Lab1_2 C Tutorial

Due Sep 18 at 11:59pm **Points** 100 **Questions** 10
Available Aug 29 at 12am - Sep 18 at 11:59pm 21 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

You have until the end-of-the-day on September 18th to complete the C Tutorial. It has 10 tasks but only Task 10 will be graded.

As you complete tasks 1 - 9 it is highly recommended that you save the c_tut project in Canvas for your reference. Please make sure to zip the whole project, that way you can easily continue from where you left off.

Please follow the conventions described in the "About Lab Assignments" tab.

This quiz was locked Sep 18 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	29,255 minutes	100 out of 100

Correct answers are hidden.

Score for this attempt: **100** out of 100
Submitted Sep 18 at 11:58pm
This attempt took 29,255 minutes.

Question 1 0 / 0 pts

Create a folder `comp232` on your computer desktop. Download C tutorial for Java Programmers (https://cilearn.csuci.edu/files/557275/download?wrap=1)

and the file with the supporting source code (https://cilearn.csuci.edu/files/498998/download?wrap=1) and unpack them to the directory. All source C files are in the `src` sub-directory. The file `CMakeLists.txt` in the root directory contains instructions needed by CLion to compile the code in the tutorial.

Open CLion, and then select `Open File` from the `File` menu and navigate to the root folder of the tutorial `c_tut`. You will see the file `CMakeLists.txt` in one editor pane, and the file `hello.c` in another. You can double-click on any file in the left project pane to open it.

To ensure that CLion project adapts to your computer environment, select the `Reload CMake Project` item from the `File` menu.

The configurations to run various files in the tutorial are predefined; they are available in the drop down menu in the top bar of CLion. To run one of the programs, select the corresponding configuration, and then click the `Run` (a green triangle) or `Debug` (green "bug") icon in the top bar, or select the desired action item in the top `Run` menu. To practice the routine, select `hello` from the drop-down configuration menu and run the `hello` program.

The configurations instruct CLion to re-build the executables before running. That can also be done manually by selecting `Clean` item from the `Run` menu, and then selecting `Build` item from the same menu (or clicking the "hammer" icon).

Examine the code of the program and verify that the output reflects the program's logic.

[BrianSkinnerLab1Pat1.zip (https://cilearn.csuci.edu/files/559002/download)](https://cilearn.csuci.edu/files/559002/download)

Unanswered

Question 2

0 / 0 pts

Modify the code in `types.c`, so that the program asks also for the number of faculty, and then prints the student-to-faculty ratio for CSUCI. You will need to pay attention to the types of variables and use proper format string in `printf()`, so the precision is limited to one position after the dot.

You may use man pages to get help on C functions and on system calls. You many want to explore:

```
$ man printf
$ man scanf
```

You may also get the same help inside CLion by positioning the cursor on a function and then using `F1` key.

Unanswered

Question 3

0 / 0 pts

Modify the code in `if-then-else.c`, so the value of `level` is read from the standard input as a character. Replace the `if/else` statement with a `switch` on the input character in lower case: only e, h, or f are valid cases; print appropriate message if the tank is **e**(mpty); **h**(half full); or **f**(ull). default case should display an error message.

You may use man pages to get help on C functions and on system calls. You many want to explore:

```
$ man puts
$ man getchar
$ man tolower
```

To use function `tolower` you must include `ctype.h` file

You may also get the same help inside CLion by positioning the cursor on a function and then using `F1` key.

Unanswered

Question 4

0 / 0 pts

Modify `array.c` so that the program reads a number of doubles from the standard input and computes their average.

The number of the doubles to process should be read from the standard input. It will be followed by the corresponding number of doubles. For example:

```
Provide a desired number of doubles?
5

Provide the doubles?
1.0 2.0
3.0
4.0 5.0
```

Note that `scanf()` will search for doubles in the standard input independent of the placement input line.

A maximum number of doubles that the program can handle is defined by a define `MAX_NUM_OF_CELLS`. Your program should create an array with that many elements, but then verify that the number entered by the user is no larger than the limit. If it is, then a warning message should be printed, and the number should be assumed to be the said `MAX_NUM_OF_CELLS`.

The program should print all the numbers in one line separating them by spaces, and follow with the second line with the average. For example, the following could be one output:

```
Data: 1.0 2.0 3.0 4.0 5.0
Average: 3.0
```

Unanswered

Question 5

0 / 0 pts

Examine the code in `strings.c`, `pointer.c`, and `mem-alloc.c`. Then, set breakpoints at the beginning of the `main()` functions, and execute the programs. When a program stops at a breakpoint, continue the execution by stepping through the instructions. At each step, examine the current values of the variables and make sure that you understand what is going on. Ask the instructor for explanation if you do not.

Using `strings.c`, `pointer.c`, and `mem-alloc.c` as prototypes, implement a new program in file `words.c`. To create a new file in CLion in the src directory of your project

1. right-click on the src directory in the list in the left project pane,
2. select `New` from the menu, and then `C/C++Source File`; this will open a new file dialog,
3. in the new file dialog, provide the name for the file by typing `words` (without the extension!) in the `Name` box,
4. select `.c` from the `Type` drop-down menu, and
5. uncheck the `Add to targets` checkbox.

After adding the new file to the `src` directory of the project you need to create a target for your program by following these steps:

1. edit `CMakeLists.txt` and add the following line at the end:

   ```
   add_executable(words src/words.c)
   ```

2. select `Reload CMake Project` from the `File` menu to create a run configuration for your program,
3. select `Edit Configurations...` from the `Run` menu, find `words` in the list, and then select `words` in the drop-down menu for the `Executable` entry field.

The new program should:

- read words from the standard input in a loop until a word `END` is read,
- count all read words (excluding the sentinel),
- store the words in an array of strings, and
- print the number of read words in the first line followed by the whole array with one word per line. For example:

```
The following 3 words have been read:
veni
vidi
vici
```

You should create an array of pointers to strings that will hold pointers to the read words. Set the size of the array to an arbitrary number of the words that the program can process; that number should be defined with `#define`. To start with, the array will not have any valid pointers, so you must allocate space to hold read words with `malloc()`. Each word will have a different length, so you must use a buffer (an array of characters) that will hold the line that is read from the input with `scanf()`, and then use `strlen()` to find out how long is the word in the buffer. With that length, you will know for how large storage you should ask in `malloc()`, so that it can hold the word. Keep in mind that `strlen()` returns the number of characters in the buffer excluding the end-of-string marker (`'\0'`); therefore, you must request one character more from `malloc()`. Subsequently, use `strcpy()` to copy the new word from the buffer into the newly allocated space and set the current element of the words array to point to that space.

You many want to explore:

```
$ man scanf
$ man malloc
$ man strlen
$ man strcpy
$ man strcmp
```

Unanswered

Question 6

0 / 0 pts

In this task, you will modify `file.c` program.

First, create a file call `data.txt` with the following content:

```
Hello there!
```

in the sub-directory `cmake-build-debug` by right-clicking on the name of the sub-directory in the left-side project pane, left-clicking on `New` from the pop-up menu, and finally selecting `File` from the secondary pop-up menu. Enter the file name, and then the file content in the editor pane. Remember to save the file by clicking on the `Save` icon or selecting `Save All` from the `File` menu.

Extend the code in `file.c`, so that the program reads the data from the `data.txt` file and writes it to a new file. The name of the new file should be requested from the user.

Please note that the executables in our configuration of CLion are created in the sub-directory `cmake-build-debug`, so by default any output file will also be generated in there.

You can overwrite the default location of input and output files. For example, to use a file in the `src` directory, prefix the file name with `../src/`.

You many want to explore:

```
$ man fgetc
$ man fputc
$ man fopen
$ man fclose
$ man scanf
```

Unanswered

Question 7

0 / 0 pts

`ptr-arg-2.c` contains an implementation of a function `swapIntegers()` that swaps values of two integers using their pointers. Add a function `swapStrings()` that will swap two strings. The new code should include:

- a declaration of the new function,
- declarations of two strings initialized to some arbitrary values, and
- the definition of the function that swaps the strings.

Please do not use `strcpy()`; rather, you should just swap the pointers.

Things start to get tricky here, so using the debugger is indispensable.

Unanswered

Question 8

0 / 0 pts

Starting with the code in `ptr-func.c`, create a file `calc.c` that implements a simple calculator utility that performs four basic operations: `+ - / *` on two operands. The program should prompt the user for the operation to perform in an endless loop. For example:

```
calc> 3 + 6
9
calc>
```

You must implement the calculator in such a way that there is one function `calc(...)` that takes a pointer to the actual function that performs the specified operation (`add()` `sub()` `mult()` `div()`), and two operands.

Unanswered

Question 9

0 / 0 pts

Using `struct.c`, as a starting point, implement an application that creates a database of employee personal records. Your implementation should follow these guidelines:

- the definition of the structure `PERSON` should be provided in an h-file called `person.h` that you need to create in the `src` sub-directory,
  - for the content, refer to the lecture notes,
- `typdef` should be used for referencing the `PERSON` structure,
- an array `employees[]` should be declared in the main C file (that is `struct.c`),
- a new C file `person.c` needs to be created; `person.c` should include implementations of the following functions:
  - `addEmployee()` to read the person data from the standard input and to add it to the array,
  - `displayEmployee()` to display the data for a single employee,
  - `displayAllEmployees()` to display data for all employees,
- the main program should read in the data for a number of employees by first prompting the user for the number of employees, allocating sufficient space for the employees in the array, and then calling the `addEmployee()` function to populate the current element of the array.

Here are the signatures of the functions that must be declared in `person.h` and defined in `person.c`:

```
void addEmployee(PERSON *employee);
void displayAllEmployees(PERSON employees[], int
numberOfEmployees);
void displayEmployee(PERSON *employee);
```

To compile the new `person.c` along with the `struct.c`, you need to modify the configuration file `CMakeLists.txt` in the root directory of the project by changing this line:

```
add_executable(struct src/struct.c)
```

to:

```
add_executable(struct src/struct.c src/person.c)
```

Question 10

100 / 100 pts

This is a graded question.

Using the CLion project [lab_1_2_processor.zip (https://cilearn.csuci.edu/files/500025/download?wrap=1)](https://cilearn.csuci.edu/files/500025/download?wrap=1), implement a program that processes batches of messages in a loop as follows:

- reads a message from the standard input into `inputBuffer`,
- collects the messages in `messageCache` using the function `addMessageToCache()` that takes `inputBuffer` as a parameter,
- when needed as explained in the next bullet, invokes a function `messageDispatcher()` to process a batch of messages that are in the cache one by one using the function `processMessage()`, and
- continues to take batches of messages from the input until either:
  - the capacity of the cache (arbitrarily chosen to be some fixed value; for example 16) is reached in which case the message dispatcher should be called to process the messages in the buffer, and then to continue reading messages, or
  - a line with a sentinel `END` is read, in which case the program should process all remaining messages in the cache, and then terminate.

There should be one message per line of input (assume correct input for the valid type). Messages can be in a number of formats:

- TYPE 1: a string of unknown size
- TYPE 2: 5 integers
- TYPE 3: 4 doubles
- TYPE 4: 5 five-character words

The input should indicate the type of the message and the content; e.g.:

```
1 abcdefghijklmnopqrst
3 123.34 23.12345 5439.234 0.000231
4 aaaaa bbbbb ccccc ddddd eeeee
```

```
2 123 45 6 7890 87592
```

After receiving the sentinel your program should finish processing the messages in the buffer and print processing statistics:

- the number of batches (including the last one even if incomplete),
- the total number of messages processed, and
- the number of messages of each type.

The `processMessage()` function should be invoked from `messageDispatcher()` for each message in the cache. It should take a single parameter that is the next message from the cache to process. It should recognize what type the message has, and then print the message type along with its content as follows:

- for type 1, the complete string,
- for type 2, all integers separated by commas,
- for type 3, all doubles separated by a slash, and
- for type 4, all words separated by spaces.

For example:

```
TYPE 1: abcdefghijklmnopqrst
TYPE 2: 123,45,6,7890,87592
TYPE 3: 123.340000/23.123450/5439.234000/0.000231
TYPE 4: aaaaa bbbbb ccccc ddddd eeeee
```

After all messages are processed (user entered the SENTINEL value) the final statistics should be printed by calling `printStatistics()` function:

```
Number of message batches: 1
Messages processed:
Total: 4
Type 1: 1
Type 2: 1
Type 3: 1
Type 4: 1
```

The cache should be implemented as an array of the `struct` constructs that include two fields: one for the type of the message and the second should be a `union` construct to hold the content of the message that is specific to its type. An unsigned short should be used as the data type for the message type field. The type should be used in a `switch` statement to save the message in the union according to the message type.

`processor.c` should contain a global declaration of the message cache and a global declaration of a number of messages read, as well as implementations of the functions `addMessageToCache()` and `messageDispatcher()`, and `processMessage()`.

`addMessageToCache()` should use `sscanf()` to read the type of the message and the message content from the parameter. To do that, an appropriate pattern for `sscanf()` needs to be designed. A `switch` statement should be used to select proper handler for each message type.

The program must use dynamic allocation for storing messages of type 1. The memory allocated for that purpose must be freed after the message is processed.

You should have `processor.h` file for declarations of data structures and declarations of common functions.

`test_processor.c` should include the `main()` with the testing driver that implements the main loop of the application.

KeithSkinnerLab1.zip (https://cilearn.csuci.edu/files/596531/download)

Quiz Score: **100** out of 100