

CS 200 Finals

CS 200
Fall 2020
Final Exam
12/08/2020
Time Limit: 3:00 hours

- **Exam Structure:** This exam contains two parts. Part 01 is the theoretical exam. Part 02 is a practical exam. The theoretical exam part contains 12 questions (a mix of multiple-choice and descriptive questions) provided through an online portal. The practical exam part contains one or more programming requirements outlined in this document that require implementing code files to address all the requirements provided.
- **How to take this exam?** You may *not* use your books, notes, phone, and other resources on this exam. You are allowed to use a resource sheet (1 page can be 2 sided) handwritten or typed notes in order to help you to do well in this exam.

Think carefully before answering the following questions. After the answer submitted, then you may not be able to change it!

To prioritize health and to follow the college's guidelines on physical distancing, this exam is administered online. Please note: All test takers are required to sign a pledge to abide by the class honor code. Any violation(s) will be considered fraud, and outside the norms of staying true citizens of the gator community. In-person attendance is not required during the exam day. The final exam is from 2:00 to 5:00 PM. The exam will begin once a student begins the exam and does not pause if the student navigates away from the exam. It is required for everyone to take the test during the scheduled time, unless otherwise notified. If extra time is needed, then the student should contact the professor as soon as possible, and an arrangement will be done to accommodate the request.

- **Additional Details:** I had tried to proofread the exam carefully, but if you see something that looks like an error, please notify me immediately so that I can look at it and possibly correct it. I had tried my best to create the questions so that it is simple to understand and easy to read, but if you find the choice of words to be confusing or unclear, then you should take a break and stop by at my Slack space as soon as possible so that I can clarify.
-

EXAM SHEET

1. (50 points) **PART 1:** Navigate to the online portal link below and follow the **Part 1 guidelines** provided in the portal.

<https://forms.gle/vjMzttSAUSuGLRKW7>

2. (50 points) **PART 2:** Provide detailed implementation for the requirements outlined below:
Hard Requirements: There will be full points awarded if you implement all the requirements thoroughly as indicated. Partial points will be awarded if there is any compromise done to implement any of the requirements listed below. If the code does not compile and/or throw any runtime errors, then there will be additional points taken. So be cautious to make necessary backups during the exam when you have working code and check the time regularly to make sure you finish the requirements within the allocated period. Add necessary comments in your program files, to maximize your chances to receive partial credit.

Make sure to sign your name in the honor-code.md file. Edit this file to include your name at the end of the file.

Please note, as per the college policy, we are unable to **grade** your work if you fail to add the statement above in all your code files. All your commits are logged in GitHub so please make sure to follow the protocols outlined above.

1. **MIPS Down Counter [15 points]:** The `counter.asm` program, prints the numbers from 0 to 10 (inclusive) in an ascending order using an iterative logic. To complete this part of the exam, it is required to do the following:

- Carefully read through the code file, and identify how the starter code works. It may be required to compile and execute the program a few times to fully grasp and understand the logical flow of the starter code.
- Implement the appropriate code changes to replace the core logic of the program to print the numbers from 0 to 10 (inclusive) in descending order using an iterative logic.
- Once, this implementation is done, the code should be tested out. The expected result should be in the opposite order of the result printed from the results printed in the starter code. A sample screenshot is shown here for your reference.

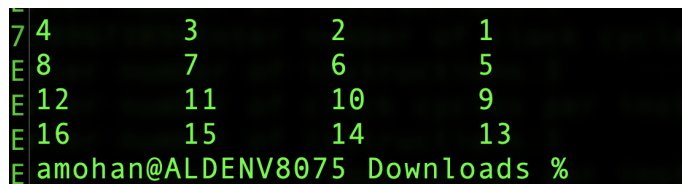
A screenshot of a terminal window showing the output of a MIPS assembly program. The text is green on a black background. At the top, it says "MARS 4.5 Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar". Below that, the numbers 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, and 0 are printed on separate lines, in descending order.

2. **Circuit Development [20 points]:** Develop a logical circuit diagram for the expression provided below using the Logicly web interface. The web interface may be accessed using the link below:

<https://logic.ly/demo>

- $z = (\neg a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c)$
- Once the circuit design is completed and the correctness verified, capture a clear screenshot of the circuit and name the file as `circuit`. Add this file for submission.

- Develop a logical truth table for the logical expression provided in the previous part. There are three inputs, and hence there is a total of 8 combinations for the total number of inputs. Edit the truth-table markdown file to provide the correct details related to the logical expression. In the markdown file, $t1$ represents the first term, $t2$ represents the second term, $t3$ represents the third term respectively. That is, $t1 = (\neg a \wedge b \wedge c)$, $t2 = (\neg a \wedge \neg b \wedge c)$, $t3 = (\neg a \wedge \neg b \wedge \neg c)$.
 - Note: Make sure to verify the correctness of this truth table by testing out the combinations using the circuit diagram developed in the previous part. Once work is completed, add the completed truth-table markdown file for submission.
3. **Array Processor [15 points]:** The `arrays.c` program, prompts the user to input two numbers, for row and column sizes. The starter code provides a platform to load and print the values in the two-dimensional array space (dynamically) using the malloc feature. There are two schemes used in the starter code program. The first scheme (`scheme1` method), is fully implemented. This scheme loads the array incrementally starting from 1. To complete this part of the exam, it is required to do the following:
- Carefully read through the code file, and identify how the starter code works. It may be required to compile and execute the program a few times to fully grasp and understand the logical flow of the starter code.
 - Implement the `scheme2` method, to load and print the dynamic malloc based two-dimensional array, with contents in the reverse order. A sample screenshot of a 4 by 4 array using `scheme2` is provided below for reference:



```
7 4      3      2      1
E 8      7      6      5
E 12     11     10     9
E 16     15     14     13
E amohan@ALDENV8075 Downloads %
```

- Once, the `scheme2` method is implemented, the code should be tested out. The expected result should be matching the data points from the sample output above. Of course, the output above is for a 4 by 4. The program is expected to run for different row and column sizes.
4. **Self Reflection:** It is required that we complete a self-reflection about your exam preparation, exam performance, and provide feedback about the exam and our CS-200 class. Please edit the reflection markdown file `reflection` to include your self-reflection with the following details:
- What did you do from your side to do well in this exam? (Feel free to comment on anything).
 - What did you think of this exam? Feel free to comment on anything — fairness, length, coverage, difficulty, or anything else.) Any constructive criticism is welcome.
 - What are the most helpful and likeable aspects in this course?
 - What changes in the course, do you think may be helpful for the next year students?