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.

SYSC 2006 Foundations of Imperative Programming

Assignment 3

Objective:

Write a modular program using dynamically allocated compound data structures.

Post:

- Nothing

Introduction

You will write the game **MineSweeper**. All instructions for the game are to be found on Wikipedia. You are required to implement the **first three** paragraphs of the game's **Overview**. If you are unsure about what is "allowed" or what is "correct", **anything goes as long as it coincides and does not contradict this overview.**

Program Requirements

The program shall meet the following specifications:

1. The program shall use command-line arguments to configure the execution of the game. Further instructions are given below.
2. All major data structures are to be dynamically allocated.
3. The user interface will be text-driven (not graphical), meaning that the user will type in his/her move and the grid shall be printed to the console.
4. The program shall be structured, using three files: **assign3**.c, **minesweeper**.h and **minesweeper**.c

Testing and Command-Line Arguments

In the game, mines are randomly located. To facilitate testing, you should be able to load in a test file that fixes the mine locations so that you can focus on particular areas of your code. Command line options will allow both **game mode** and **test mode**.

The first command-line option will tell whether to play in test mode ("-t") or game mode ("-g").

In either mode, two more options **<row>** and **<column>** are used to give the dimensions of the grid.

In test mode, two additional options <infile> and <outfile> will be used to provide the names of a file from which to read the initial mine locations and a file to which to log (i.e. print) the entire game execution as seen by the user, respectively. Caution: The row and column options must match the dimensions of the data provided in <infile> (Do not test for it, just make sure you do it).

e.g. Game mode: minesweeper –g 100 100

e.g. Test mode: minesweeper –t 10 10 initial.txt log.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. Play the game out on paper, to identify the key trouble spots.

   Finding the neighbours of a cell is going to be one of your issues. One way is to categorize the cells according to the number of neighbours that it has: those in the middle (with 8 neighbours, including those diagonally), those on the left vertical (with neighbours to the right, above and below), those on the right vertical (with neighbours on the left, above and below), similarly for the top and bottom, and finally the four corners. Draw a picture to understand and relate "neighbour" to some simple arithmetic on <row, column> indices.

2. If you are writing the assignment before we have fully covered dynamic allocation (or you don't yet understand them), have no fear! The entire program can be written with statically allocated arrays, and then at the end, simply converted to dynamically allocated arrays.
3. 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. Make sure to dynamically allocate any/all major data structures.
b. Do the easy part first. Read in a fixed set of mine locations from a sample <infile>. Make the grid small, say 10x10 or even 8x8, so that you can try things out but not get overwhelmed. Display the grid (with mines shown) and then display the grid (with mines hidden).
c. Develop the user interface by which the user will input their moves. To begin, just write the prompt-and-input loop and simply print out the kind of move requested.
d. Now, start to actually process the move. Forget about neighbours for now. Just reveal the requested cell.
e. Now add in neighbours.
f. Now add in game logic. When does the game quit? When does the user lose or win?

Submission

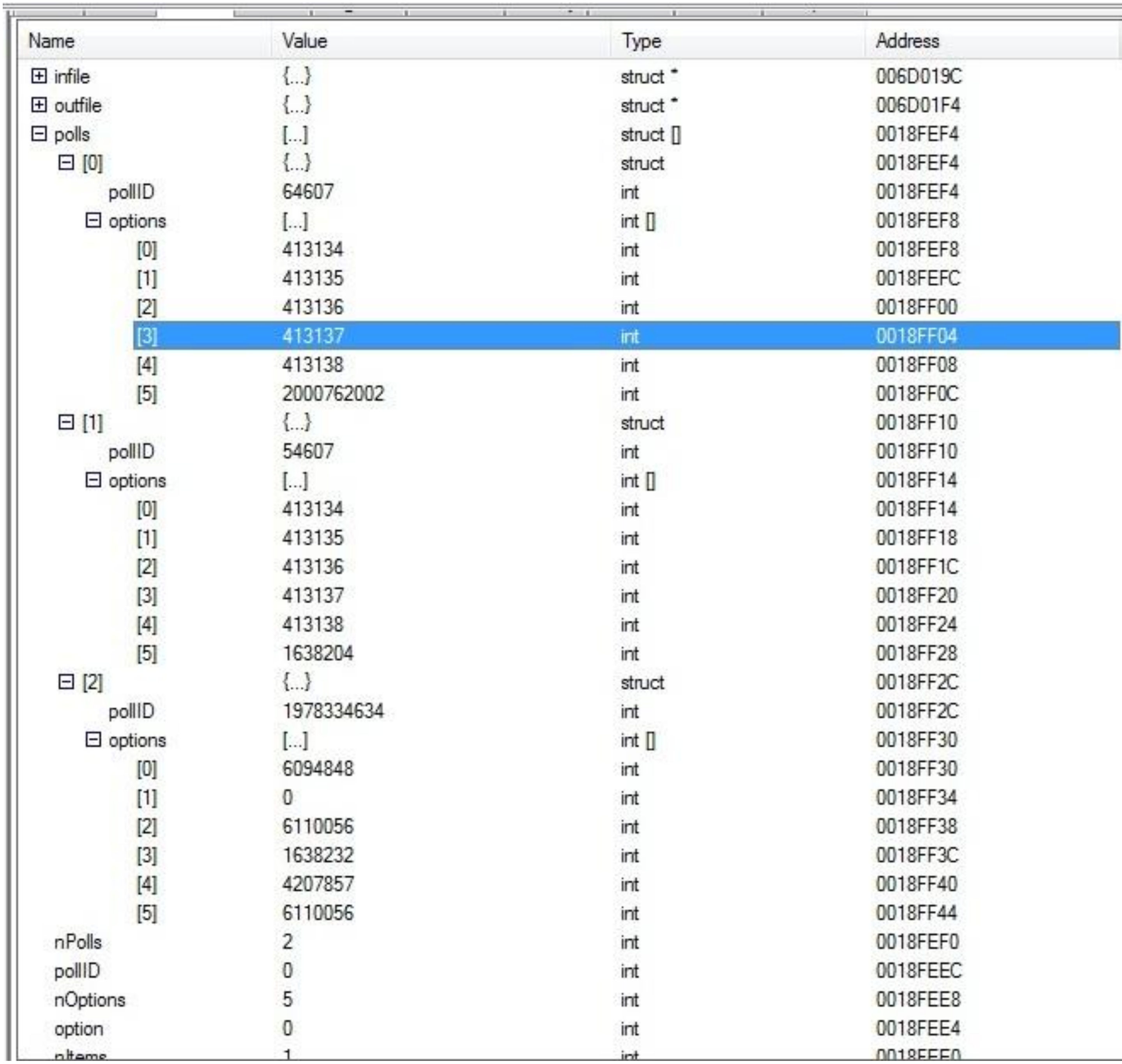
The following files shall be submitted through the SUBMIT program that is available for download from a link on the Course Resources page, before the deadline.

assign3.c: The program
assign3data.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).
infile.txt: A sample test layout of mines.
outfile.txt: The log from running your sample test layout with your program.
minesweeper.c
minesweeper.h

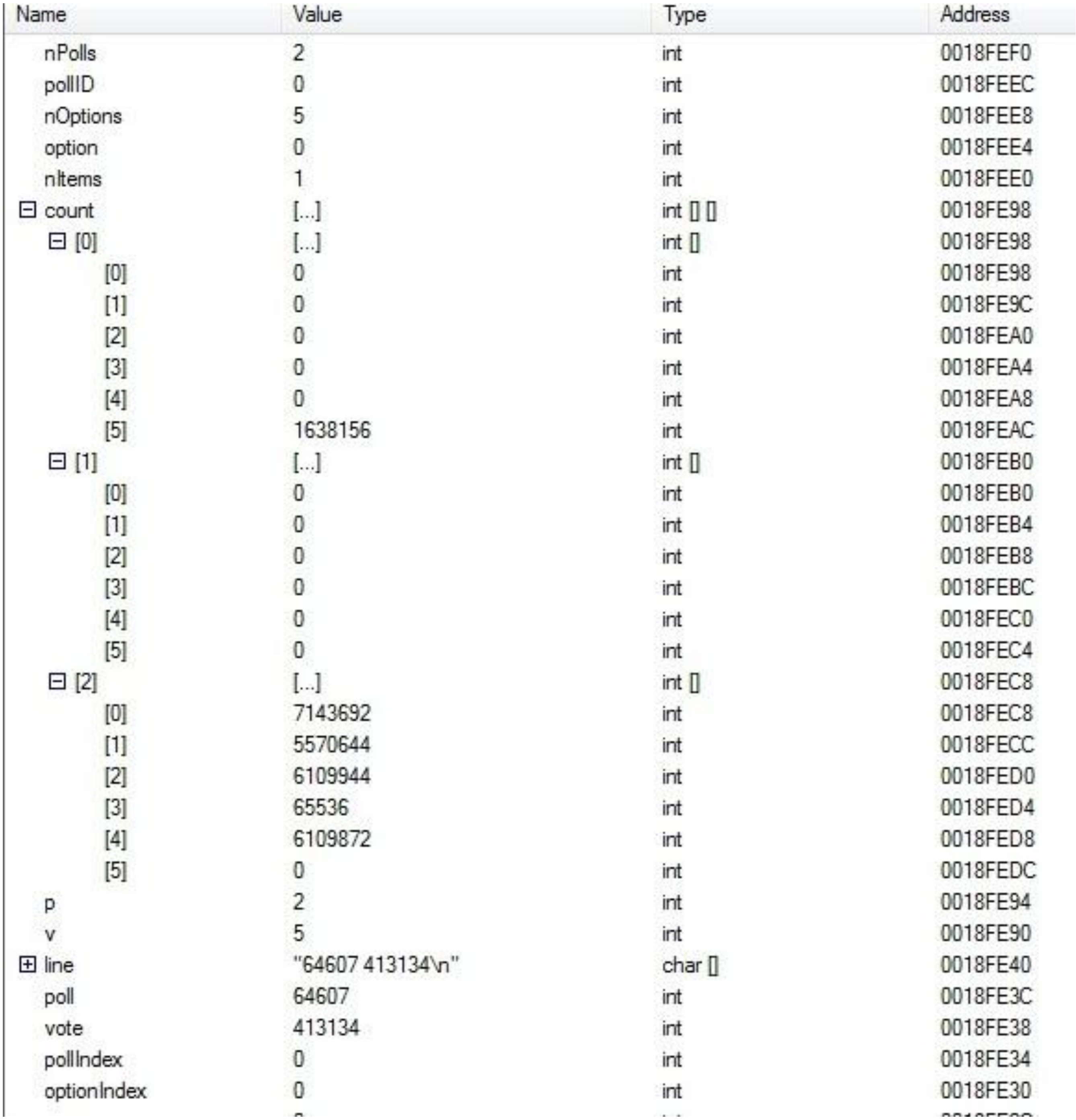
Marks will only be assigned if the contents of outfile.txt matches your program. (Don’t submit someone else’s).

If you do not get the whole program working, you still get marks! Please just put an informative comment at the top of assign3.c to help the TA know what is complete and what is not.

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