry. The model and register number are included as function parameters.
- **print_registry** that outputs all vehicles in the registry

In addition, implement a main function that calls the aforementioned functions. The main function should add at least three vehicles to the registry, and outputs its content after that.

The proram allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_vehicle: **2p**
- working print_registry: **2p**
- working main function: **2p**

Implement a program that maintains a student registry that contains information about students. You should store the following information:

- student name
- student number
- starting year

Student's name can be long, and you cannot assume a maximum length for it. Student number is six characters long, that contains mostly numbers, but can also contain letter characters.

Implement the following functions:

- **add_student** that adds a new student to the register. You should give the name, student number and starting year as a parameter.
- **print_students** that prints all students along with their information.

Implement also a main function that calls the aforementioned functions. The main function should add at least three students and print the register after that.

The prorgram should allocate the needed memory dynamically, but it should use only as much memory as is really needed.

Scoring:

- appropriate data structures: **2p**
- working add_student: **2p**
- working print_students: **2p**
- working main function: **2p**

Implement a KELA-registry that maintains the names and social security numbers of people, along with information of their monthly pension. For each person the following information should be stored:

- name (contains full name, and can be long)
- social security number (11 characters)
- monthly pension (in euros, but not necessary exact euro amounts, and can contains cents, too)

You cannot make assumptions about the maximum length of the name (some people have very long names).

Implement the following functions:

- **add_person** that adds a new person to the registry.

- **print_pensioners** that outputs all persons in the registry, along with their full information.

In addition, implement a main function that calls the above two functions. In main function, add at least three persons to the registry, and output its content afterwards.

The program allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_person: **2p**
- working print_pensioners: **2p**
- working main function: **2p**

Implement a tool for a satellite factory to help them follow their projects. The program needs to keep track of new spacecraft under construction which requires you to store the following information:

- *project name* (can be long)
- *identifier* (15 characters)
- *project budget* in euros

You cannot make assumptions about the maximum length of the name as engineers have a habit of inventing very long acronyms for their projects.

Implement the following functions:

- **add_satellite** that adds a new satellite project to the database.
- **print_satellites** that outputs all satellites in the database, along with their full information.

In addition, implement a main function that calls the above two functions. In main function, add at least three satellites to the database, and output its content afterwards.

The program allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_satellite: **2p**
- working print_satellites: **2p**
- working main function: **2p**

Implement a log book for a ship. The program needs to keep track of the route of the ship which requires you to store the following information:

- *port name* (can be long)
- *IMO GISIS identifier of the port* (5 characters and 4 numbers separated with a dash, e.g. FIHEL-0030)
- *date of arrival*

You cannot make assumptions about the maximum length of the port's name as some have really long ones.

Implement the following functions:

- **add_entry** that adds a new log entry to the database.
- **print_logbook** that outputs the full information of all entries in the logbook.

In addition, implement a main function that calls the above two functions. In main function, add at least three entries to the logbook, and output its content afterwards.

The program allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_entry: **2p**
- working print_logbook: **2p**
- working main function: **2p**

Implement the supporting program for a departures display at an airport. The display needs to keep track of multiple aircraft which requires you to store the following information:

- *destination* (can be long)
- *flight number* (max. 6 characters, first a two letter identifier, then numbers, e.g. AY666, AA1776)
- *time of departure* (time as hh:mm)

You cannot make assumptions about the maximum length of the destination's name as some places have really long names.

Implement the following functions:

- **add_flight** that adds a new flight to the database.
- **print_flights** that outputs all flights in the database, along with their full information.

In addition, implement a main function that calls the above two functions. In main function, add at least three flights to the database, and output its content afterwards.

The program allocates its memory dynamically, but should only use as much memory as it really needs.

Scoring:

- appropriate data structures: **2p**
- working add_flight: **2p**
- working print_flights: **2p**
- working main function: **2p**

`Download folder url`

(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/skeleton.zip)

A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays1.zip)
A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays2.zip)
A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays3.zip)
A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays4.zip)
A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays5.zip)
A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays6.zip)
A7 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays7.zip)
A8 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/arrays8.zip)
A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings1.zip)
A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings2.zip)
A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings3.zip)
A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings4.zip)
A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings5.zip)
A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings6.zip)
A7 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings7.zip)
A8 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/strings8.zip)
A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits1.zip) A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits2.zip) A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits3.zip) A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits4.zip) A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits5.zip) A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits6.zip) A7 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits7.zip) A8 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/bits8.zip) A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code1.zip) A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code2.zip) A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code3.zip) A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code4.zip) A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code5.zip) A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/fix-code6.zip) A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program1.zip) A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program2.zip) A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program3.zip) A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program4.zip) A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program5.zip) A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/program6.zip)

In the exam, there are five questions on the following topics:

1. **Arrays in C**: You are required to implement a single function that performs the specified operation.
2. **Strings in C**: You are required to implement a single function that performs the specified string operation.
3. **Bit operations in C**: You are required to implement a single function that performs the specified bit operations.
4. **Fix code**: You are required to fix errors in the provided code.

5. **Program**: You are required to write a program that has the specified functionality.

Getting Started

1. Each question is shown in its exercise box. **By the end of the description, you can find a link to download** *Visual Studio Code* **workspace folder along with the source code template for each question.**
    - **Note that you need to download different files for each question!**
2. You can download the ZIP file as you did with the exercise modules.
    - If you are using Windows Subsystem for Linux (WSL), Linux or MacOS based system, you can use `wget` tool. `download url` can be copied by right clicking the download link and selecting `copy link`.

      ```
      wget --no-check-certificate <download url>
      ```

    - After downloading the template, you need to unzip its content. If you are using Windows Subsystem for Linux (WSL), Linux or MacOS based system, you can use `unzip` tool.
    - The zip file names end with a number. These are used for personalization of the exam, and does not signify anything on your side.
3. **You can open the *Visual Studio Code* workspace.**
    - Navigate to the directory (folder) you extracted the template.
    - If you are using command-line, enter the following command:

      ```
      code exam.code-workspace
      ```

    - Otherwise, you can do the same by clicking `File - Open Workspace ...`.
4. **Read the comments in the template.**

Testing

- The provided workspace has the following `Run Tasks` pre-configured. You can access them by clicking `Terminal - Run Task...`.
    - **Build**: builds the source file `main.c` and creates executable `main.out`.
    - **Clean**: deletes the executable file `main.out`.
    - **Valgrind**: runs `valgrind` with `main.out`.
- The provided workspace has a launch configuration pre-configured so that you can use the debugger right away.
- The template files **do not** have any tests for your implementations. You are responsible for creating the tests for the first 3 questions if you need validation.
    - Your test code will not be evaluated (except question 5: **Program**).

Submission

- You can submit your solutions as you did in the exercises.

- **You should use the correct submission box of the solution.**
- *Your file should have a valid* `main` *function.*
- The number of submissions is not limited, but it is faster and easier to develop and test on your local environment.
- After submission, you can see `compiler` and `valgrind` outputs below the question description.
- If there are `compiler warnings` or memory leaks detected by `valgrind`, you will see some penalties. Their importance depends on the question. In some questions, `valgrind` errors are ignored, and in some `compiler warnings` originating from `main` or `test` functions are ignored.

Evaluation

- We will manually grade your solutions.
- You will be graded based on your last submissions.
- Your coding style might affect your grade as it is in the programming task. Therefore, pay attention when naming your structures, variables and functions.

⚠ The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

You cannot submit this assignment

Question 1

main.c

Choose File No file chosen

Submit

The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

Show anyway

⚠ The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

You cannot submit this assignment

Question 2

main.c

Choose File No file chosen

Submit

The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

Show anyway

⚠ The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

You cannot submit this assignment

Question 3

main.c The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

Choose File No file chosen

Show anyway

Submit

⚠ The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

You cannot submit this assignment

Question 4

main.c The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

Choose File No file chosen

Show anyway

Submit

⚠ The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

You cannot submit this assignment

Question 5

main.c The deadline for the assignment has passed (Monday, 30 August 2021, 16:59).

Choose File No file chosen

Show anyway

Submit

« Exam (/elec-a7100/2021-summer/exam... December 2021 Retake Exam » (/elec-a71...