CSE 102 Programming Assignment 2

DUE

October 27, 2024, 23:55

Description

- This is an individual assignment. Please do not collaborate.
- If you think that this document does not clearly describe the assignment, ask questions before its too late.

You won't be given a chance to correct any mistakes.

- Write a C code according to the following items:
  - Read `input.txt`
  - `input.txt` includes integers separated by whitespace (space, tab, newline etc...). The number of integers is not known. It can be 3 or 300000 or more. Don't make any assumptions about the number of integers in the file. (Read till the end-of-file. Don't try to store all the numbers in memory. You don't need to store all the numbers.)
  - Define a macro for the size of a segment. Let this macro be `X`. `X` is an integer value between 1 and 10.
  - Your program will count the repetitions of **different** segments in the input file. Count the `Y` most recently seen different sequences. `Y` is a macro defined at the beginning of the program and it can take values between 1 and 20.
  - Write the repetitions of the `Y` most recently seen different sequences to `output.txt`.

Example

- macros:

```
#define X 2 \* segment size is 2. A segment is defined by 2 consecutive integers. *\
#define Y 3 \* This means, you are going to count 3 different segments *\
```

- `input.txt`:
- (there will be actual numbers instead of these labels `int1, int2` etc...)

```
int1 int2 int3 int2
int2 int1 int2
```

- In this sequence we are going to count size 2 segments. Because `X` is defined to be `2`
- read 2 numbers: `int1 int2`. Check if this segment has been seen before. If it has been seen before, increment its count. If it is a never-seen-before segment, start counting.
- read another number from the sequence. The segment is now `int2 int3`. (we are basically sliding a window of size `X` one step at a time.). Count this segment and slide the window again. Repeat this process until you reach the end-of-file.
- You can only remember at most `Y` number of different segments and their counts. If you encounter a new segment, you have to drop the oldest segment you have been counting and start counting the newly seen segment instead. (You need to keep the remembered segments and their counts in the order they initially appear.)
- Suppose that you count segment1 segment2 and segment3. If you see a new segment(segment4), you will stop counting segment1. You program can remember segment2 segment3 and segment4. If segment1 re-appears, it is new segment. In this case, you have to drop segment2 and start counting segment3, segment4 and segment1.
- Once you reach the end-of-file, you store all the counted segments(the last ones you remember) and their counts to an output file. The name of the file is `output.txt`
- Be careful with the format and order of the data.
- segment: `<segment> := space separated integers in a particular segment`

- format of the output file. (each segment is printed to a new line.)

```
<segment> : <count>
<segment> : <count>
...
```

- `output.txt` after the execution(int2 or int1 etc will be replaced by actual integers, of course):

```
int2 int2 : 1
int2 int1 : 1
int1 int2 : 1
```

Remarks

- You can only create arrays of sizes in the order of `c*Y*X` or less. (`c` here is a small constant.) If you create bigger arrays you get `0.0`. Code efficiently. You may lose points if you create unnecessary arrays.
- Be careful with the name of the files (`input.txt` and `output.txt`). You won't be given a second chance if you make a mistake about this. Your program will fail and your grade will be `0.0`.
- Do not use any elements which is not covered in class. (You can use arrays)
- Do not submit your code without testing it with several different scenarios. Test with very big input files.
- Write comments in your code.
- You can use `ftell()`, `fseek()` and other useful functions for file read/write operations.
- There can be negative and positive integers.There can be `0`s in the file.
- You can assume that the file is error-free. (i.e. there are only integers in the file.)
- Properly check end-of-file and successful read/write operations.

Turn in:

- Source code of a complete C program. Name of the file should be in this format: `<full_name>_PA2.c`.
- Example: `david_hilbert_PA2.c`. Please do not use any Turkish special characters.
- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). GCC will be used.
- Make sure that your program does not require specific encodings/markings/line-ending-chars. Make sure it works with a file created in a linux environment.
- Make sure you don't get compile errors when you issue this command : `gcc <full_name>_PA2.c`.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Not accepted.

Grading (Tentative)

- `Max Grade : 100`.
- Multiple tests(at least `5`) will be performed.

All of the followings are possible deductions from `Max Grade`.

- use macros instead of hard-coded values, otherwise you may lose: `-10`.
- No submission: `-100`.
- Compile errors: `-100`.
- Irrelevant code: `-100`.
- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- inefficient implementation: `-20`.

- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values, avoid hard-to-follow expressions, avoid code repetition, avoid unnecessary loops).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-5`. (Comments are in English).
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`).
- Missing or wrong output values: `Fails the test`.
- Output format is wrong: `-30`.
- Infinite loop: `Fails the test`.
- Segmentation fault: `Fails the test`.
- Fails 5 or more random tests: `-100`.
- Fails the test: `deduction up to 20`.
- Prints anything extra: `-30`.
- Requires space/newline at the end of the file: `-20`.
- Requires specific newline marking (CR/LF): `-20`.
- Unwanted chars and spaces in output: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.