University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Fall 2010 Instructors: Dan Garcia and Brian Harvey 2010-10-27

CS10 Paper Midterm

Last Name	
First Name	
Student ID Number	
`cs10-` *Login First Letter*	`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)	**Jon** **Luke**
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.
- **Question 0 (1 point) involves filling in the front of this page and putting your login on the top of every sheet of paper.**
- 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	0	1	2	3	4	5	6	7	8	9	10	11	12	13	Online	Total
Points	1	3	2	2	3	3	3	3	2	2	3	5	10	8	10	60
Score																

Short-answer Questions

Question 1 : A “sea-change” in CPU design occurred around 2004 and most agreed the “free lunch” was over. *In one sentence*, what caused it and what did manufacturers do in response?

Question 2: You have a piece of code, half of which is executed serially, the other half is executed in parallel. In the ideal case with an infinite number of “helpers” dividing up the parallel portion, what is the overall speedup of your code? ______

Question 3: Let us imagine Twitter mandated that every one of their users “follow” every other user. How does the *total* number of “connections” between people (connections were visualized as arrows, or links in the graph) scale with the number of users?

(a) It remains relatively constant, regardless of the number of people.
(b) It scales linearly with the number of people.
(c) It scales quadratically with the number of people.
(d) It scales exponentially with the number of people.

Question 4: *In one sentence*, how are search engines (e.g., Google) able to have a tweet available in their search results in less than a second after the tweet happens? (It normally takes a search engine hours to “crawl” the web and update their search index.)

Question 5: Fill in the blanks with the appropriate technology that has changed the world.

a) ____________________ is revolutionizing the way people collaboratively author documents.

b) ____________________ and _________________ are a hardware and software combination that put professional-quality document production in the hands of the masses.

c) The Internet was “invented” in 1962, but it wasn’t until 1993 when Marc Andreesen at NCSA introduced _____________________________ (known as its first “Killer App”) did its use explode.

Question 6: Give an argument why educational microworlds might be preferable to either pure tools or pure courseware.

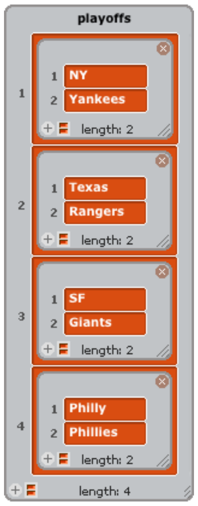
Question 7: Besides obeying local laws, name one reason why Google makes deliberate (non-automatic) interventions in the display of search results.

Question 8: Why did the writers of the US Constitution require a limit to the duration of patents?

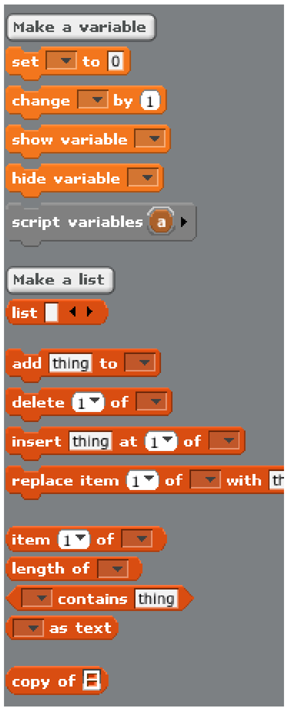
Login: **cs10-**____

Question 9: The Giants win the Pennant! The Giants win the Pennant!

Provide a single, simple expression that reports **SF** from the nested list **playoffs**. On the right is a screen capture of the Variables tab if that helps.

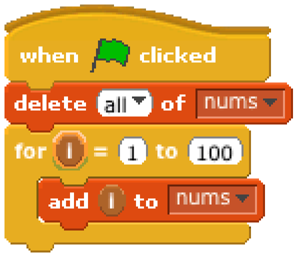


```
playoffs
1  1 NY
   2 Yankees
   length: 2
2  1 Texas
   2 Rangers
   length: 2
3  1 SF
   2 Giants
   length: 2
4  1 Philly
   2 Phillies
   length: 2
length: 4
```



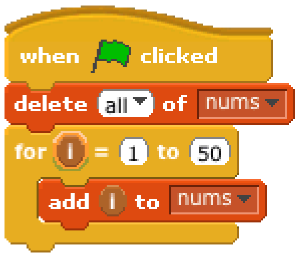
Question 10: Faster, Puppydog! Live! Live!

You wish to fill a list **nums** with the numbers from 1-100 (you don't care about order). So, you write the following (the **for** block simply goes through like a **repeat** block with a built-in index, called **i** in this case):

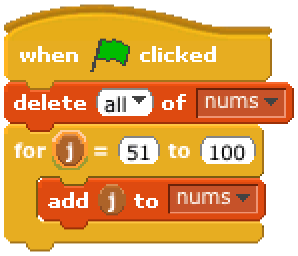


```
when green flag clicked
delete all of nums
for i = 1 to 100
    add i to nums
```

which works perfectly, but it's a little slow. However, your friend asks *"why don't you break the problem into two parts and fill the list in parallel!"* So, you duplicate that script, and adjust the numbers in the **for** loop like so:



```
when green flag clicked
delete all of nums
for i = 1 to 50
    add i to nums
```



```
when green flag clicked
delete all of nums
for j = 51 to 100
    add j to nums
```

In the best case, this works! What happens (or could happen) in the worst case and why?

Question 11: Talking' about my gggggeneration...

If you've ever spilled soda into a keyboard, you know that keys start to stick and repeat. We want to write a block to remove the extra consecutive *repeat character* we request, leaving only one. E.g., if the sentence were:

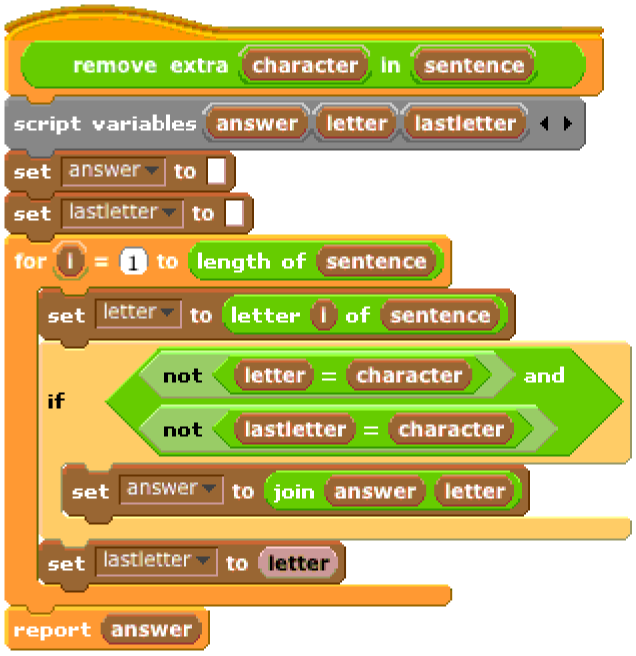
```
I love CS10....... Me too....... Me three!!!
```

...and we asked our block to remove the extra "**.**"s, we would get:

```
I love CS10. Me too. Me three!!!
```

(We'll limit this to punctuation, since the meaning could be lost if we used it on letters and numbers. Imagine: "**The national debt is $10000000000000!!**" would become "**The national debt is $10!!**")

We've tried to write code to do this for us, but we believe it has a bug. Briefly, **answer** and **lastletter** initialize to the empty character, and then the **for** loop (these were explained in question 10) sets **letter** to every character as we walk forward in the **sentence**. We only add that **letter** to the **answer** (which we report at the end) if the **letter** is not the **character** to remove and not the same as the previous character **lastletter** (to prevent the repeats).



```
remove extra (character) in (sentence)
script variables (answer) (letter) (lastletter)
set answer to [ ]
set lastletter to [ ]
for (i) = (1) to (length of (sentence))
    set letter to (letter (i) of (sentence))
    if <<not <(letter) = (character)>> and <not <(lastletter) = (character)>>>
        set answer to (join (answer) (letter))
    set lastletter to (letter)
report (answer)
```

a) Describe all **sentence**s that don't trigger the bug, i.e., our program returns a correct answer.

b) Briefly describe the small change(s) needed to fix the bug.

Login: **cs10-____**

Question 12: Two roads diverged in a wood...

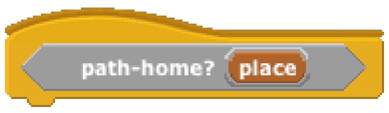
You're lost in the forest. Every place in the forest is either a *dead-end* or has exactly 2 *one-way paths*: *left* and *right*. Your goal is to find out if there is a way home. We introduce a new data type called a place, but you don't know *(and you don't need to know)* how it is represented; it could be a string, a number, or a list. You are presented with four new blocks, two predicates and two reporter blocks (all take a place as an argument):

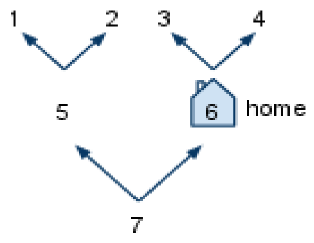
- home? place returns true if the place is your home, false otherwise.
- dead-end? place returns true is the place is a dead-end (i.e., no paths from it).
- go-left place follows the *left* path, returning a new place.
- go-right place follows the *right* path, returning a new place.

It is an error to go-left place or go-right place if place is a *dead-end* (because it has no paths!). There is no way in this forest to follow a sequence of left paths and/or right paths and end up where you started. I.e., there's no way to walk in circles. Your *home* (if one exists) might be at a dead-end or it might not. You might actually start your search at home.

Write path-home? place, which uses the four functions above and returns true if you can get home following a (possibly zero) number of lefts and rights starting from place, and false otherwise. Use the technique we described for authoring BYOB code on paper. We've provided an example forest for you, but **your solution needs to be able to work with ANY forest.** Below, we present a table that shows the responses of various blocks when you are at different places in the sample forest on the lower right.

place	home? place	dead-end? place	go-left place	go-right place	path-home? place
1	false	true	ERROR	ERROR	false
2	false	true	ERROR	ERROR	false
3	false	true	ERROR	ERROR	false
4	false	true	ERROR	ERROR	false
5	false	false	1	2	false
6	true	false	3	4	true
7	false	false	5	6	true

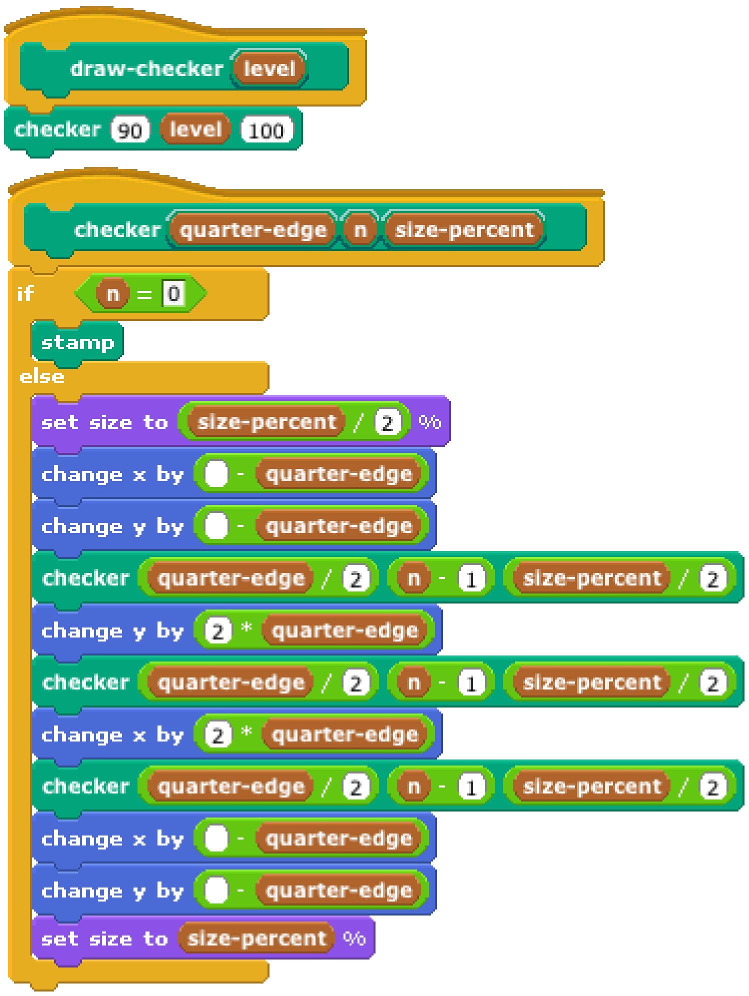

path-home? place



Question 13: Anyone up for a game of checkers? Hey, you, Sierpinski!

We've designed a fractal "checker" whose base (n=0) case simply stamps out a filled square sprite (initially 360x360 pixels), and whose recursive case places three half-sized copies of the previous generation in the Northwest (NW), Northeast (NE) and Southwest (SW) corners. The code that generated them is on the left, **but you can answer the question without it.**

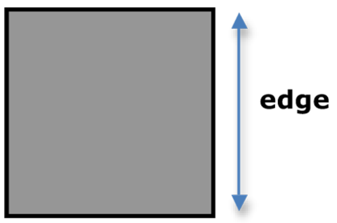
a. Sketch the **n=2** fractal below on the right.
b. Sketch the **n=3** fractal below on the right.
c. If we forgot to write the Northwest (NW) recursive call (the middle call to **checker**), roughly sketch what the **n=100** call would be.



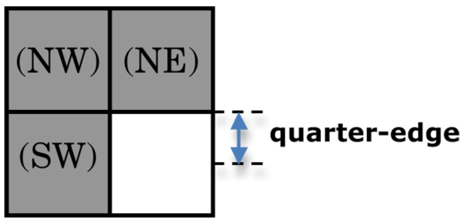
```
draw-checker (level)
checker (90) (level) (100)

checker (quarter-edge) (n) (size-percent)
if <(n) = (0)>
    stamp
else
    set size to ((size-percent) / (2)) %
    change x by (( ) - (quarter-edge))
    change y by (( ) - (quarter-edge))
    checker ((quarter-edge) / (2)) ((n) - (1)) ((size-percent) / (2))
    change y by ((2) * (quarter-edge))
    checker ((quarter-edge) / (2)) ((n) - (1)) ((size-percent) / (2))
    change x by ((2) * (quarter-edge))
    checker ((quarter-edge) / (2)) ((n) - (1)) ((size-percent) / (2))
    change x by (( ) - (quarter-edge))
    change y by (( ) - (quarter-edge))
    set size to (size-percent) %
```

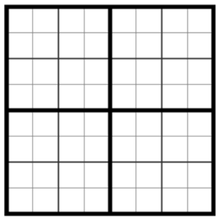
n=0



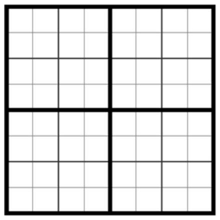
n=1



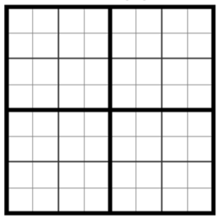
n=2



n=3



n=100

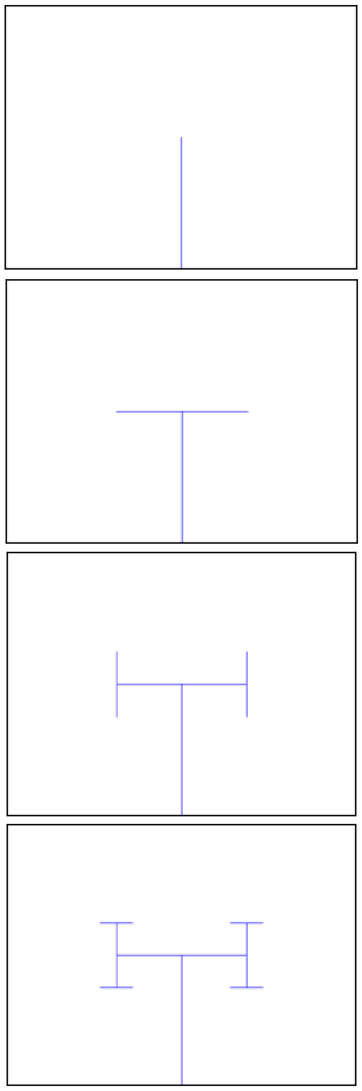


Without the NW call

2010Fa CS10 Online Midterm

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal, 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.

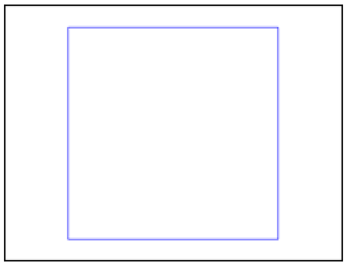
The stalks of the "tree" get half as short at every branch.

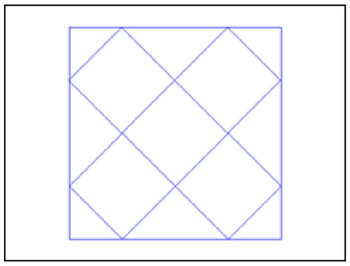


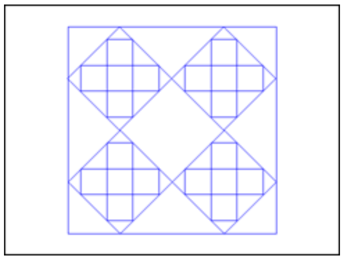
2010Fa CS10 Online Midterm

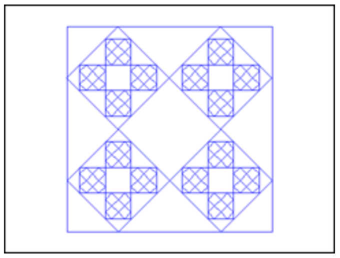
Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal, and name it **`FractalYourfirstnameYourlastname.ypr (e.g., FractalBarackObama.ypr)`**. Also, save a screenshot of the *fifth* iteration (right-mouse-click on the stage and choose *"save picture of stage..."*) and name the resulting GIF similarly, i.e., **`FractalBarackObama.gif`**). Submit both on bspace under the "midterm" assignment.

Our starting square is 300 pixels on a side, centered on the screen. The length of the sides of the four smaller diamonds is the length of the square side divided by the square root of 8.
Hint: We found the **`repeat`** *block very helpful in both the base case and recursive case.*





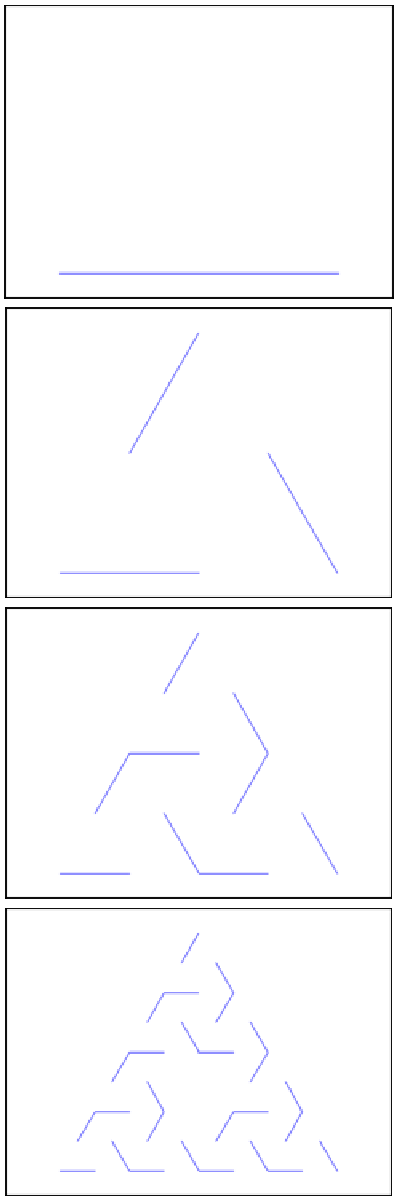




2010Fa CS10 Online Midterm

Below are screenshots of the first four iterations of a beautiful fractal. Write code that generates the fractal, 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.

The exact size and location of your initial triangle are not important, so don't spend time worrying about that, just make it big enough so we can see it. The recursive calls simply aim the half-sized smaller version toward the middle from the corners of the original triangle.
*Hint: We found the **`repeat`** block very helpful in the recursive case.*

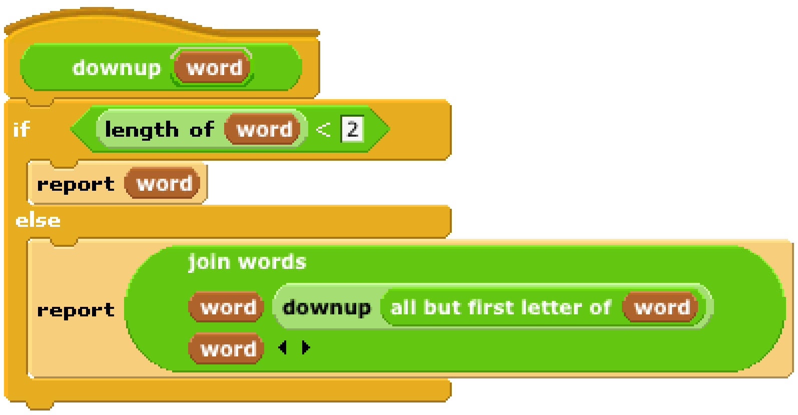


Writing Scratch/BYOB code on paper

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):


downup word
if length of word < 2
report word
else
report join words word downup all but first letter of word word

...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))
```