CSE 102 Programming Assignment 1

DUE

October 23, 2021, 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 which does the following (`90 pts`):
  - Read `file.txt`
  - Remove repeated integers
- At the beginning of your code, comment about the efficiency of your implementation(`10 pts`) Theoretically compare the running times of your implementation for different input sizes. For example: if `file.txt` includes 100 numbers, lets say your program completes the task in 1 second. What happens if `file.txt` includes 1000 numbers? Will it take 10 seconds, 100 seconds or more? Don't try to measure the exact running time of your program. Try to understand and comment about the time complexity of your algorithm.

Example

- `file.txt` before any operation:

```
12     -5           67      12 4      -888 23 5 7 67 2
56   3        67 6       5
   45       -5                3  12 7 1
```

- Your program finds the repeated integers and replaces their position with empty space
- `file.txt` after the execution:

```
12     -5           67         4      -888 23 5 7    2
56   3           6
   45                                 1
```

Remarks

- Be careful with the name of the file (`file.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. Do not use arrays.
- Do not submit your code without testing it with several different scenarios.
- Write comments in your code.
- You can use `ftell()`, `fseek()` and other useful functions for file read/write operations.
- **Preserve the spacing in the `file.txt`. The positions of the numbers in the file should not be altered.**
- There can be negative and positive integers.
- You can assume that the file is error-free. (i.e. there are only negative and positive integers in the file.)
- Properly check end-of-file and succesful read/write operations.
- Efficiency of your implementation is important. Comment about the efficiency of your code. (`10 pts`)

Turn in:

- Source code of a complete C program. Name of the file should be in this format: `<full_name>_<id>.c`.
- Example: `gokhan_kaya_000000.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>_<id>.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 lose 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`.

- HARD_CODED_VALUES `-10`. (if you don't use macros you may lose up to `10` pts)
- No submission: `-100`. (be consistent in doing this and your overall grade will converge to `N/A`) (To be specific: if you miss `3` assignments you'll get `N/A`)
- 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`.

File Read and Write

Line Endings Normally, this shouldn't be an issue for a programmer who does error checking while reading files. Windows and Unix have different conventions for marking the end-of-line in text files. Windows uses carriage return and line feed (`\r\n`) as a line ending. Unix uses just line feed (`\n`). If a text file is created in unix environment and if you try to open it in a windows environment you will see that a basic text editor cannot really recognize the new lines. Advanced text editors can recognize the line endings and properly show the text file. Many advanced text editors have an option about how to mark the line endings.

Many of the students do not properly check the data read from a text file and they basically tune their program to run with files which have line endings marked with either `\r\n` or `\n`. A similar thing happens with files which have an empty line at the end or no empty line at the end. Some submissions strictly require an empty line and some require the empty line to be removed. Almost all of these derivations are not design decisions of students. It happens just because student manages to create a working version for a particular case. And (s)he is unaware of it.

Almost all of the programming assignments will include file read and write operations. You are going to be responsible from all of the cases listed below:

- Your program can be tested with a text file which has line endings marked with `\r\n`.
- Your program can be tested with a text file which has line endings marked with `\n`.
- Your program can be tested with a text file which has an empty line at the end.
- Your program can be tested with a text file which does not have an empty line at the end.

If your program fails the tests because it's unable to handle any of the cases listed above, any arguments such as *it's working on my computer* will be ignored.

How to properly read and write to text file?

```
#include <stdio.h>                \* Include the library *\


int main () {
  FILE *fp;                 \* A Pointer which represents the file.*\

  fp  = fopen ("file.txt", "w+");  \*open for reading and writing, overwriting a file*\

  int status;
  int y;
  status = fscanf(fp,"%d",&y);
  if(status == 1)
  {
     /* process y */
  }
  fseek(fp, -numofDigit, SEEK_CUR); \* move cursor numofDigit times backward *\
  fputs(" ", fp); \* puts a space in the current position *\
  fclose(fp); \* close the file *\

  return 0;
}
```