NAME: _____

CS 330 Midterm
Spring 2013

Doodle here.

1. *Accepts or Regects*
   Consider the strings in the left column of the following table. Each remaining column has at its head a Ruby-type regular expression. If a regex matches a string, place an X in the corresponding cell.

	`^[0-1]$`	`[^A-z]`	`$`	`^(0|\$)`	`.*`	`..+`
`A`						
`01`						
`LOLdog`						
`$$$`						
`0`						
`_/-\_`						
`squeaks`						

2. *Star-crossed Lovers*
   For each type below, describe a situation in which you would use a variable of that type:

   (a) `char`

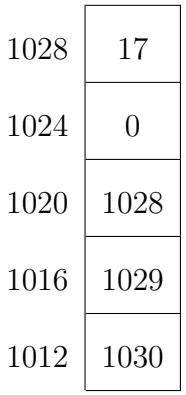
   (b) `char *`

   (c) `char **`

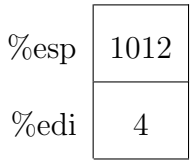
3. *Splitsville*
   What is the difference between the stack and the heap? Why do we have both?

4. *My Name Is Legion*
   Arrays in C do not store their length. This makes iterating through elements a challenge. I can think of at least three ways to address this problem—without going so far as changing the language. Match or surpass me.

5. *Backtrack*
   Suppose memory and registers have the following contents:



(a) Memory



(b) Registers

   Write any assembly code you want that yields this setup when executed. Recall that `%esp` points to the last "declared" element on the stack—not the first free element. You may assume that `%esp` holds 1032 prior to your code being run.

6. *Wreck You Self || Check Yo Self*
The shell does little in the way of checking calls to scripts. For example, consider this script in file `colonize`:

```
#!/bin/sh
echo "$1:$2:$3"
```

When you run `colonize a 3`, the output is `a:3:`, which may not be what you wanted.

C compilers, on the other hand, are more persnickety, checking functions calls before producing object code. Augment this code so that it compiles and type checks:

```
#include <stdio.h>


int main(int argc, char **argv) {
  colonize(argv[1], argv[2], argv[3]);
}


void colonize(const char *a, const char *b, const char *c) {
  printf("%s:%s:%s\n", a, b, c);
}
```

However, the executable may still be invoked with the wrong number of arguments: `./a.out a 3`. Describe or code how you might fix this.

7. *SVN*
   By annotating our data with type information, we empower the compiler to "double-check" our work and make sure that we do not try to perform unsupported operations on our data. The C language, despite its typing system, allows us to undermine types. We say it's a "weakly-typed" language. Describe or code one example that shows why C deserves this label.

   (Note that this doesn't mean C is not a language worth knowing and using. P.J. Moylan quips that C combines "the power of assembly language with the readability of assembly language.")

8. *Zupcakes*
   People whose names are are the end of the alphabet do not get the attention they deserve. You decide to bake for the 5 people in this class whose names are alphabetically last. Write a single shell command using pipe redirection that prints out these 5 names. The names appear in unspecified order in file `names.txt`. The `sort` utility sorts its input linewise. `tail -n #` prints out the last # lines in its input. `cat` dumps its input to standard output. `y` prints out an endless string of `yes`es to standard output.