Assigned Questions¶

Arrays in C

Implement a function that creates a sorted copy of an integer `array` of `n` elements in **ascending order**.

The function creates a sorted copy of the input array by dynamically allocating the destination array (i.e. you should allocate the memory for the destination array). The function should return a new array that contains the same elements of the argument array, but its elements should be sorted in ascending order.

Hint

`stdlib.h` has some useful functions that might help you to sort the array. You might need the comparison function `compare_ints_ascending` provided in the template if you prefer to use *stdlib* function for sorting the array.

Implement a function that creates a sorted copy of an integer `array` of `n` elements in **descending order**.

The function creates a sorted copy of the input array by dynamically allocating the destination array (i.e. you should allocate the memory for the destination array). The function should return a new array that contains the same elements of the argument array, but its elements should be sorted in descending order.

Hint

`stdlib.h` has some useful functions that might help you to sort the array. You might need the comparison function `compare_ints_descending` provided in the template if you prefer to use *stdlib* function for sorting the array.

Implement a function that finds the difference between **median** and **mean** value of the elements in an integer `array` .

The function calculates the mean and finds the median value of the `array` , which contains `n` number of elements. It assumes that `n` is an odd integer when finding the median value. The mean value is calculated using integer division on the sum of the all elements in the array.

Hint

- **Median** is the middle element of the array when it is sorted.

`stdlib.h` has some useful functions that might help you to sort the array. You might need the comparison function `compare_ints` provided in the template if you prefer to use *stdlib* function for sorting the array.

- **Mean** value is the arithmetic average of the values in the array.

  When calculating the mean value, you should use integer arithmetic to find the average. If you use floating point arithmetic, you might get a wrong result.

Implement a function that concatenates two arrays with floating point values into a new dynamically allocated array.

The function concatenates `arr1` of `n1` double elements and `arr2` of `n2` double elements into a new dynamically allocated array. The destination array should have the values of `arr1` in its first `n1` elements, and its preceding elements (`n1` onwards) should have the values of the elements of `arr2`.

Important

You don’t need to free the memory for `arr1` or `arr2`.

Implement a function that finds the **median** of the elements in an integer `array`.

The function calculates the median value of the `array`, which contains `n` number of elements. It assumes that `n` is an odd integer.

Hint

Median is the middle element of the array when it is sorted.

`stdlib.h` has some useful functions that might help you to sort the array. You might need the comparison function `compare_ints` provided in the template if you prefer to use *stdlib* function for sorting the array.

Implement a function that compares each pair of elements of two integer arrays, and swaps them if the value of element of `arr2` is larger than the value of element of `arr1`.

The function gets two integer arrays of `n` elements. It goes through all the elements of both arrays and compares the element of `arr1` with the element of `arr2`. If `arr2` element has a larger value than `arr1`, the function swaps the element values between the two arrays. So, after running the function, `arr1` contains the larger numbers and `arr2` contains the smaller numbers in each pair.

Hint

A function for swapping two values is provided in the template.

Implement a function that compares each pair of elements of two integer arrays, and swaps them if the value of element of `arr2` is smaller than the value of element of `arr1`.

The function gets two integer arrays of `n` elements. It goes through all the elements of both arrays and compares the element of `arr1` with the element of `arr2`. If `arr2` element has a smaller value than `arr1`, the function swaps the element values between the two arrays. So, after running the function, `arr1` contains the smaller numbers and `arr2` contains the larger numbers in each pair.

Hint

A function for swapping two values is provided in the template.

Implement a function that finds the smallest and the largest elements in an integer `array` of `n` elements, and returns their sum.

Hint

`stdlib.h` has some useful functions that might help you to sort the array to find the smallest and the largest values of an array. You might need the comparison function `compare_ints` provided in the template if you prefer to use *stdlib* function for sorting the array.
Implement a function that finds the maximum values of two arrays `arr1` with `n1` integer elements and `arr2` with `n2` integer elements and swaps them.

The function first finds the largest elements in the arrays `arr1` and `arr2`, and then swaps them. After calling the function, `arr1` should contain maximum value of `arr2`, and `arr2` must contain the maximum element of `arr1`. The other elements of both `arr1` and `arr2` should remain the same.

Hint

A function for swapping two values is provided in the template.

Implement a function that finds the smallest and the largest elements in an integer `array` of `n` elements, and swaps them.

For example, `array = {1, 2, 3}` and `n = 3`, after calling the function the `array` becomes `array = {3, 2, 1}`.

Hint

A function for swapping two values is provided in the template.

Implement a function that finds the minimum values of two arrays `arr1` with `n1` integer elements and `arr2` with `n2` integer elements and swaps them.

The function first finds the smallest elements in the arrays `arr1` and `arr2`, and then swaps them. After calling the function, `arr1` should contain minimum value of `arr2`, and `arr2` must contain the minimum value of `arr1`. The other elements of both `arr1` and `arr2` should remain the same.

Hint

A function for swapping two values is provided in the template.

Strings in C

Implement a function that converts all non-alphabetical characters in a null-terminated string to lower case a, `'a'`. The function changes the found non-alphabetical characters inplace so that no dynamic allocation is needed.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including alphabetical characters.

Implement a function that converts all non-numerical characters in a null-terminated string to zero character, `'0'`. The function changes the found non-numerical characters inplace so that no dynamic allocation is needed.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including numerical (digit) characters.

Implement a function that converts all non-printable characters in a null-terminated string to a space character, `' '`. The function changes the found non-printable characters inplace so that no dynamic allocation is needed.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including printable characters.
Implement a function that converts all punctuation characters in a null-terminated string to a space character, `' '`. The function changes the found punctuation characters inplace so that no dynamic allocation is needed.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including punctuation characters.

Implement a function that converts all white space characters in a null-terminated string to a tilde character, `'~''`. The function changes the found white space characters inplace so that no dynamic allocation is needed.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including white space characters.

Implement a function that copies a string by reversing its characters and converting all upper case letters to lower case.

This function reverses its first null-terminated string argument `str` into the string `copy` while converting all upper case characters to lower case characters. For example, if

```
str[] = "HeLLo"
```

After calling the function, `copy` would be

```
copy[] = "olleh"
```

The function should make copy a valid C string by properly terminating it with null termination character.

Hint

stdlib provides useful character handling functions in `ctype.h`.

Important

`copy` can be assumed to have enough space for copying the `str`. You do not need to allocate memory for it.

Implement a function that copies a string by reversing its characters and converting all lower case letters to upper case.

This function reverses its first null-terminated string argument `str` into the string `copy` while converting all lower case characters to lower case characters. For example, if

```
str[] = "HeLLo"
```

After calling the function, `copy` would be

```
copy[] = "OLLEH"
```

The function should make copy a valid C string by properly terminating it with null termination character.

Hint

stdlib provides useful character handling functions in `ctype.h`.

Important

`copy` can be assumed to have enough space for copying the `str`. You do not need to allocate memory for it.

Implement a function that counts all non-alphabetical characters in a null-terminated string.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including alphabetical characters.

Implement a function that counts all non-numerical characters in a null-terminated string.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including numerical (digit) characters.

Implement a function that counts all non-printable characters in a null-terminated string.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including printable characters.

Implement a function that counts all punctuation characters in a null-terminated string.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including punctuation characters.

Implement a function that counts all white space characters in a null-terminated string.

Hint

stdlib provides useful character handling functions in `ctype.h`. `ctype.h` documentation also states different character classes, including white space characters.

Binary operations in C

Implement a function that accepts two binary number strings, converts them to integers, ANDs these numbers and returns the result.

The function receives two null-terminated strings of binary numbers, which are composed of at most 16 bits. After converting these strings into integers, it ANDs these two numbers, and returns the result.

For example, if `str1 = "1101"` and `str2 = "0101"`, the function would perform the AND operation, which result in the binary number `0101` (`0x05` in hexadecimal format).

Hint

stdlib.h has very useful function for converting null-terminated strings into numbers. In particular, check `strtol` function.

Implement a function that combines one 2-bit number with a 6-bit number, and returns a string of a byte of which 6 left most bits are of the bits of the second number and 2 right most bits are the bits of the first number.

The function get 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 6 bits are the bits from `b`, and the last 2 bits are the bits from `a`. Then, the function should write the binary number to the argument string, `str`.

For example, given `a = 3` (binary `11`) and `b = 44` (binary `101100`), after the function is called, `str` should be `str[] = "10110011"`.

Implement a function that gets two 4-bit binary numbers, say `a` and `b`, and returns a 16-bit integer value by performing several bit-wise operations on them. The 16-bit number is formed as follows:

- 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`.

For example, when the first number is `a = 1111` and the second is `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 a function that counts the number of cleared (value 0) bits in a byte array of `n` elements.

For example, if `array[] = {0xA3, 0x58}` and `n = 2`, the array bytes has binary values `1010 0011 0101 1000`, and the function returns 9. Since

```
1010 0011 0101 1000
 ^ ^ ^^   ^ ^   ^^^
```

has 9 clear bits.

Implement a function that counts the number of set (value 1) bits in a byte array of `n` elements.

For example, if `array[] = {0xA3, 0x58}` and `n = 2`, the array bytes has binary values `1010 0011 0101 1000`, and the function returns 7. Since

```
1010 0011 0101 1000
^ ^    ^^  ^ ^ ^
```

has 7 set bits.

Implement a function that fetches left most four bits of a byte to create a number, and fetches the rightmost four bits of the same byte to create another number. Then, it ANDs these two parts to obtain a four bit number, and ORs the parts to obtain another four bit number. The results of AND operation and OR operation are combined in a way that result of AND is the left most four bits, and the result of OR operation is the four right most bits of a new byte.

For example, if `byte = 10101101` (`0xAD` in hexadecimal), the function would separate two binary numbers, `1010` and `1101`. Then, it would AND these numbers, resulting in `1000` (`0x8` in hexadecimal), and OR them, which would result `1111` (`0xf` in hexadecimal). The function would return the value `10001111` (`0x8f` in hexadecimal), after combining them in a byte.

Implement a function that fetches bits from a byte array, of which bit indexes are stated in `bit_idxs` array containing `n` elements. It returns the sum of these bits after clearing the bits that are set (value 1).

The function fetches bits from a byte array, which contains suitable number of bytes to cover the largest bit index in `bit_idxs` array. It assumes that the bytes in the array are composed as follows: if `array[] = {0xA3, 0x58}`, it assumes a number `0xA358 = 1010 0011 0101 1000`, and assigns index 0 to the left most bit. If `bit_idxs[] = {0, 2, 10}` and `n = 3`, it fetches

```
1010 0011 0101 1000
^ ^          ^
```

1, 1, 0 since bit 0 is 1, bit 2 is 1, and bit 10 is 0. Then, it sums these bit values, which is 2. Before returning the sum, it clears the bits that are set, that is, it clears bit 0 and bit 2 so that the array becomes

```
0000 0011 0101 1000
^ ^          ^
```

`array[] = {0x03, 0x58}`.

Implement a function that fetches bits from a byte array, of which bit indexes are stated in `bit_idxs` array containing `n` elements. It returns the sum of these bits.

The function fetches bits from a byte array, which contains suitable number of bytes to cover the largest bit index in `bit_idxs` array. It assumes that the bytes in the array are composed as follows: if `array[] = {0xA3, 0x58}`, it assumes a number `0xA358 = 1010 0011 0101 1000`, and assigns index 0 to the left most bit. If `bit_idxs[] = {0, 2, 10}` and `n = 3`, it fetches

```
1010 0011 0101 1000
^ ^          ^
```

1, 1, 0 since bit 0 is 1, bit 2 is 1, and bit 10 is 0. Then, it sums these bit values, and returns 2.

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"`.

Implement a function that fetches bits from a byte array, of which bit indexes are stated in `bit_idxs` array containing `n` elements. It combines these bits to form a new byte.

The function fetches bits from a byte array, which contains suitable number of bytes to cover the largest bit index in `bit_idxs` array. It assumes that the bytes in the array are composed as follows: if `array[] = {0xA3, 0x58}`, it assumes a number `0xA358 = 1010 0011 0101 1000`, and assigns index 0 to the left most bit. If `bit_idxs[] = {0, 2, 10}` and `n = 3`, it fetches

```
1010 0011 0101 1000
^ ^          ^
```

1, 1, 0 since bit 0 is 1, bit 2 is 1, and bit 10 is 0. Then, it combines these bits such that the bit value in the bit_idxs[0] is the left most bit of the resultant number, and other bit values follow it. Therefore, the function would return `110 = 0x6`.

Implement a function that combines the right most 5 bits of one byte, say `a`, a with the rightmost 3 bits of another byte, say `b` to form a new byte, of which the left most 5 bits are from the first byte and the rightmost 3 bits are from the second byte. It fetches the bit value at `idx` index of the resultant byte and returns its value.

For example, if `a = 8` (binary `01000`), b = `7` (binary `111`) and `idx = 5`, the function first forms byte `01000111` (hexadecimal `0x47`), and then fetches the bit at index 5,

```
0100 0111
      ^
```

which is 1. Then, it returns 1.

Fix code

The code in the provided template doesn't work as it is supposed to. Fix the functions so that they work as the comments describe and they don't produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- No includes are missing.
- In general, the logic of the code is correct.

The program handles map data, more specifically points of interest (POI). The program saves them in a `Map` structure, that holds a location name, a POI array and the number 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:

- 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 in the provided template doesn't work as it is supposed to. Fix the functions so that they work as the comments describe and they don't produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- No includes are missing.
- In general, the logic of the code is correct.

The program handles map data, more specifically points of interest (POI). 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:

- 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.

The code in the provided template doesn’t work as it is supposed to. Fix the functions so that they work as the comments describe and they don’t produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- 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 an array of 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:

- 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 in the provided template doesn’t work as it is supposed to. Fix the functions so that they work as the comments describe and they don’t produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- 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 a 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:

- 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 in the provided template doesn’t work as it is supposed to. Fix the functions so that they work as the comments describe and they don’t produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- 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:

- 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 in the provided template doesn't work as it is supposed to. Fix the functions so that they work as the comments describe and they don't produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- 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 and a MapTile linked list. 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:

- 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 in the provided template doesn't work as it is supposed to. Fix the functions so that they work as the comments describe and they don't produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- No includes are missing.
- In general, the logic of the code is correct.

The program handles map data, more specifically public transportation stops. The program saves them in a `Map` structure, that holds a location name and an array of `TransportStop` structures. The `TransportStop` structure holds the data for a single public transportation stop, and stores the name and type of stop, as well as the location (a latitude-longitude pair) of the stop.

There are six mistakes in the program:

- There are two mistakes in the `createStops` function.
- There are two mistakes in the `createMap` function.
- There is one mistake in the `printStopInfo` function.
- There is one mistake in the `freeMemory` function.

The code in the provided template doesn't work as it is supposed to. Fix the functions so that they work as the comments describe and they don't produce compiler warnings/errors or valgrind errors. You can make the following assumptions:

- The main function has no problems.
- The structures are defined correctly.
- No includes are missing.
- In general, the logic of the code is correct.

The program handles map data, more specifically public transportation stops. The program saves them in a `Map` structure, that holds a location name and a linked list of `TransportStop` structures. The `TransportStop` structure holds the data for a single public transportation stop, and stores the name and type of the stop, as well as the location (a latitude-longitude pair) of the stop.

There are six mistakes in the program:

- There is one mistake in the `createStop` function.
- There are two mistakes in the `createStops` function.
- There is one mistake in the `createMap` function.
- There is one mistake in the `printStopInfo` 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/retake_summer2021_arrays1.zip)
`A2`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays2.zip)
`A3`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays3.zip)
`A4`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays4.zip)
`A5`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays5.zip)
`A6`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays6.zip)
`A7`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays7.zip)
`A8`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays8.zip)
`A9`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays9.zip)
`A10`
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays10.zip)
`A11`

(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_arrays11.zip)
A1
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings1.zip)
A2
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings2.zip)
A3
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings3.zip)
A4
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings4.zip)
A5
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings5.zip)
A6
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings6.zip)
A7
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings7.zip)
A8
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings8.zip)
A9
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings9.zip)
A10
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings10.zip)
A11
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings11.zip)
A12
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_strings12.zip)
A1
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits1.zip)
A2
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits2.zip)
A3
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits3.zip)
A4
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits4.zip)
A5
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits5.zip)
A6
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits6.zip)
A7
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits7.zip)
A8
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits8.zip)
A9
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits9.zip)
A10
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_bits10.zip)
A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code1.zip) A2
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code2.zip) A3
(https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-

code3.zip) A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code4.zip) A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code5.zip) A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code6.zip) A7 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code7.zip) A8 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_fix-code8.zip) A1 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program1.zip) A2 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program2.zip) A3 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program3.zip) A4 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program4.zip) A5 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program5.zip) A6 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program6.zip) A7 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program7.zip) A8 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program8.zip) A9 (https://gitmanager.cs.aalto.fi/static/elec_a7100_2021summer/_downloads/retake_summer2021_program9.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.**

   Caution

   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.** The templates have doxygen comments for the functions, and make sure that you follow the instructions there. The instructions in the template are more detailed than the ones in A+ system.
5. **DO NOT CHANGE THE NAME OF THE FUNCTIONS IN THE TEMPLATE.**
6. **Do not write to `stdout` to check the functionality of your implementation, but use `my_tests`** function in the template for that purpose.

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.**
- **After submitting your implementation, please validate that the question you received is the same in the feedback page.** If they differ, you must use the correct question, since in some platforms (e.g. MacOS) the question page is not refreshed appropriately.
- *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 (Tuesday, 7 December 2021, 16:59).

You cannot submit this assignment

Question 1

The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

main.c

Show anyway

Choose File No file chosen

Submit

⚠ The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

You cannot submit this assignment

Question 2

The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

main.c

Show anyway

Choose File No file chosen

Submit

⚠ The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

You cannot submit this assignment

Question 3

The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

main.c

Show anyway

Choose File No file chosen

Submit

⚠ The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

You cannot submit this assignment

Question 4

The deadline for the assignment has passed (Tuesday, 7 December 2021, 16:59).

main.c

Show anyway

Choose File No file chosen

Submit

Loading assignment...

« December 2021 Retake Exam (/elec-a7100/2...