SYSC 2006 Foundations of Imperative Programming

Assignment 2

Filenames must be exactly as shown for the submit program. Read and follow the instructions.

Email submissions will not be accepted by TAs or instructor.

Objective:

Write a program called **assign2.c** involving structs, two-dimensional arr

Post:

- test.txt and histogram.txt

Introduction

We will pretend that we are writing the "business logic" of the voting system we have been using in class (so I will assume I don't have to explain the system too much because you are familiar with it from class). We will not incorporate the reception of messages (either via Twitter, text or web) but will instead assume that the votes have already been received and are stored in a text input file. Our program will read that file. and produce and store the corresponding histograms in a separate text output file. Our program will be able to handle files that include the votes for more than one poll interspersed within each other. The format of the file is strictly defined. If you know the format, you can write a program to read that format.

Program Requirements

To help you understand the problem, you are provided with a sample input file (**test.txt**) and a sample of the expected corresponding output file (**histogram.txt**).

Some additional requirements (also known as a shall-list):

1. The program shall not have any output on the console, except in the case of an error condition. Error conditions must be printed to the console using stderr. . If you use printf() to debug, the extraneous outputs must be removed before submission.
2. No functions are needed and in fact, you are required not to use any functions (An odd request but one that will provide some fodder for a coming lesson).
3. The program must use command-line arguments to provide the name of the input voting file and the name of the output histogram file. Tip: Start with hard-coded names. Once it is working, then modify to use command-line arguments.
4. With minimal compilation, the program should be easily changed to handle different number of polls and options. (i.e. Use #define constants instead of hard-coded values)

Input and Output Format

The program will work for all files that follow the **SYSC 2006 Voting Format (S2VF)**. Compare the description of this format with the sample input file (test.txt) to gain a full understanding of the task ahead.

The file begins with a header that describes one or more polls that were conducted in the last voting session. Each poll is described by an integer *poll identifier* followed by one or more voting *options* (represented as integers) terminated by a 0. The poll identifier and the voting options all appear on a separate line. The header is followed in the file by the actual data. Each of the lines in the data section contain a single {pollID, vote} pair.

The required format of the output file is (hopefully) obvious from the given sample (**histogram.txt**). Before programming, as a final test of your understanding, verify that **histogram.txt** correctly describes the data in **test.txt**.

Instructions

1. The first step in writing a program is to plan out the key data structures. You may devise your own solution or you may use the screenshots at the end of this document to understand a possible solution. Figure out the "data flow" in and/or out of each variable. Summarize (in a one-line comment beside each variable) the role of each variable? This analysis will lead you to the code you need to write.

   Draw this plan by hand on a sheet of paper. You will submit a snapshot of this drawing with your assignment.

2. It is suggested that you now write your program in incremental steps. After each step, compile and run (the execution may not do much at first, but at least you know it does not hang or crash)
   a. Declare your variables and initialize them where appropriate.
   b. Do the easy part first. Write out the histogram to a file. Again, the histogram will contain all zeros, but at least you know you have written a file.
   c. Now do the hard part. Read in the data from the test file.
   d. Now do any remaining work to link the input data to the output data.

Testing Requirements

As well as making your code run on the given input file, you must also test it on an input file of your own making (to be called **mytest.txt**). Additionally, your code will be tested by another version of the input file, secret but following the same format. In sum, your code will be tested against three input files: test1.txt, mytest.txt and TAtest.txt

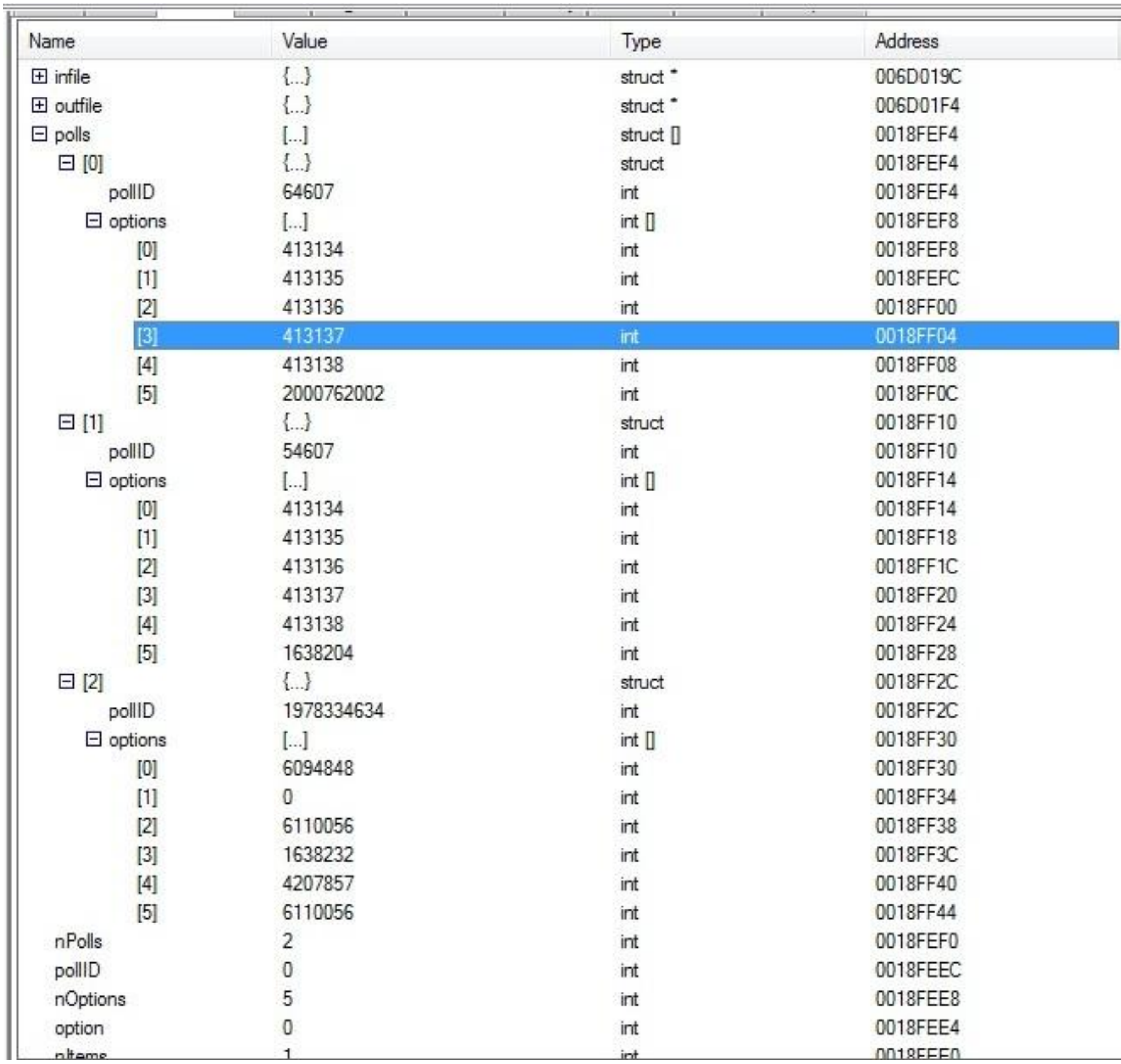
If cheating continues: Students will be required to submit a you-tube video (less than 1 minutes) with your face visible at >1 time and your voice explaining some aspect of your code, running your code in the debugger or explaining the input format or output file.

Submission

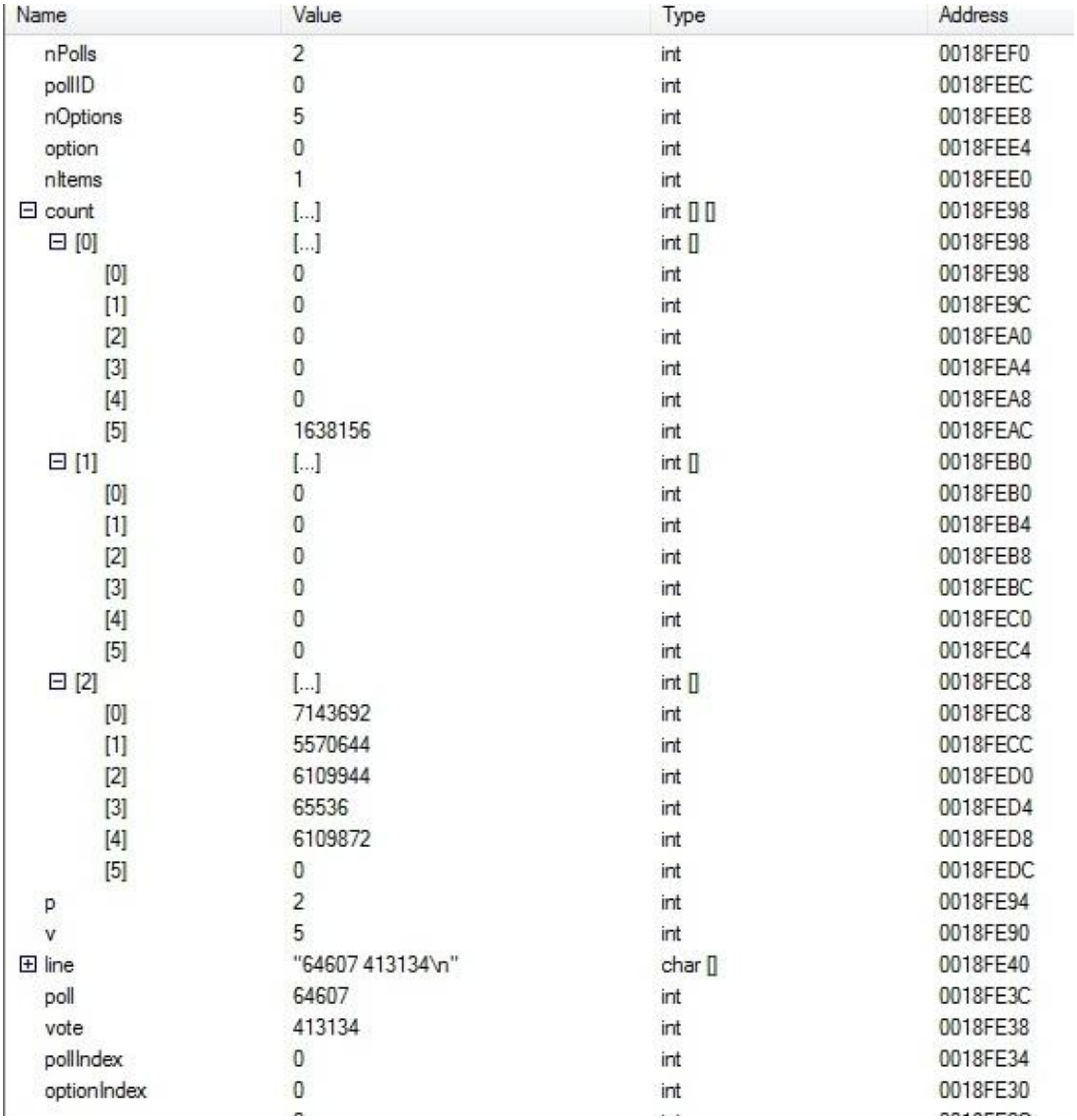
The following files shall be submitted through the SUBMIT program that is available for download from a link on the Course Resources page, before the deadline.

assign2.c: The program
assign2data.jpg: A drawing in your own handwriting of the main data structures of the Pelles screenshots. Take a picture of it with your camera or phone, and save as a Web-small (448x336) format (for example, using Microsoft Picture Manager, Export).
histogram.txt: The output of your program run with test.txt
mytest.txt: Your version of an input file, with different number of polls, different poll identifiers and different options and votes.
myhistogram.txt: The output of your program run with mytest.txt

Sample Data Structures



Name	Value	Type	Address
⊞ infile	{...}	struct *	006D019C
⊞ outfile	{...}	struct *	006D01F4
⊟ polls	[...]	struct []	0018FEF4
⊟ [0]	{...}	struct	0018FEF4
pollID	64607	int	0018FEF4
⊟ options	[...]	int []	0018FEF8
[0]	413134	int	0018FEF8
[1]	413135	int	0018FEFC
[2]	413136	int	0018FF00
[3]	413137	int	0018FF04
[4]	413138	int	0018FF08
[5]	2000762002	int	0018FF0C
⊟ [1]	{...}	struct	0018FF10
pollID	54607	int	0018FF10
⊟ options	[...]	int []	0018FF14
[0]	413134	int	0018FF14
[1]	413135	int	0018FF18
[2]	413136	int	0018FF1C
[3]	413137	int	0018FF20
[4]	413138	int	0018FF24
[5]	1638204	int	0018FF28
⊟ [2]	{...}	struct	0018FF2C
pollID	1978334634	int	0018FF2C
⊟ options	[...]	int []	0018FF30
[0]	6094848	int	0018FF30
[1]	0	int	0018FF34
[2]	6110056	int	0018FF38
[3]	1638232	int	0018FF3C
[4]	4207857	int	0018FF40
[5]	6110056	int	0018FF44
nPolls	2	int	0018FEF0
pollID	0	int	0018FEEC
nOptions	5	int	0018FEE8
option	0	int	0018FEE4
nItems	1	int	0018FEE0



Name	Value	Type	Address
nPolls	2	int	0018FEF0
pollID	0	int	0018FEEC
nOptions	5	int	0018FEE8
option	0	int	0018FEE4
nItems	1	int	0018FEE0
⊟ count	[...]	int [] []	0018FE98
⊟ [0]	[...]	int []	0018FE98
[0]	0	int	0018FE98
[1]	0	int	0018FE9C
[2]	0	int	0018FEA0
[3]	0	int	0018FEA4
[4]	0	int	0018FEA8
[5]	1638156	int	0018FEAC
⊟ [1]	[...]	int []	0018FEB0
[0]	0	int	0018FEB0
[1]	0	int	0018FEB4
[2]	0	int	0018FEB8
[3]	0	int	0018FEBC
[4]	0	int	0018FEC0
[5]	0	int	0018FEC4
⊟ [2]	[...]	int []	0018FEC8
[0]	7143692	int	0018FEC8
[1]	5570644	int	0018FECC
[2]	6109944	int	0018FED0
[3]	65536	int	0018FED4
[4]	6109872	int	0018FED8
[5]	0	int	0018FEDC
p	2	int	0018FE94
v	5	int	0018FE90
⊞ line	"64607 413134\n"	char []	0018FE40
poll	64607	int	0018FE3C
vote	413134	int	0018FE38
pollIndex	0	int	0018FE34
optionIndex	0	int	0018FE30