**University of California, Berkeley – College of Engineering**Department of Electrical Engineering and Computer Sciences

Fall 2011 Instructor: Dan Garcia 2011-10-25

# CS10 Paper Midterm

|  |  |
| --- | --- |
| *Last Name* |  |
| *First Name* |  |
| *Student ID Number* |  |
| *cs10- Login First Lette*r | **a b c d e f g h i j k l m** |
| *cs10- Login Last Letter* | **a b c d e f g h i j k l m n o p q r s t u v w x y z** |
| *The name of your LAB TA (please circle)* | **Aijia   Glenn   Luke   Navin   Rabbit  Samir** |
| *Name of the person to your Left* |  |
| *Name of the person to your Right* |  |
| *All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS10 who have not taken it yet.* ***(please sign)*** |  |

**Instructions**

* Don’t Panic!
* This booklet contains 6 pages including this cover page. Put all answers on these pages; don’t hand in any stray pieces of paper.
* Please turn off all pagers, cell phones and beepers. Remove all hats and headphones.
* You have 110 minutes to complete this exam. The midterm is closed book, no computers, no PDAs, no cell phones, no calculators, but you are allowed two double-sided sets of notes. There may be partial credit for incomplete answers; write as much of the solution as you can. When we provide a blank, please fit your answer within the space provided.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Question** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **Online** | **Total** |
| **Points** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **6** | **2** | **4** | **10** | **8** | **4** | **10** | **60** |
| **Score** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

If you can draw and you have time, feel free to doodle all over this front page!

## Short-answer Questions (2 pts each, the last one 6 pts)

**Question 1**: We almost always want our algorithms to be correct. When would we soften that requirement? (i.e., what other concern might we have so that we would happily accept answers that weren’t 100% correct?)

*When the right answer would take too long to compute and a close-to-correct answer would also do.*

**Question 2:** Mobile computing users claim that it isn’t *Moore’s law* that affects them the most, but *Koomey’s law*, which found something very important doubled every 18 months (unabated, since 1950). What was it?

Energy efficiency (for a fixed computing “load”)

**Question 3:** What was the unintended consequence of the shift to multiple-choice (computer-graded) tests?

The redefinition of what knowledge is – if it can’t be asked in a multiple-choice exam, it’s not knowledge!

**Question 4:** Sir Ken Robinson talked about “changing educational paradigms” … name one paradigm.

That we should move away from standardization (the industrial one-size-fits-all model)

That we should support, not suppress collaboration “the stuff of growth”

That we should be thinking differently about human capacity (getting over the old conception of academic/non-academic, abstract/theoretical/vocational)

**Question 5:** What is so great about *Creative Commons*? (i.e., what does it allow an author to do?)

It allows an author to maintain copyright, and explicitly describe the conditions for sharing, remixing, and reuse of their work without having to be contacted, or involve intermediaries (e.g., lawyers).

**Question 6:** The *Connected* movie ends with a powerful quote: *“For centuries we’ve been declaring independence. Perhaps it’s time we declared our interdependence”*. What dominant technology has brought us to this point, and once we’ve made this new declaration, what is the movie suggesting we do next?

The Internet (and mobile phones) will soon connect the entire globe; she’s suggesting we think about the ripple effect of our actions, behave as one connected family, and work together to take humanity to the next level.

**Question 7:** In 1997, the MA Group Insurance Commision (GIC) released a 135,000-patient dataset, but made sure to “de-identify” it, removing names, addresses, SSNs, and telephone numbers. What happened next?

Researchers revealed that by combining other publically available datasets, they could identify all of them!

**Question 8:** For *four thousand years*, cryptography was done a certain way, until the 1970s. What changed?

You no longer had to keep the key secret – you could have Alice and Bob agree on a key without meeting!

a9,b3,c7,d2,e10,f4

**Question 9:** Match the person with what made them famous. Not all numbers need to be used.

|  |  |
| --- | --- |
| 1. Vint Cerf 2. John Warnock 3. Doug Englebart 4. Sir Tim Berners-Lee 5. Judah Schwartz 6. Harri Hursti | 1. Invented the first laserprinter 2. Built first web server in 1990, inventor of the world wide web 3. Invented Postscript, a language used for compactly specifying documents 4. Showed how to hack a Diebold voting machine to produce improper results 5. Wrote VisiCalc, one of the first PC spreadsheets 6. One of the inventors of the transistor. 7. Gave “mother of all demos”, showing first mouse and videoconference 8. Created one of the first popular web browsers 9. One of the developers of TCP, an important piece of Internet Protocol Suite 10. Came up with Tools-vs-Microworlds-vs-Courseware theory |

Login: **cs10-**\_\_\_\_

**Question 10: *A classy question!*** (2 pts)

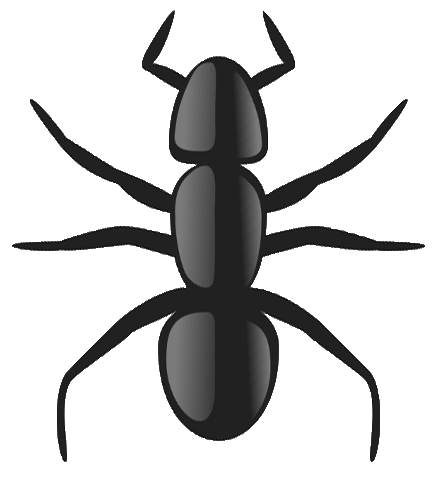
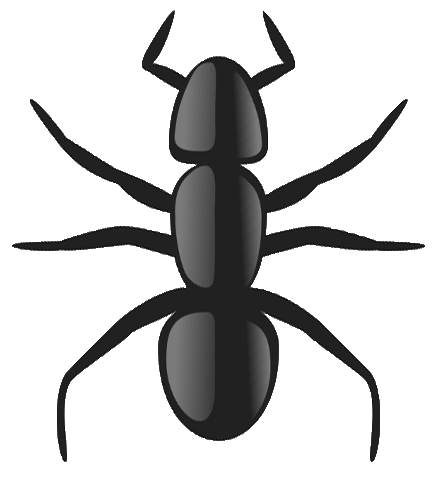
Draw the **CLASSES** list so that the following code returns **CS10**.

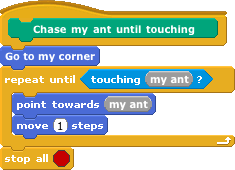
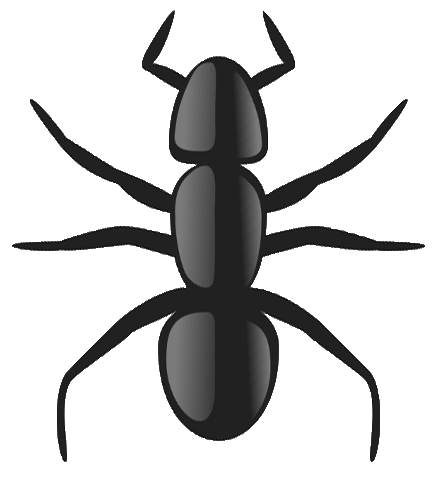
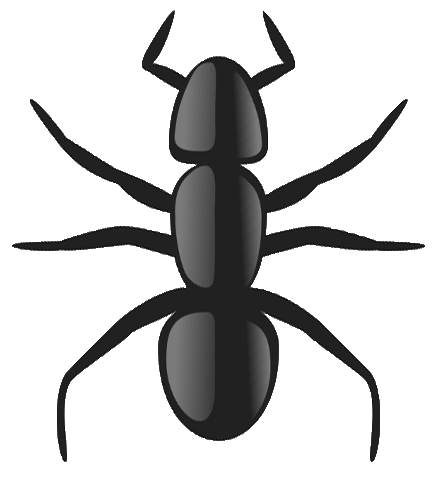
**(foo (CS10))**

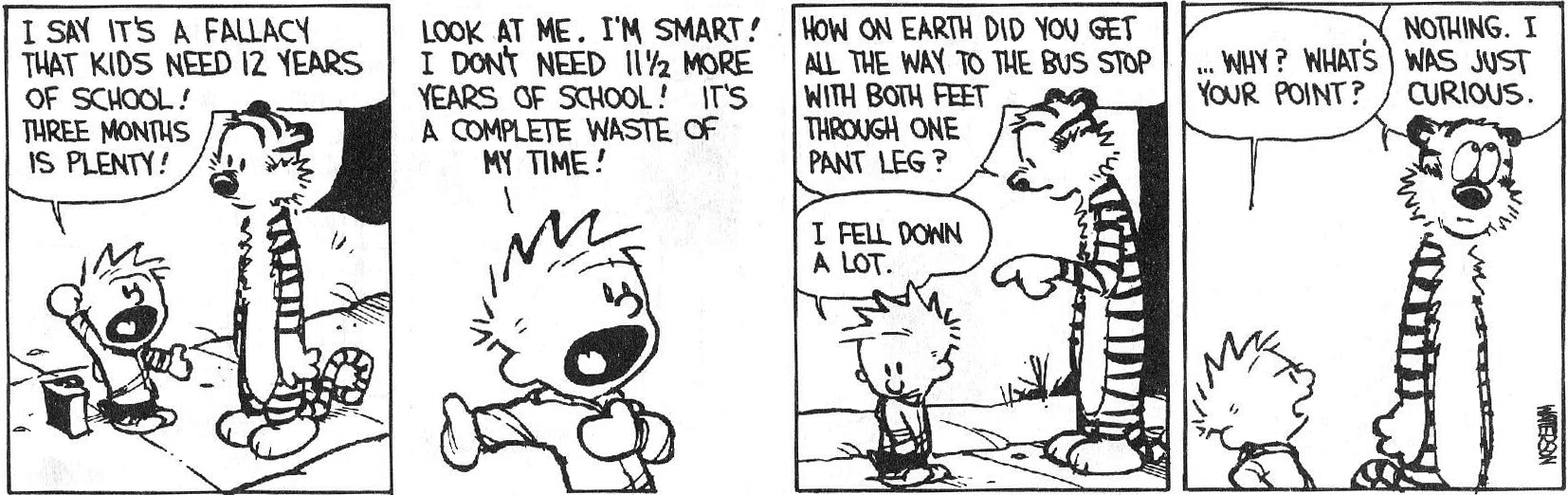
****

**Question 11: *I’ve got ants in my BYOB!*** (4 pts)

You place 4 ants (each a different sprite, with the “pen” in the center of the ant) on the four corners of an imaginary square shown in the picture below, each facing the ant to their left. Each runs the same command when the green flag is clicked (the block **my-ant** reports the ant you’re chasing).   
Draw what lines you would see on the stage after the program stops.

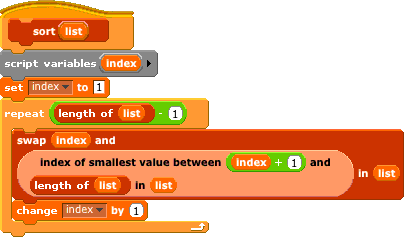






**Question 12: *Some sort of problem with my code...*** (10 pts)

We would like to take an unsorted list and *sort* it (into *increasing* order – the smallest element at index 1, the second smallest at index 2, etc). We’ve tried to write code to do this for us, but we believe it has a bug. The idea (that would work if we could get the details right) is to divide the list into two parts: the sub-list of items already sorted, which is built up from left to right and is found at the beginning, and the sub-list of items remaining to be sorted, occupying the remainder of the array. This is called “selection sort”.



We use two bug-free helper blocks described below:

|  |  |
| --- | --- |
| **Helper block** | **Description** |
|  | Search the **list** for the smallest value between the **left-index** and the **right-index** (*inclusive*, meaning *including* the elements living at the left and right indices) and report the index of the smallest value in that range |
|  | Swap the elements at **left-index** and **right-index** in the **list**. The length of the list remains unchanged. |

### What is the running time of sort? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Quadratic*

### If list is (4 3 2 5 1), what is list after sort(list)? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *(1 2 4 3 5)*

### Most of the time sort(list)*doesn’t* leave list sorted. Describe what an 100-element list (of all the numbers 1 through 100 in some order) would look like so that sort(list)does leave list sorted.

Two obvious patterns emerge: (2 3 4 5 … 98 99 100 1) and (100 1 2 3 … 98 99)

1. Briefly describe the single, very small change needed to fix the bug. Change index+1 to index.

Login: **cs10-**\_\_\_\_

### Question 13: *Strawberry Fields Forever...* (8 pts)

Strawberry plants are funny. Every year they send “runners” to their left and right neighbors, which take seed and become an entirely new strawberry plant the next year. We’d like to model this process and count how many strawberry plants we’ll have in our garden (that we’ve divided into columns, like the number line) starting from a single strawberry plant in column 0 in year 1, the top row. All other numbers in the top row are 0 (no other strawberry plants). The number in every subsequent row is the sum of the three numbers directly above it, to the above left and to the above right as shown below. We’ve filled in the table for years 1 through 5:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **C** | **O** | **L** | **U** | **M** | **N** |  |  |  |
|  |  | **…** | **-4** | **-3** | **-2** | **-1** | **0** | **1** | **2** | **3** | **4** | **…** |
|  | **1** | … | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | … |
| **Y** | **2** | … | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | … |
| **E** | **3** | … | 0 | 0 | 1 | 2 | 3 | 2 | 1 | 0 | 0 | … |
| **A** | **4** | … | 0 | 1 | 3 | 6 | 7 | 6 | 3 | 1 | 0 | … |
| **R** | **5** | … | 1 | 4 | 10 | 16 | 19 | 16 | 10 | 4 | 1 | … |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

You are to write a function **Plants** that takes two integer arguments, a *column* and a (positive integer) *year*, and returns the number of plants that will be in that column on that year. For example, if (column, year) = (1,5) then **Plants** should return 16. **Plants** should return 0 for any value not in the infinite triangular shaded area.

**Plants(column, year)**

**column = 0 and year = 0**

**if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**1**

**report ( \_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**year = 1**

**if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

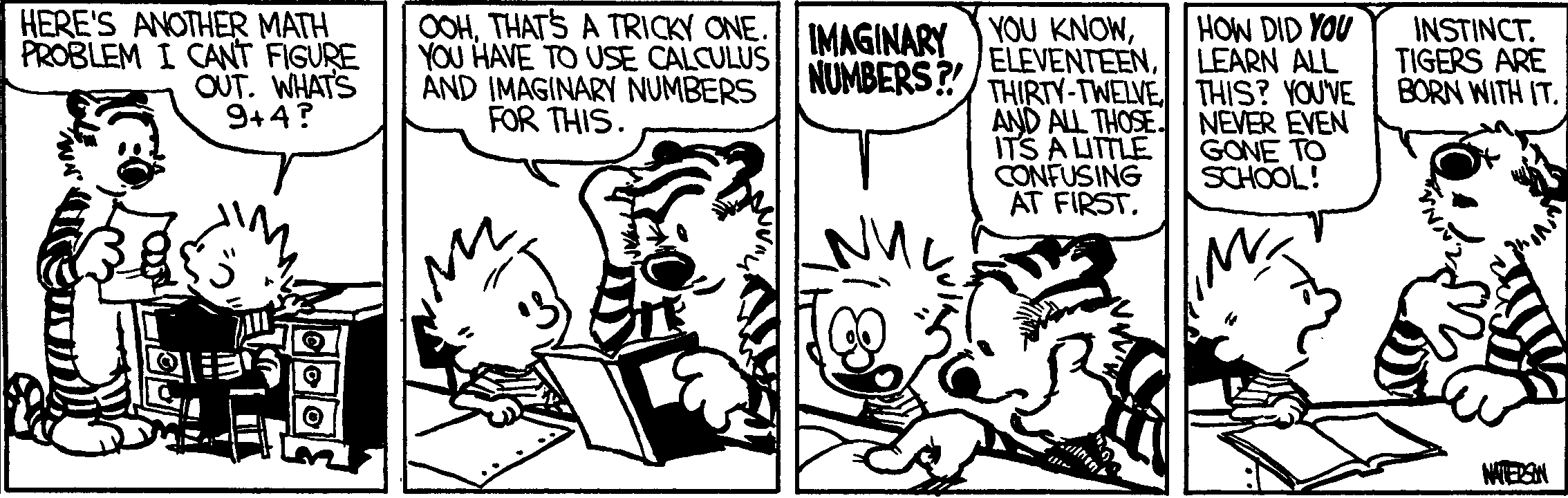
**0**

**report ( \_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**plants(column-1,year-1)+plants(column,year-1)+plants(column+1,year-1)**

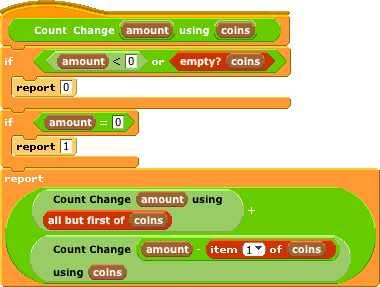
**report ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**



**Question 14: *Spare Change?*** (4 pts)

In lecture we showed **Count-Change**, which takes an amount and a list of coins and returns the number of ways to make change of that amount using those coins. This code is repeated below.



We’d like to see if the code would work if we *reversed* the input list of coins.

On the left is the call tree for the normal sorted-big-to-small coin list (assume a 2¢ coin exists); *yes*

draw the call tree for the reversed coin list on the right. Does it work if we reverse the coins? \_\_\_\_\_\_\_\_\_\_\_\_

**Use Coin**

**Skip Coin**

**Use Coin**

**Skip Coin**

2 (1 2)

2 (2) 1 (1 2)

2 () 0(2) 1(2) 0(1 2)

0 1 1() -1(2) 1

0 0

**The beauty and joy of computing**

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Your name – write it loud and proud!*

*Use this page to let your creativity flow! Draw a picture (or write a poem, haiku, limerick, etc.) about CS10, or BYOB, or computing in general.   
We’ll share these with the rest of the class.*

# CS10 Online Midterm (Fall 2011, sec 1)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# CS10 Online Midterm (Fall 2011, sec 2)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 3)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 4)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 5)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 6)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 7)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

# 

# CS10 Online Midterm (Fall 2011, sec 8)

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal (you don’t have to match our exact placement on the screen as), and name it **FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *“save picture of stage...”*) and name the resulting GIF similarly, i.e., **FractalBarackObama.gif**). Submit both on bspace under the “midterm” assignment.

Though this may look daunting at first, it really isn’t that bad. Remember, every fractal has a base case (**n = 0**) and recursive case. We’ve drawn the fractal with bold lines to indicate the parts of the drawing that will recurse; the other parts of the drawing at **n = 1** are just lines. (You don’t have to copy our bold-vs-normal technique.) Look at how the single **L** shape at **n=0** transforms into the **n=1** case -- this happens for every bold **L** when it goes to the next level.

|  |  |
| --- | --- |
|  |  |
| **n = 0** | **n = 1** |
|  |  |
| **n = 2** | **n = 3** |

**2010Fa CS10 Paper Midterm Answers**

**Question 1:** In the never-ending quest for increasing CPU performance [1 pt], chips running at high clock frequencies were getting too hard to cool (the power density was approaching that of a nuclear reactor) [1 pt] so they decided to put multiple, slower, cooler cores on a chip [1 pt].

Meta-comment: Almost everyone got this question. Some didn’t mention that the reason for going to multi-core was the issue with chips getting too hot unable to be cooled.

**Question 2:** Amdahl’s law states that the speedup in the perfect, infinite-helpers case is 1/s, so the speedup is 1/0.5 = 2x, or twice as fast. People received full credit for writing 1/2, because we assumed they were talking about the time (inversely related to speedup). Rubric was pretty much 2 or 0, and most everyone got this question.

**Question 3:** If every one of Twitter’s N users followed every other (N-1) user, that would be N \* (N-1) = N2 - N connections, which is (c) quadratic growth. All (2 pts) or nothing, and most people did not get this, choosing (d) exponential instead. Certainly Twitter’s populatiry growth has been exponential vs time, but this question asks about connections vs people.

**Question 4:** Twitter “pushes” the results to Google (and Google pays handily for the data). All (3 pts) or nothing, and surprisingly it seemed that 75% had this right.

**Question 5:**

a) Google Docs (and other Software as a Service (SaaS) utilities), Wikipedia too.

b) The laserwriter and Postscript. The phrase “put professional-quality in the hands of the masses” was meant to imply that it was one of the **first** to do it. So WYSIWYG and Microsoft Office are half-right (but they’re both software). They too made life easier but without the laserwriter, you don’t get professional-quality output by your desktop.

c) his web browser, NCSA Mosaic.

Grading standard. Of the 4 blanks, 4 right = 3 pts. 0-or-1 right = 0-or-1 pts. 2-or-3 right = 2 pts.

**Question 6:**

Courseware devalues creativity and critical thinking in favor of memorization, but tools make it too easy for a learner to get lost down dead ends or not know where to begin. Microworlds establish the general topic of exploration but don't pose overly specific problems.

**Question 7:**

* Removing a result at the request of its owner (e.g. for copyright reasons).
* To prevent or punish attempts to mislead the ranking algorithm (e.g., bmw.de)
* Special arrangements with high-volume sources (e.g. Twitter, CNN).

Wrong: Payment by advertisers.

**Question 8:**

The purpose of the patent system is to make a bargain between inventors and society, whereby inventors can profit from their work, but technology is ultimately public property, available to all.

**Question 9:**



You received the full **2** points if you answered (item (1) of (item (3) of (playoffs))) or provided an equivalent answer. You received 1 point if you used incorrect indices or nested the "item" blocks incorrectly; for example, (item (item (3) of (playoffs)) of playoffs) -- this is incorrect because the result of (item () of ()) is a list in this question, and you cannot put a list into an input meant for a number. You also received 1 point if you returned a list containing "SF", instead of "SF" itself; this is, for example, the result of simply (item (3) of (playoffs)). All other incorrect answers received 0 points.

**Question 10:**

The list would have fewer than 100 items because of the concurrency *race condition* on the multiple “**delete all of nums**” commands (this is actually what happens in Scratch/BYOB). The 51 never makes it to the final list because the right script finishes its “**delete all of nums**” then “**add 51 to nums**” first, THEN the left script executes its “**delete all of nums**” which *removes* the 51. In the absolute worst degenerate case, one block could add all 50 of its numbers only to have them deleted by the other block ... yikes!

You received the full **3** points for a correct answer, or if you mentioned that there would be a "race condition", where less than 100 numbers would be present in the final list, or where some numbers would be deleted. If your answer mentioned that one script would overwrite the result of the other script, or if your answer does not mention that elements from the final list would be deleted, or if your answer lacked mentions of a "race condition" but implied the existence of one, then your answer received 2 points. If your answer mentioned that only one script would run, or that there would be multiple errors but a correct final result, it received 1 point. All other incorrect answers received 0 points.

Based on the answers that we received for this question, we wanted to clarify: Scratch and BYOB do support concurrency, and both scripts will always run. However, one script can never overwrite the result of the other script in this case, since each script is either deleting all of the elements of the final list, or adding one element to the final list successively: only the "replace" block could potentially cause an overwrite.

**Question 11:**

a) All **sentence**s that do not have the input character.

b) Change the **and** to an **or**.

The **5** points for this question were split between **3** points for part (a) and **2** points for part (b).

Your answer for part (a) received **1** point if it provided a subset of the correct answer: for example, many answers claimed that the code would only work “if the first letter is not character”. While this is true, it does not include all sentences that do not trigger the bug. If you gave specific examples, then your answer received no points.

Your answer for part (b) was graded based on your answer for part (a): if your suggested change correctly fixed the bugs in the sentences that you described in part (a), you received 2 points even if your answer to part (a) was incorrect, but such answers were rare. In general, many students correctly answered part (a), but suggested a fix that would not work for all input sentences: such answers generally received 1 point.

For example, a common, but incorrect, suggested fix was to remove the if-constraint that letter should not be equal to char; in other words, the script should only add the current letter to the final answer if the last letter does not match the character. This would correctly replace consecutive repeated characters with one of the characters, but it would not include the letter that comes just after a consecutive string of these repeated characters. For instance, in the sentence abbbbcd, the b’s would be reduced to one b, but the c after the b’s would not be included, resulting in the sentence abd. Another common, but incorrect, suggested fix was to replace the if-constraint with the constraint (not (last-letter = letter)); in other words, the script should only add the current letter to the final answer if the last letter is not equal to the current letter. This would definitely work and replace all consecutive characters with one character. However, this means that it would replace *all* consecutive characters, even the ones that are different from the character provided to the block. Thus, the sentence abbbcddde would result in abcde, even if we specify that the only character we need to consider is b.

A few students suggested the constraint (not (letter = character)) or ((letter = character) and (not (lastletter = character))), which basically says that the current letter must be added to the answer if it is not character, but if the letter *is* character, then we must ensure that the last letter is not character. This is the equivalent to our fix, but is perhaps more intuitive. This answer received 2 points.

**Question 12:**

**path-home?(PLACE)**

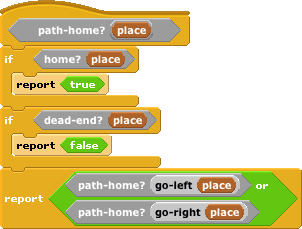
**if home?(PLACE)**

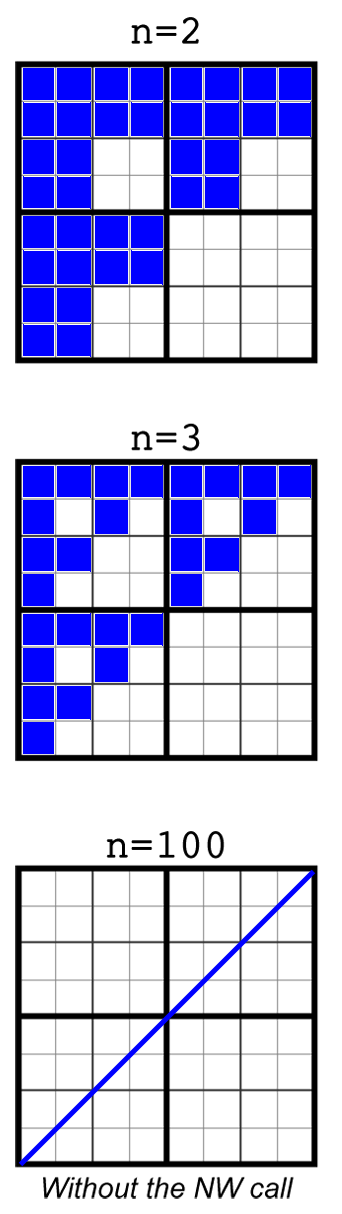
**report(true)**

**if dead-end?(PLACE)**

**report(false)**

**report( path-home?(go-left(PLACE)) or path-home?(go-right(PLACE)) )**

**Question 13:** 



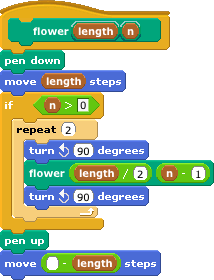
The **8** points for this problem were split between **5** points (in total) for the fractals where n = 2 and n = 3, and **3** points for the question that asked what the n = 100 call would look like, in the absence of the NW recursive call.

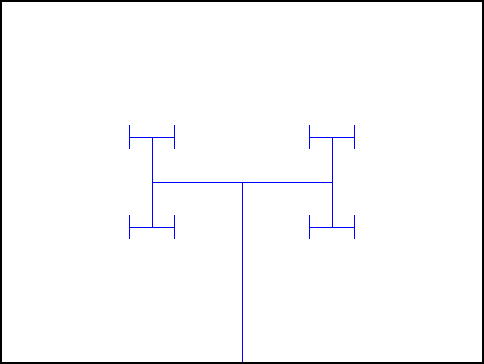
You received 3points (out of 5) for the first two parts if your answers were incorrect but *consistent*: it looked like your answers for the two fractals were related, and the fractal for n = 3 could conceivably be obtained from the fractal for n = 2 using appropriate recursive calls.

You received 1 point for the last part if you had a “checkerboard” pattern, which possibly implied that you did ignore the NW recursive call, but still called the resulting pattern recursively in all four directions. All other incorrect answers received 0 points. The most common mistake on this part was to forget that ignoring the NW recursive call also implies that you never go into that region of the fractal “checker”.

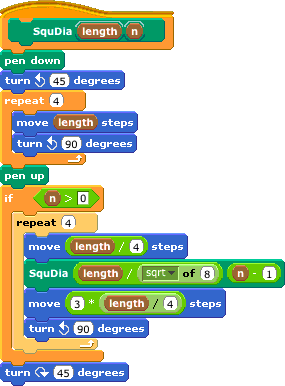
**2010Fa CS10 Online Midterm Answers**

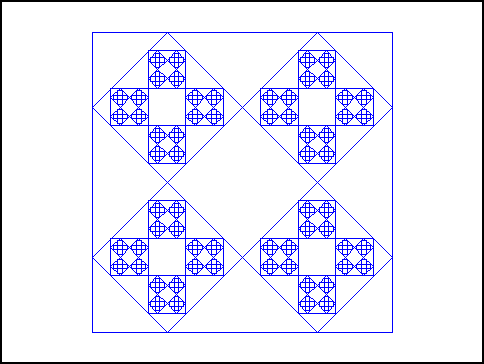
**Fractal Tree:**



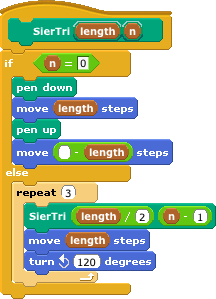


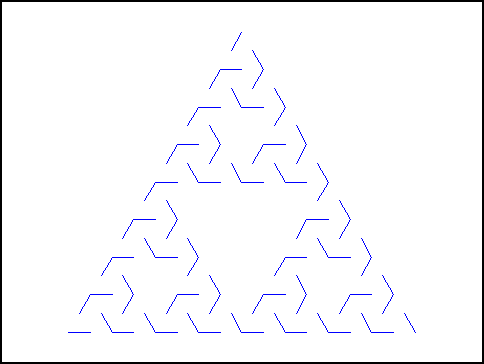
**Fractal Square Diamond:**





**Fractal Sierpinski Triangle:**



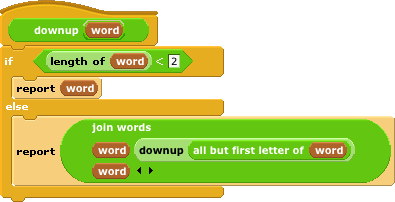


# Writing Scratch/BYOB code on paper

You might be asked to write Scratch/BYOB code on exams, so we’ve developed a technique for writing it on paper. There are a few key things to notice:

* + We write variables in **UPPERCASE**.
  + We change spaces between words in block names to dashes (this makes it much easier to read).
  + Parentheses mark the start and end of a parameter list, and we separate consecutive parameters by commas
  + We use indentation just as Scratch/BYOB does, to help us understand what is “inside” the **if,** **else**, and other Control structures.

Here’s a sample (and a familiar piece of BYOB code):



...and here’s how we would write it on an exam using our technique:

**downup(WORD)**

**if length-of(WORD) < 2**

**report(WORD)**

**else**

**report(join-words(WORD, downup(all-but-first-letter-of(WORD)), WORD))**

Here’s how you could write the **factorial-of** block from lab.

**factorial-of(NUM)**

**if NUM = 1**

**report(1)**

**else**

**report(NUM \* factorial-of(NUM - 1))**