Homework 1

As part of this assignment, you will practice your understanding of C.

Problem 1

You are given a number N, where N is a natural number such that N >=1. Write a program that prints the first Fibonacci numbers. The program will accept a single command line argument which is N. The output will include the first N Fibonacci numbers, each printed on a new line. As an example, calling ./a.out 10 should print:

```
0
1
1
2
3
5
8
13
21
34
55
```

Problem 2

You are given a number N, where N is a natural number such that N >=1. Write a program that prints all the prime numbers between 1 and N (inclusive). The program will accept a single command line argument which is N. The output will include on the first line the number N, followed by a prime on each of the following lines. As an example, calling ./a.out 10 should print:

```
10
1
2
3
5
7
```

Problem 3

You will have to implement bubble sort to sort a doubly-linked list of integers in ascending order. As a starting point, I'm providing problem2_template.c. The template provides build_list which when is passed the length of the list to be generated. The function generates a sequence of random numbers that are to be sorted. The print_list function is a utility that prints the list. You have to implement two approaches to perform bubble sort.

- In bubble_sort_copy_value you have to implement bubble sort by manipulating the value field in the linked list. So, you can simply perform the bubble sort by swapping the values of two consecutive elements. This function should be easier to implement since you do not have to reorganize the list.
- The bubble_sort_copy_ref you have to implement bubble sort without modifying the value fields (you can read the values, just not write to them). The function has to be implemented by manipulating the pointers of the elements in the linked list. This will require some careful consideration of how pointers must be assigned.

Output: The output of the code should be the sorted array.

Hint: For the bubble_sort_copy_ref, I found the following illustration useful. Originally, we need to consider four variables before_a, a, b, after_b that refer to elements in the list. I'm allowing for prev_a and next_b to be pontetially NULL (which happens in the beginning/end of the list). The next and prev pointers for these are as follows:

```
before_a   ===(next)==>   a   ===(next)==>  b ===(next)==> after_b
before_a   <==(prev)===   a   <==(prev)===  b <==(prev)=== after_b
```

If we need to switch the values of a and b, we need to switch the pointer around so that we end up with the following organization:

```
before_a   ===(next)==>   b   ===(next)==>  a ===(next)==> after_b
before_a   <==(prev)===   b   <==(prev)===  a <==(prev)=== after_b
```

Now, you just need to think about how you need to change the pointers of before_a, a, b, and after_b from the top case to the bottom case. Be careful about when before_a and after_b are NULL. If the list has at least two elements, a and b cannot be NULL.

To debug, drawing the relationships between variables as you step through code is quite useful. Remember gdb (or CLion's debugger) and valgrind are your friends. Use them in times of trouble!

Problem 4

As part of this assignment, you will be using Numpy, a library for the Python programming language, to perform linear algebra operations. Numpy provides a high-performance multidimensional array object, and tools for working with these arrays.

Step 1: I'm providing below an implementation for loading and saving such arrays to disk in load_save.py. Read the save_3d_raw and load_3d_raw and write a short one-paragraph description of what is the storage format used by these functions. This should include a description of the data types, their size, and the layout of the data in the file.

Step 2: As a starting point, I'm providing array.h that provides a simple 3D array structure and the definition of functions that you should implement including code for loading and saving 3D arrays to disk.

Your submission should include a file named "problem5.md" answering step 1 and a file named "arrays.c" implementing step 2. You are not to make any changes to array.h. Additionaly, provide "main.c" that implements a test case for the functions you implemented.

Problem 5: 1D Convolution for Audio Processing

1. Importance of 1D Convolutions

1D convolutions are a fundamental operation in signal processing, particularly for:

- **Audio Processing**: Filtering operations like noise removal (low-pass), extracting frequency bands (band-pass), or adding effects like echo/reverb.
- **Time Series Analysis**: Smoothing trends, detecting anomalies, or forecasting in financial or sensor data.
- **Speech Recognition**: extracting features from raw audio waveforms.

Formula

For a 1D input signal x and a kernel k of size K, the 1D cross-correlation (to compute convolution, the kernel should be flipped) is defined as:

$$y[t] = \sum_{i=0}^{K-1} x[t+i] \times k[i]$$

Where: - $y[t]$ is the output at index t.

- We use **valid** convolution (no padding). This means we only compute the output where the kernel fully overlaps with the input.
- If input length is L and kernel length is K, the output length will be $L_{out} = L - K + 1$.

2. C Implementation Task

Your goal is to implement the **1D convolution** operation in C. Below is its definition:

```
// input: pointer to input samples (length L_in)
// kernel: pointer to kernel samples (length K)
// output: pointer to pre-allocated output (length L_out = L_in - K + 1)
void conv1d_valid(const float *input, size_t L_in,
                  const float *kernel, size_t K,
                  float *output);
```

Requirements:

0. **Bounds checking**: Make sure that K >=1, L_in >= K, and input, kernel, and output are not NULL. If any of these conditions are not met, call exit(-1).
1. **Load Data**: Use your `array3d_load` from Problem 4.
   - Note that `input.bin` and `kernel.bin` will be loaded as 3D arrays.
   - **Crucially**: The audio is mono (1 channel). The data is stored such that `dim1 = 1`.
   - You can effectively treat `input.data` as a single flat array of length `dim2` (or `dim2*dim3`), or just access the first row.
2. **Implement Convolution**:
   - Write a function to perform 1D convolution.
   - **No padding is required.** You should blindly apply the kernel to valid positions.
   - The loop for the output should range from `0` to `L_in - K`.
   - `output[t]` corresponds to the dot product of `kernel` and the window `input[t : t+K]`.
3. **Save Output**:
   - Save your result to `output_c.bin` using `array3d_save`.
4. **Verify**:
   - Compare your `output_c.bin` with the reference `output.bin`.
   - (Optional) You could write a small python script to convert your `output_c.bin` to a wav file to hear if it sounds the same!

Hint

Since we are doing **valid** convolution (padding=0): - The output will be shorter than the input. - `output[i]` is computed by taking the dot product of the `kernel[0...K-1]` and the input segment `input[i...i+K-1]`. - You do NOT need to handle indices outside the bounds of the input array.