Some C-related tasks

In/output

Adder

Write a program that adds two numbers!

The program should read both numbers from the standard input and output to standard output.

Adder 2

Write a program that adds two integers!

The program receives the inputs as arguments (argv) and output to standard output.

Adder 3

Write a program that adds two real numbers!

The program should read both numbers from the standard input and write to standard output. The output should be formatted: it should output the HTML code for the result, boldened and with a colour.

The colour should be specified as an argument, but if no argument is specified, the colour light grey should be used (#dedede).

Calculator

Write a program that can do arithmetic operations on two numbers.
The program expects the input on the standard input and outputs the result to the standard output.

The format of the input is:
`NUMBER1 OPERAND NUMBER2`, example: `1 + 2`.

Programmable calculator

Write a program that can do arithmetic operations on two numbers!
The program expects two filenames as arguments: the first arguments point to the input file and the second argument points to the output file.

The input file contains a list of expressions similar to the last task, that is, `NUMBER1 OPERAND NUMBER2`.

There is a single expression per line. The output should contain the results of each expression, each result in it’s own line.

Example input:

```
1 + 2
1 - 1
1 - 2
```

Example output:

```
3
0
-1
```

Pointers and strings

Sum of array

Write a function that can sum up an array of integers!

The two arguments passed to the function is an array pointer
and the size of the array.

Three smallest elements

Write a function that takes an integer array and outputs the three smallest elements!

Within radius

Write a function that takes a double array, it’s size and a radius R. The function examines whether each element of the array is within the radius R of it’s neighbors (that is, the absolute difference of the examined item and each neighbor is less than R).

Return with a boolean value whether this condition is true for all elements!

Twice

Write a function that takes an integer array and returns with a pointer to the first element that occurs at least twice within the array!
If no such element exists, return with NULL.

Tricky

Write a function that takes a double array and returns with the element with index 4. If no such element exists, return 0.

Hello X

Write a program that can read the user’s name from standard input and output a greeting to said user.

Spaces

Write a program that reads a sentence from standard input and:

- counts and displays the number of spaces within the sentences
- outputs the same sentence without any spaces

Indefinite article

Write a program that can count the indefinite article “a” within a sentence!

The indefinite article may appear at the beginning of the sentence, but not at the end, and a comma may precede or follow it, otherwise it is delimited by spaces.

Note: from now on, we refer to strings. Strings are null-terminated character arrays.

Decimals

Write the function with function signature `int dec_to_int(char* s)` , which takes a string containing decimal digits and converts it to an integer. The input is in base 10.

Write two versions of this function, one making use of `sscanf` and one without using it!

Length

Write your own version of `strlen` !

This function takes a string and counts the number of characters in it.
The null terminator should not be counted.

Please do not use the standard library string handling functions.

Copy

Write your own version of `strcpy` !

This function takes two pointers to strings and copies the content of the second string argument (source) to the first string argument (destination).

You can assume that there is enough “space” in the array given by the first argument.

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.

Concatenation

Write your own version of `strcat` !

The function takes two strings and appends the second string to the end of the first string.

The function returns nothing (void), as it performs the operation on the first string. You can assume there is enough space in the character array given by the first argument to store the two strings concatenated.

Don’t forget null termination!

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.

Structs and enums

Points in space

Create a struct that can store a coordinate in a two-dimensional coordinate system!

You are free to use integers or real numbers as you see fit. The program reads the coordinate from the standard input, the X and Y component separated by a semicolon.
Example: `1;-3`

Points in space, but in use

Write a program that uses the previously declared Point struct!

The program should read two different coordinates and output the length of the straight line that interconnects the two.
Make use of functions!

Calculator 2

Write a program that can do arithmetic operations on two numbers!

The program expects the input on the standard input and outputs the result to the standard output.

The format of the input is:
`NUMBER1 OPERAND NUMBER2` , example: `1 + 2` .

Make use of enums to identify the operands within the code.

Can you do this?

Can you do this with a struct? Why not? Name at least 2 reasons!

```
struct Wrong {
  int count;
  double numbers[count]; // WRONG
}
```

Dynamic memory

Polygons

Create a polygon struct! You should re-use the Point structure from earlier.

Write a program using this polygon struct: it should first read the amount of vertices / points from the standard input, then read all the points for the polygon. Finally, you should output the polygon in such format:

```
[1.5;3 2.1;4 4.4;1.1]
```

Dynamic concatenation

Write a function that takes two strings and returns with a pointer to a new string that contains the two strings concatenated!

Make sure to allocate the exact amount of memory required to store the result and not more. The input strings should not be modified (`const char*`).

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.

Write a program that makes use of this function!
Make sure there is are no memory leaks and everything is freed.

Dynamic extension

Write a function that takes two strings and concatenates the second argument onto the first one! The two previous implementations suffer from flaws: the first implementation requires the caller to provide enough space for the concatenated result, the second one creates an additional allocation.

Let’s write a smarter implementation that modifies and extends the first string in place!

The function takes two arguments like before, but it re-allocates the first string to be able to accommodate the new, larger string. Following that, it concatenates the second string onto the first string.

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.
Make sure that there are no memory leaks.

Example snippet to use the code:

```
char* x = (char*) malloc((5+1) * sizeof(char));
my_strcpy(x, "apple");

x = my_dynstrcat(x, "tree");
printf("%s", x); // appletree

free(x);
```

Reading from input like adults

So far, if you are reading from input, you needed to make sure to not enter too many characters: your char array had a static size and if your input was too long, your program either read just part of it (good) or crashed (bad).

Let’s write our own program to handle indefinite sized inputs!

To do this, let’s read the text in fixed-size chunks using `fgets`. The buffer should be 50 bytes large. `fgets` signals us whether the input line has ended and we can take advantage of this in our program:

- if the line has ended, the read input has a ‘\n’ newline character at the end.
- if not, there is no such character in the character array.

While you read the text, dynamically extend the size of the string containing the whole text as you read it in fixed-size chunks.
Make sure you allocate exactly the right amount of space for the string!

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.

Trimming

Write a function that takes a constant string and outputs a string that has the space characters trimmed from the beginning and on the end.
The allocation for the new, output string should be exactly as large as the new string.

Please do NOT use the string-handling functions of the standard library, feel free to reuse previously written code.

Dynamic String struct

Create a struct for a string that contains both the character array that is the string and it’s size (DynString)!
The character array should NOT be null-terminated.

This struct should be implemented in it’s own header file (dynstring.h).

String to DynString

Write a function that takes a string and fills out an already existing DynString struct!
The character array should be exactly as large as the stored string and the length member should reflect the accurate length of the string.

DynString write

Write a function that takes a DynString and writes it to standard output!
You can use `printf("%c", ...);` to write individual characters, remember that.

Example code:

```
DynString sz;

string_to_dynstring(sz, "hello");
dynstring_write(sz); // hello
```

DynString space-removal

Write a function that removes all spaces from such DynString (“this is a string” -> “thisisastring”)!
All operations on the DynString should end with the character array allocation contained within the DynString
being exactly as large as required and not larger and the string length structure member being correct.

Example code

```
DynString sz;

string_to_dynstring(sz, "hello world");
dynstring_unspacify(sz);
dynstring_write(sz); // helloworld
```

DynString to string

Write a function that takes a DynString and turns it into a string!

Example code:

```
DynString sz;

string_to_dynstring(sz, "hello");

printf("%s\n", dynstring_to_string(sz));
```

Dynamic DynsSrings

Write a function that takes a string and creates a completely new DynString structure! Do not forget to also write a function that frees up such DynString.

Example code:

```
DynString* sz = dynstring_create("hello world");
dynstring_write(sz); // hello world
dynstring_free(sz);
```