CST8234- C Language

Assignment 1

Problem Statement:

In this assignment, we will look at a way to develop a students' registration system using C for some college. The goal of this program is to allow the administrators to register a student in offered courses, and also allows them to drop a course for a student.

This assignment is intended to get you to practice using arrays, pointers, functions, header files, user inputs, and formatting output among other things.

Background Information:

In this assignment we will have two arrays, one to store the student IDs, and the second is to store the course codes.

In addition, there will be a registration table, represented by two-dimensional array, that will store the courses each student is registered in. This will be presented as a simple yes/no value stored in the registration table for each student using their index in the students array, and the course index in the courses array.

For example, if the system has three students with IDs {12345, 34567, 56789}, and have two courses with codes {"CST8234", "CST8288"}. Then the administrator can register a student with ID "34567" in a course with code "CST8234" by recording "Yes" in the registration table for index 1 (representing the student index) and 0 (representing the course index). The following tables illustrate the memory representation for of each of the arrays.

Students		
Address	Index	Value
0x303300	0	12345
0x303304	1	34567
0x303308	2	56789

Courses		
Address	Index	Value
0x308800	0	CST8234
0x308809	1	CST8288

Registration Table			
Index	Student Index	Course Index	Value 0 = no / 1 = yes
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1
4	2	0	1
5	2	1	0

Therefore, from the above table we can see that the student with index 0 (ID: 12345) is registered in course with index 1 (Code: CST8288) only.

Requirements:

Write a program that achieves the following requirements:

1. The program should prompt the user to enter:
    1. The number of students they wish to register.
    2. The student ID for each student. Student IDs are 5-digit integers, i.e. 45234.
2. Students should be stored in their own array.
3. The program then should prompt the user to enter:
    1. The number of courses offered.
    2. The course code of each of these courses. Course codes are 7-digit alpha-numeric strings, i.e. CST8234.
4. Courses should be stored in their own array.
5. The program then should prompt the user to choose one of three actions:
    1. Register a student into a course.
    2. Drop a student from a course
    3. Display Registration table.
6. If the user chooses to register a student or drop a student, then the program should prompt for the student ID first then the course code second and perform the correct action as follows:
    1. Validate the student ID as entered by the user. Display an appropriate error message if the student ID is not found and re-prompt the user for a valid student ID.
    2. Validate the course code as entered by the user. Display an appropriate error message if the course code is not found and re-prompt the user for a valid course code.
    3. Registering a valid student will update the registration table by adding “1” (i.e. true) to the element with the [student index][course index] element.
    4. Dropping an existing course will update the registration table by adding “0” (i.e. false) to the element with the [student index][course index] element.
7. If the user chooses to display the registration table, the program should print all entries in the table.
8. Additional requirements:
    1. Add another action to the menu that will allow the user to quit the program by choosing the quit action.
    2. The program would loop until the user quits the program. The program will loop only starting from point 5 above, so the user doesn’t need to enter the students or the courses information anymore.
    3. Validate the menu action as entered by the user. Display an appropriate error message if the menu action is not found and re-prompt for a valid menu action.

Design Requirements:

1 Make sure you use functions, design your program to separate functionality into its own functions. Using only **`main`** function will make you lose points.
2 Give your functions and variables a descriptive name. For example, `students[]`, not `x[].`

Supporting Files:

Along with this document, you will find two files hosted on Brightspace. The files are named **helper.h**and **helper.c.**These files contain generic functions that can be used to help on finishing this assignment. Please read the comments above each of these functions to understand the intent and the usage example. If you are going to use these functions, you will need to include **helper.h** in you program.

Documentation Requirements:

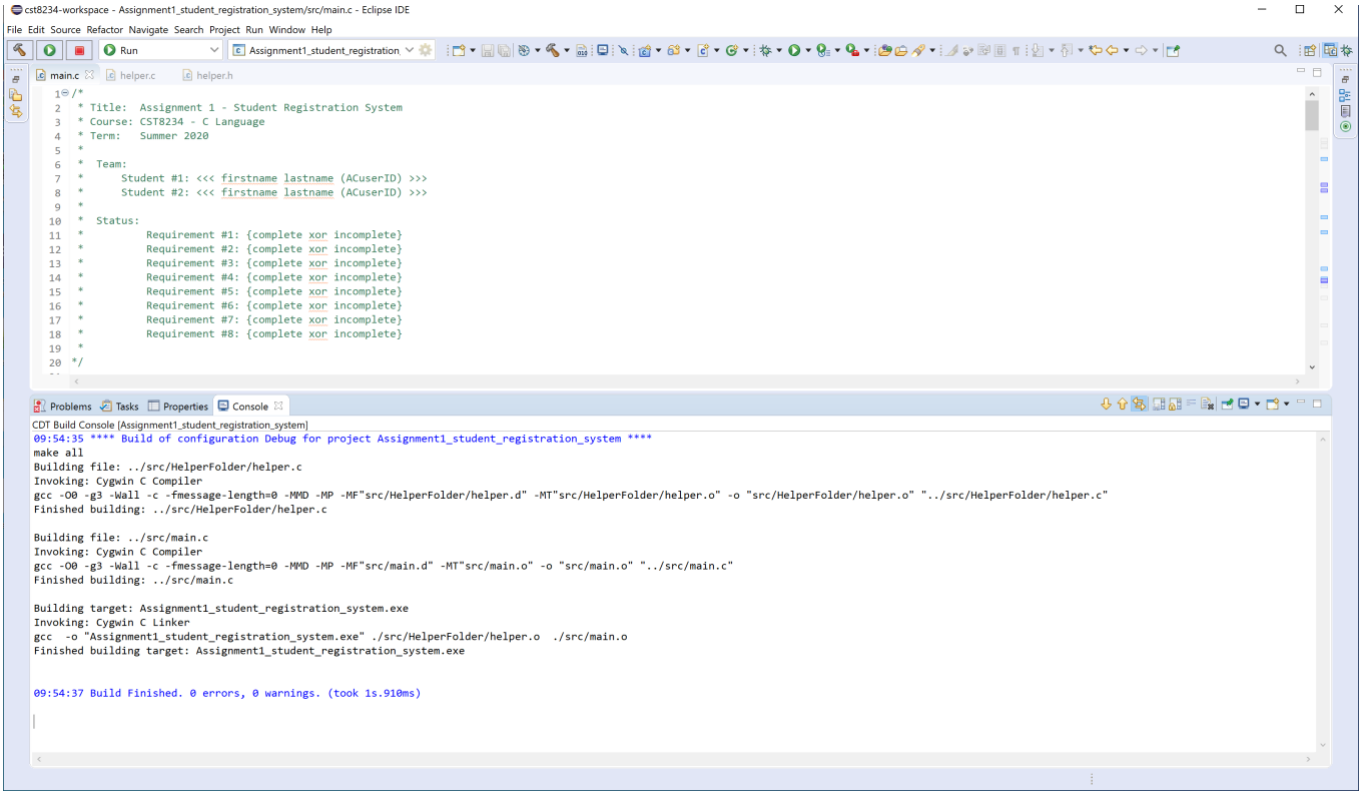
Document your team's solution by adding a header comment to the start of your C source file that has the main( ) function:

```
/*
 * Title:  Assignment 1 - Student Registration System
 * Course: CST8234 - C Language
 * Term:   Summer 2020
 *
 *    Team:
 *          Student #1: <<< firstname lastname (ACuserID) >>>
 *          Student #2: <<< firstname lastname (ACuserID) >>>
 *
 *    Status:
 *                Requirement #1: {complete xor incomplete}
 *                Requirement #2: {complete xor incomplete}
 *                Requirement #3: {complete xor incomplete}
 *                Requirement #4: {complete xor incomplete}
 *                Requirement #5: {complete xor incomplete}
 *                Requirement #6: {complete xor incomplete}
 *                Requirement #7: {complete xor incomplete}
 *                Requirement #8: {complete xor incomplete}
 *
*/
```

Reference Screenshot #1:

Your team's project must compile without warnings and without errors.

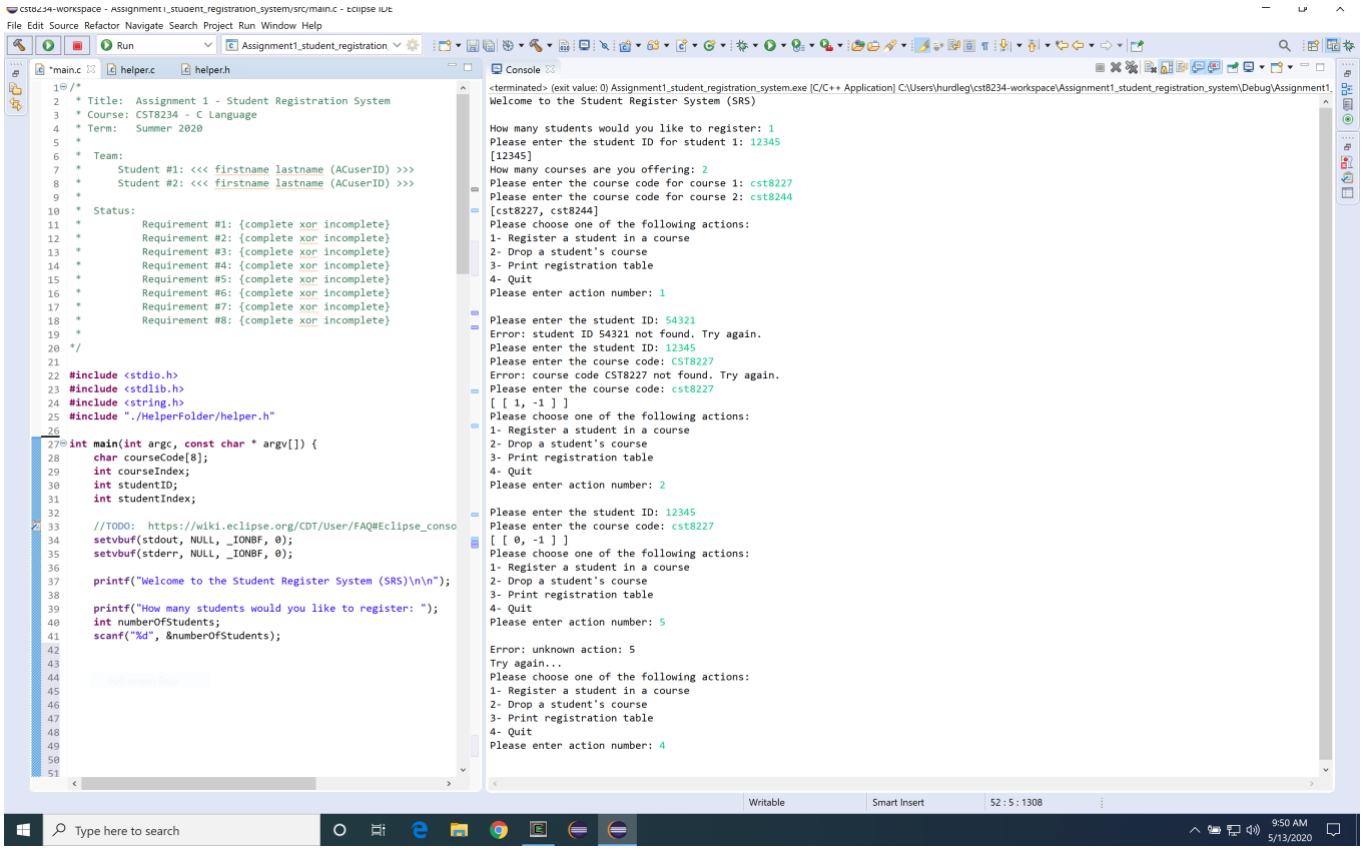
Compare your screenshot to the reference screenshot:



Reference Screenshot #2:

Demonstrate your team's solution implements the (functional) requirements.

Compare your screenshot to the reference screenshot:



Typescript Deliverable

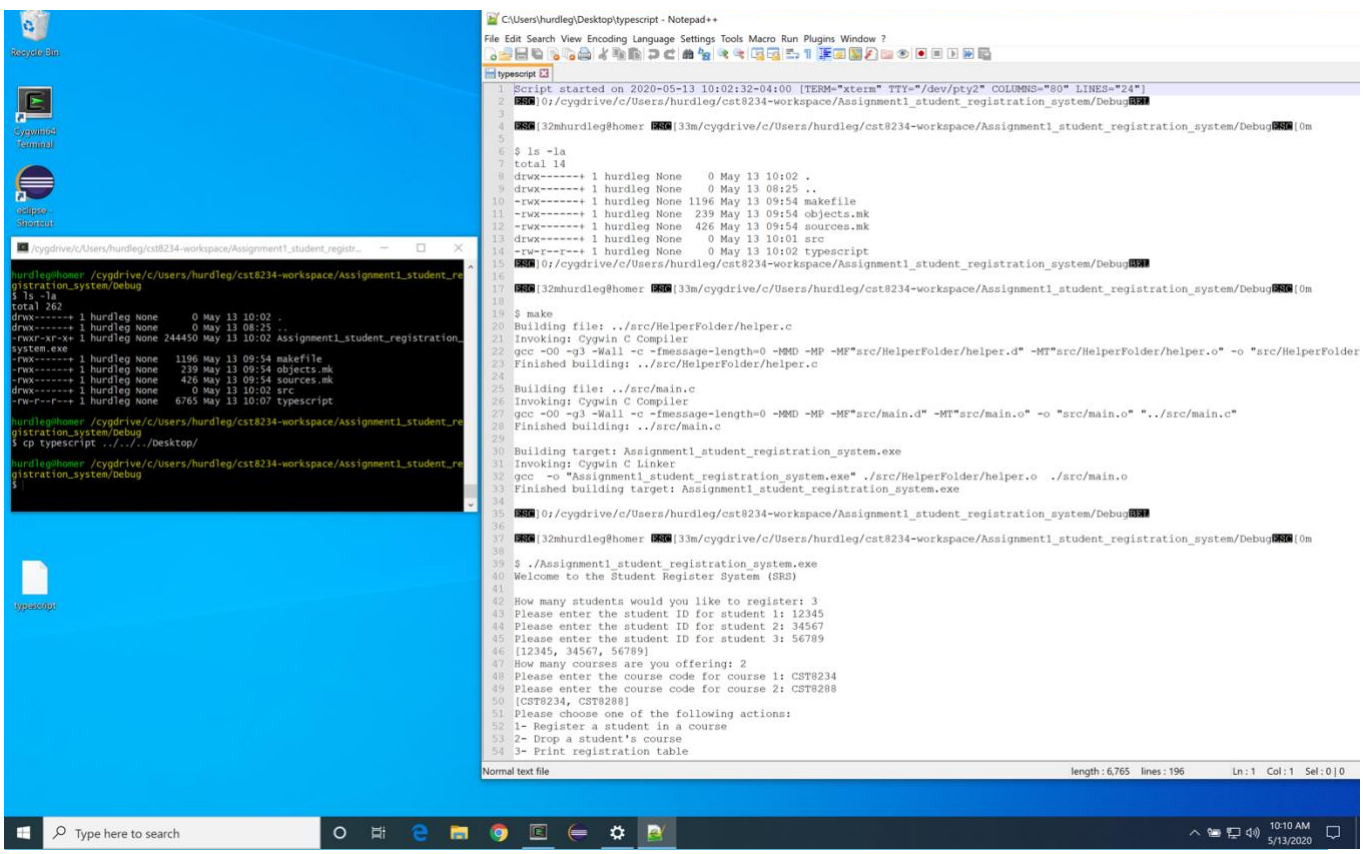
Your team will prepare a typescript file as the acceptance test for this assignment. Be sure to compare your results with the reference typescript file hosted on Brightspace.

To make a typescript file, open a Cygwin terminal and type the command: script
From this point on, everything that you type will be recorded and written to a file named (by default): typescript
To stop recording, enter the command: exit

For details, man script ☺

```
SCRIPT(1)                                User Commands                                SCRIPT(1)

NAME
       script - make typescript of terminal session

SYNOPSIS
       script [options] [file]

DESCRIPTION
       script  makes  a typescript of everything displayed on your terminal.  It is useful for students who need a hardcopy record of an interactive session as proof of an assignment, as the typescript file can be printed out later
       with lpr(1).

       If the argument file is given, script saves the dialogue in this file.  If no filename is given, the dialogue is saved in the file typescript.

OPTIONS
       Below, the size argument may be followed by the multiplicative suffixes KiB (=1024), MiB (=1024*1024), and so on for GiB, TiB, PiB, EiB, ZiB and YiB (the "iB" is optional, e.g. "K" has the same meaning as "KiB"), or the suf-
       fixes KB (=1000), MB (=1000*1000), and so on for GB, TB, PB, EB, ZB and YB.

       -a, --append
              Append the output to file or to typescript, retaining the prior contents.

       -c, --command command
              Run the command rather than an interactive shell.  This makes it easy for a script to capture the output of a program that behaves differently when its stdout is not a tty.

       -e, --return
              Return the exit code of the child process.  Uses the same format as bash termination on signal termination exit code is 128+n.  The exit code of the child process is always stored in type script file too.

       -f, --flush
              Flush output after each write.  This is nice for telecooperation: one person does `mkfifo foo; script -f foo', and another can supervise real-time what is being done using `cat foo'.

       --force
              Allow the default output destination, i.e. the typescript file, to be a hard or symbolic link.  The command will follow a symbolic link.

       -o, --output-limit size
              Limit the size of the typescript and timing files to size and stop the child process after this size is exceeded.  The calculated file size does not include the start and done messages that the script command prepends
              and appends to the child process output.  Due to buffering, the resulting output file might be larger than the specified value.

       -q, --quiet
              Be quiet (do not write start and done messages to standard output).

       -t[file], --timing[=file]
              Output timing data to standard error, or to file when given.  This data contains two fields, separated by a space.  The first field indicates how much time elapsed since the previous output.  The  second  field  indi-
              cates how many characters were output this time.  This information can be used to replay typescripts with realistic typing and output delays.

       -v, --version
              Display version information and exit.

       -h, --help
              Display help text and exit.

NOTES
       The script ends when the forked shell exits (a control-D for the Bourne shell (sh(1)), and exit, logout or control-d (if ignoreeof is not set) for the C-shell, csh(1)).

       Certain interactive commands, such as vi(1), create garbage in the typescript file.  script works best with commands that do not manipulate the screen, the results are meant to emulate a hardcopy terminal.

       It  is  not  recommended  to run script in non-interactive shells.  The inner shell of script is always interactive, and this could lead to unexpected results.  If you use script in the shell initialization file, you have to
       avoid entering an infinite loop.  You can use for example the .profile file, which is read by login shells only:

              if test -t 0 ; then
                  script
                  exit
              fi

       You should also avoid use of script in command pipes, as script can read more input than you would expect.

ENVIRONMENT
       The following environment variable is utilized by script:

       SHELL  If the variable SHELL exists, the shell forked by script will be that shell.  If SHELL is not set, the Bourne shell is assumed.  (Most shells set this variable automatically).

SEE ALSO
       csh(1) (for the history mechanism), scriptreplay(1)

HISTORY
 Manual page script(1) line 1 (press h for help or q to quit)
```

The typescript file can be viewed with a text editor. Important: you can look (read) but don't change (write) the typescript file. Editing the typescript file could be interpreted as falsifying the acceptance test. Please don't change the typescript.



Submission Instructions:

1. This assignment is to be completed in teams of size two (2). That is, you and a partner will collaborate on this assignment. Your partner can be in a different lab section. Please choose your partner wisely as squabbles will be solved by dividing the mark in half (i.e. ½). Individual submissions will not be accepted.
2. From eclipse IDE, export your Assignment 1 project as an archive file. Refer to eclipse documentation for details on how to export:
   https://help.eclipse.org/202003/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Ftasks%2Ftasks-59ag.htm
3. Make a zip-file that contains the following items:
   - eclipse IDE archive file
   - screenshot #1: clean compilation
   - screenshot #2: acceptance test
   - typescript file
4. Name your zip-file to include your AC userID and your partner's AC userID. Follow this format: cst8234_assignment_1_yourACUserID_yourPartner'sACUserID. For example: cst8234_assignemnt_1_bond0007_jaws0001.zip
5. Partner #1: upload and submit the zip-file to Brightspace before the due date.
6. Partner #2: ditto