HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2021
ASSIGNMENT 2: PROGRAMMING IN C

It is recommended to do Lab 1 and Lab 2 before this assignment.

It is important that you submit the assignment in time. **See the deadline in the Studium.**

The assignment consists of three parts, described in the sections below. Start by creating a directory for this assignment, and put the resulting files for each part in subdirectories `part1`, `part2` and `part3`. When you are ready to submit your assignment, prepare your submission by carefully following the instructions in the section "Preparing your submission" below.

Important:

- You must write the code by yourself.
- Your code must compile without errors and warnings and run on the computers in the lab rooms (e,g. on vitsippa.it.uu.se).
- Your code must be commented and be well-formatted.

Your code for each part should contain a makefile so that it can be built by simply doing "make" and cleaned up by doing "make clean".

Part 1

Pascal's triangle can be written as a lower triangular matrix. The entry in the nth row and kth column of Pascal's triangle is a binomial coefficient given by the formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\cdots(n-(k-1))}{k(k-1)(k-2)\cdots 1} = \prod_{i=1}^{k} \frac{n+1-i}{i}.$$

Note that if indexing starts with 0, then $n \geq 0$, $k \geq 0$ and $n \geq k$. For example, the unique nonzero entry in the row 0 is $\binom{0}{0} = 1$.

Write a C program which prints Pascal's triangle. Your program should accept the number of rows in the triangle as a parameter from the command line. The name of the executable should be "`triang`".

For example, running

```
./triang 5
```

you should get the following output

Date: January 2, 2021.

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
```

There should be no other output than the triangle; the number of lines in the output should be the same as the input number. So in the example above, there should be precisely 5 lines of output.

Part 2

In the `part2` directory that was provided with this assignment there is a small binary file called `little_bin_file`. The file size is 17 bytes. You can check the size using the command "`ls -l`".

The file contains the following data, in this order:

- An integer number represented using the datatype `int`
- A floating-point number represented using the datatype `double`
- A character represented using the datatype `char`
- A floating-point number represented using the datatype `float`

In a C program we can use `sizeof` to determine the size of each datatype; you can verify that the total size for the four types above becomes 17 bytes by adding up the corresponding `sizeof` results.

Write a C program that opens the file, reads its contents into memory, and prints out the four pieces of data. Your program should not require any input arguments; it should assume that the input file is always called `little_bin_file`. The program should use the C library functions `fopen`, `fread`, and `fclose` to open the file, read it, and close the file when you are done with it. Use `printf` to output the results, printing each piece of data on a separate line in the same order as above, so that the program gives precisely 4 lines of output.

The name of the executable should be "`readfile`".

Part 3

Create a database storing maximum and minimum temperature for days in January. Let your program accept commands until the user stops the execution. The allowed commands are:

`A index min max` - save the minimum and the maximum temperature for a day with the given index; if that index already exists in the database, replace the data. The index is supposed to be an integer in the range 1 to 31.

`D index` - remove the day with the given index from the database.

`P` - print all data as a table with columns: day min max

`Q` - stops the execution

If the user enters an invalid command, print an error message and continue the execution. Assume that the index is an integer between 1 and 31, min and max are real numbers.

The name of the executable should be "`january`".

Example:

```
Enter command: A 1 -15.2 -5.1
Enter command: A 5 -1 1
Enter command: A 3 -11 -2
Enter command: D 2
Enter command: P
day    min         max
 1    -15.200000   -5.100000
 3    -11.000000   -2.000000
 5     -1.000000    1.000000
Enter command: A 6 -4 -2
Enter command: D 3
Enter command: A 11 -8 -5
Enter command: A 6 -1 0
Enter command: A 12 -5 -2.3
Enter command: P
day    min         max
 1    -15.200000   -5.100000
 5     -1.000000    1.000000
 6     -1.000000    0.000000
11     -8.000000   -5.000000
12     -5.000000   -2.300000
Enter command: Q
```

Create the database as a linked list. Let each node of your linked list contain a structure containing data for a given day. Keep your list sorted by index when inserting or deleting data from it. For more information on linked lists see for example here: http://www.learn-c.org/en/Linked_lists.

Make sure to test your code for different scenarios that may need to be handled differently in the code, for example adding something to the beginning of the list, to the middle of the list, and to the end of the list, as well as removing an the first/last entry or an entry somewhere in the middle of the list.

Note that it is important that your code manages memory properly by calling `malloc` and `free` in the correct way. For example, if all your database entries have been deleted, then your code should have freed all the memory that was previously allocated. The total number of `malloc` calls made should then be precisely the same as the total number of `free` calls made. If a program does not free allocated memory properly, that is called a *memory leak*. Your code should not have memory leaks.

Note: if you are using function `scanf` for reading input characters, then remember that the conversion specifier %c does not consume leading whitespaces. But it can be made to do so by using a whitespace character in the format string. Example:

```
scanf("%d", &a);
scanf(" %c", &c); // consume all consecutive whitespace after %d,
                  // then read a char
```

Preparing your submission

To make it easier for your teachers to check your submissions in a systematic way, the submission is required to have a specific form, as described below.

Create a directory called `A2` and put your final `part1`, `part2` and `part3` directories as subdirectories inside the `A2` directory.

No binary files such as object files or executable files should be included.

Then use the `tar` command to create a "tar-ball" package called `A2.tar.gz` containing the `A2` directory with all its contents, in the same way as you did for Assignment 1.

When you have created your `A2.tar.gz` file in that way, copy it somewhere else and unpack it there to check that it really has the contents you want.

Before submitting the file in Studium, use the `check-A2.sh` script that was included with the assignment instructions, to verify that your file follows the requested format. To use the `check-A2.sh` script, first set execute permission for it, and then run it when standing in the same directory where you have the `A2.tar.gz` file. If the file has the requested format, the output from the script should say "Congratulations, your A2.tar.gz file seems OK!". If you do not get that result, look carefully at the script output to figure out what went wrong, then fix the problem and try again.

Note that getting the "Congratulations" message from the script does not guarantee that your assignment is fully correct, since the script does not check everything, it just checks that the directories and files exist and are named as requested and performs a few simple tests of you programs. When you have submitted the file, your teachers will check your submission more carefully.

Submission

When you are done, upload your final `A2.tar.gz` file in Studium. Note that the uploaded file should have precisely that name, and that you should have checked it using the `check-A2.sh` script before uploading it.