Lab 7 Grading Guidelines last modified by Aaron Bloomfield on August 13, 2013 11:27:16 AM EDT

Lab 7 grading guidelines

Pre-lab

Out of 10 points. This lab part **is** expected to compile.

Both IBCM programs should compile via the command-line interface, which the grading system should have done for you already. If it does not, check to see if it works in the online simulator. If it does not, then deduct the compilation penalty. Note that 'not compiling' on the online interface means that the file did not load up correctly.

For any executed IBCM file, there are three execution runs: the first is a regular execution run, showing only input prompts and output. The second execution run is a verbose trace, which decompiles each line and shows the result of the accumulator after executing that line. Note that for the second execution run for the in-lab (bubblesort.ibcm) and post-lab (quine.ibcm), it also shows the entire listing of the program both before and after program execution; this printing of the source is not enabled for the pre-lab execution runs. The third execution run is a decompilation of the program, which is provided if the submitted program was not properly translated into hexadecimal machine code.

addition.ibcm should take in three values, add them up, and print out the sum. If the sum is 0, stop and print out the three values. Otherwise, it start over again.

array.ibcm should determine the largest element in an array (hardcoded in memory) and print it out.

- 5 points: addition.ibcm file
    - Graded on a scale of 0-5, where 5 means it works as intended, 1 means it's not even close, and 0 is not submitted.
        - 5 points: Works perfectly or as well as it can in ibcm
        - 3 points: Part of it works.
        - 1 point: An attempt was made.
        - In addition, a penalty of -2 is applied for an IBCM file that is not commented
- 5 points: array.ibcm: Graded on the same scale as addition.ibcm
    - In addition, a penalty of -2 is applied for an IBCM file that is not commented

Feel free to interpolate between these values. When testing input, remember that ibcm takes in hex input and also that the cursor must always be the last thing in the input box when entering input into the simulator (otherwise it will think the last character appearing is the input).

Pre-lab input and output

The first file (addition.ibcm) takes in 3, 5, 7, -3, 0, 3 as input. The first three values (3,5,7) do not add up to 0, so it should print out the sum and continue. The second three values (-3,0,3) do add up to zero, so the program should print out zero and terminate.

The second file (array.ibcm) does not take in any input, but does provide output, which is the max of the array.

In-lab

Out of 10 points. This lab part **is** expected to compile.

The IBCM program should compile via the command-line interface. If it does not, check to see if it works in the online simulator. If it does not, then deduct the compilation penalty. Note that 'not compiling' on the online interface means that the file did not load up correctly.

For any executed IBCM file, there are three execution runs: the first is a regular execution run, showing only input prompts and output. The second execution run is a verbose trace, which decompiles each line and shows the result of the accumulator after executing that line. Note that for the second execution run for the in-lab (bubblesort.ibcm) and post-lab (quine.ibcm), it also shows the entire listing of the program both before and after program execution; this printing of the source is not enabled for the pre-lab execution runs. The third execution run is a decompilation of the program, which is provided if the submitted program was not properly translated into hexadecimal machine code.

For the bubblesort, there is a fourth execution run. This last one shows two program listings: one before the program was executed (so it should be same as the 3rd output), and one after the program executed. This allows for a comparison of the binary code before and after the sorting routine, to check if there really was a sort that took place.

bubblesort.ibcm should sort an array hardcoded into memory. Check to see that their array isn't already sorted before running their program.

- 10 points: bubblesort.ibcm. Graded on a scale of 0-10 (feel free to interpolate between these values):
  - 10 points: Works correctly
  - 8-6 points: Have the right idea, but it doesn't quite work correctly
  - 4 points: It doesn't work, but a sincere attempt was made
  - 2 point: Something was submitted, but it doesn't work at all
  - 0 points: No submission
  - In addition, a penalty of -2 is applied for an IBCM file that is not commented
  - Also, a penalty of -2 is applied if their code is already sorted

---

Post-lab

Out of 10 points. This lab part **is** expected to compile.

Both IBCM programs should compile via the command-line interface. If it does not, check to see if it works in the online simulator. If it does not, then deduct the compilation penalty. Note that 'not compiling' on the online interface means that the file did not load up correctly.

For any executed IBCM file, there are three execution runs: the first is a regular execution run, showing only input prompts and output. The second execution run is a verbose trace, which decompiles each line and shows the result of the accumulator after executing that line. Note that for the second execution run for the in-lab (bubblesort.ibcm) and post-lab (quine.ibcm), it also shows the entire listing of the program both before and after program execution; this printing of the source is not enabled for the pre-lab execution runs. The third execution run is a decompilation of the program, which is provided if the submitted program was not properly translated into hexadecimal machine code.

For the post-lab, there are two additional execution runs, in addition to the three mentioned above. The fourth execution run takes the output of the quine, and compiles and runs that. So the output from the fourth execution run should match the output from the first execution run AS WELL AS the original submitted quine.ibcm file, if it is really a quine (modulo a changed variable or two that will not affect the program execution).

Note: the execution runs for the post-lab are limited to the first 1,000 lines, to prevent infinite loops from creating huge output files.

The fifth execution run is for averagetime.sh. The supplied counter file (a.out) ignores any command-line parameters, and prints out a random integer between 50 and 59, inclusive (see Lab 6 Grading Guidelines - the inlab part - for details). So the result should be in that range.

- 4 points: averagetime.sh
  - 4 points: Works correctly, producing the average of the supplied (random) values. Note that they must have a loop (for or while) and decision (if) in there to get full credit
  - 3 points: More or less works correctly, but there are a few minor bugs
  - 2 points: Have the right idea, but something is wrong - a bash shell script parse error would be either here or in the 1 point category
  - 1 points: Does not work, but effort was made - a bash shell script parse error would be either here or in the 2 points category
  - 0 points: No submission
  - In addition, there is a 2 point deduction if they do not have a loop, and a 2 point deduction if they do not have a decision - thus, if they don't have either, they get 0 points for this part
- 4 points: quine.ibcm
  - 4 points: Works perfectly, and the quine output matches the original quine.ibcm
  - 2-3 points: Have the right idea, but something is missing so that it does not work. Score between these two is based on how close the results are.
  - 1 point: A good attempt was made
  - 0 points: No submission
  - In addition, a penalty of -2 is applied for an IBCM file that is not commented
  - There was also a penalty of -2 if the IBCM file had a file extension other than .ibcm.
  - There was a penalty of -2 if the quine never halted.
  - Not surprisingly, no one was given negative score for this section. Even if there raw score was negative.
- 2 points: postlab7.XXX
  - 2 points: Concerns, thoughts, and suggestions on IBCM are given. They only need a (single-spaced) quarter to half a

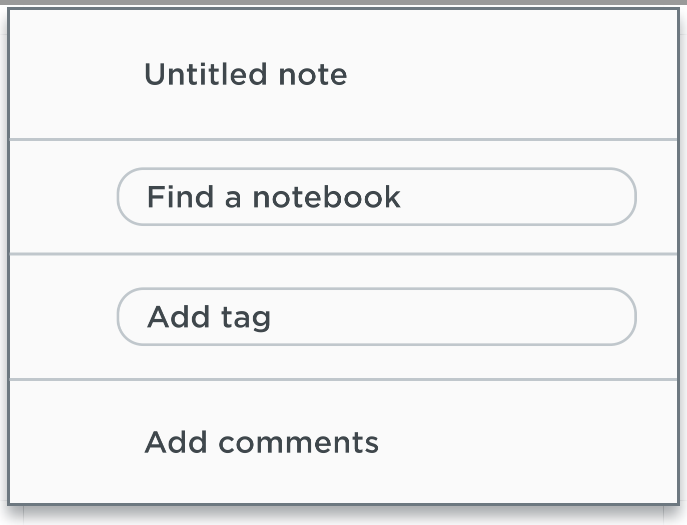
page. We aren't looking for a full-fledged write-up, but that a bit of thought and effort went into the report.
- 1 point: Did not put any effort into this report.
- 0 points: Nothing submitted

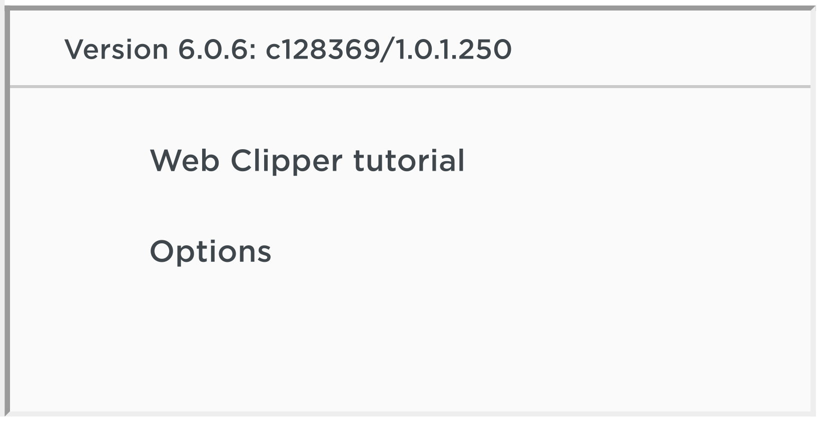
Post-lab output

The executable that the script called generated random numbers between 1 and 100. A sample output is below (theirs can vary, as long as the same information is there)

```
enter the number of times to iterate:
5
Running iteration 1…
time taken: 1256 ms
Running iteration 2…
time taken: 1232 ms
Running iteration 3…
time taken: 1238 ms
Running iteration 4…
time taken: 1240 ms
Running iteration 5…
time taken: 1256 ms
5 iterations took 6222 ms
Average time was 1244 ms
```

Be the first to comment




Version 6.0.6: c128369/1.0.1.250