22C:16 Exam 2

March 28th, 6:30 pm to 8:30 pm

Instructions:

- This is an open notes exam and you have 2 hours to complete it. There are 4 problems in the exam and these appear on 8 pages.
- Make sure that you do not have any electronic devices (phones, calculators, etc.) on your desk; you are not allowed to access these during your exam.
- Write as neatly as you can.
- Show all your work, if you want to receive partial credit.

Name:

Circle your section:

SCA (M, W evening)	A01 (9:30-10:20)	A02 (11-11:50)	A03 (12:30-1:20)
	A04 (12:30-1:20)	A05 (2-2:50)	A06 (3:30-4:20)

---

1. **[30 points]** Evaluate each expression and write down its *value*. Assume that (i) `concat` is a function that takes two arguments `a` and `b` and returns `a + b`, (ii) `isLen2` is a function that takes an argument `x` and returns `len(x) == 2`, and (iii) `L = ["Write", ["your", "name"], "your", "section", [["and", "your"], "student"], "ID"]`.

   (a) `L[:6:2]`

   (b) `L.index("your")`

   (c) `reduce(concat, map(len, L))`

Here is the list `L = ["Write", ["your", "name"], "your", "section", [["and", "your"], "student"], "ID"]` again.

(d) `L[1] + L[4][0]`

(e) `map(chr, map(concat, map(ord, ["a", "b"]), range(1,3)))`

(f) `reduce(concat, map(range, range(4)))`

(g) `"+".join(L[1] + L[4][0])`

(h) `[L[0]]+L[1][0].split("r")`

(i) `map(len, L).count(2)`

(j) `reduce(concat, map(concat, L, L)[1])`

2. [**40 points**] For each of the two programs below, write down the output produced by the program.

(a) What output does this program produce?

```
def area(x1, y1, x2 = 0.0, y2 = 0.0, scale = 1.0) :
        global width, height

        width  = abs(x1 - x2)
        height = abs(y1 - y2)
        return width * height * scale

# Main program
x1  = 2.0
y1  = 3.0
width  = abs(x1 - -x1)
height = abs(y1 - -y1)
size   = area(x1, y1, -x1, scale = 4.0)
print width * height, size
```

(b) What output does this program produce?

```
def isLen2(x):
    return len(x) == 2

def foo(s):
    upperCaseLetters = map(chr, range(ord("A"), ord("Z")+1))
    for ch in upperCaseLetters:
        s = s.replace(ch, "")
    return s

line = "The Bed is In the Top Floor, Next to The Red Box."
L = filter(isLen2, map(foo, line.split()))

for e in L:
    print e
```

3. [**40 points**] In this problem, you are given two partially completed programs. Your task is to complete each program.

   (a) This programs reads in a string `s` and a string `x`. It searches `s` and prints the position in `s` of the first occurrence of any length-2 (contiguous) substring of `x`. For example, suppose that the string `x` is "`bathrobe`". The length-2 substrings of "`bathrobe`" are "`ba`", "`at`", "`th`", "`hr`", "`ro`", "`ob`", and "`be`". Now suppose that the string `s` is "`Is Bridgetown, Barbados in the southern hemisphere?`". Then "`ba`" and "`th`" are the two length-2 substrings of `x` that occur in `s`. The substring "`ba`" occurs earliest in position 18 of string `s`. As a result, the program prints the message

   ```
   Found substring ba at position 18
   ```

   The following program is partially completed. It contains three lines that are incomplete — your task is to complete these.

```
s = raw_input("Enter the string to be searched: ")
x = raw_input("Enter the string of characters to search for: ")

i = len(s)

# Walk through the string x, picking out substrings of length-2

for ___________________________________:

    if x[pos:pos+2] in s: # check if the substring even belongs to s

        # if so find the position at which the substring occurs in s

        j = ____________________________

        if j < i :
            i = j

if i < len(s) :
    substring = ____________________________________

    print "Found substring " + substring + " at position " + str(i)
else :
    print "No search characters found in input string"
```

(b) Here is a partially completed program that attempts to read a file called `test.py` consisting of a Python program and produces as output the same program, but without any comments. More precisely, for any line that contains `#`, only the portion of the line that occurs prior to the character `#` is printed. The following program is missing two lines — your task into fill in the two blanks.

```
fin = open("test.py", "r")

for line in fin:
    if "#" in line:
        # Find the position of the first occurrence of hash symbol

        ------------------------------------------------------

        # Print only the portion of the line before first hash symbol

        ------------------------------------------------------
    else:
        print line.rstrip("\n")

fin.close()
```

4. [**40 points**] As input you are given a file that contains an arithmetic expression in each line. Your program should read each line, evaluate it, and output the answer to another file. You can suppose that the input file is called `expressions.txt` and each line contains an arithmetic expression involving additions and multiplications only. Also, each number in the expresssions is guaranteed to be a non-negative integer. For example, `expressions.txt` might be:

```
10*2 * 11 + 12*2 + 13*2* 3
110*2
10 + 2*3  *   4
10 + 12 + 13
225
```

As you can see from this example, different items in each expression are separated by 0 or more blanks.

Your program should write to an output file called `answers.txt`. For the example input file shown above, `answers.txt` would be the folowing, with each answer appearing in a new line.

```
322
220
34
35
225
```

(a) Write a function called `evaluateMult` that takes as its single parameter a string that is an arithmetic expression made up entirely of non-negative integers and the multiplication operator. The string will contain zero or more blanks between consecutive items in the expression. The function should evaluate the expression and return its value as an integer. For example, if the expression sent in as an argument to `evaluateMult` is `"3*4 *5 * 2"` then the function should return the integer `120`.
Hint: Split the given expression using `"*"`. The resulting list will contain numbers (as strings) that you need to multiply.

(b) Write a function called `evaluate` that takes as its single parameter a string that is an arithmetic expression made up entirely of non-negative integers, the multiplication operator, and the addition operator. The string will contain zero or more blanks between consecutive items in the expression. The function should evaluate the expression and return its value as an integer. For example, if the expression sent in as an argument to `evaluateMult` is `"3*4 *5 * 2 + 12*2 + 11 * 1"` then the function should return the integer `155`. The function `evaluate` should call `evaluateMult` repeatedly to complete its task. Also, `evaluate` should respect operator precedence, i.e., the multiplication operator is evaluated before the addition operator. For example, the expression `"3*4 + 7"` should evaluate to `19` and not to `33`.
Hint: Split the given expression using `"+"`. The resulting list will contain strings that are expressions containing only the multipication operator. The example from a few lines above can thus be turned into the list `["3*4 *5 * 2 ", " 12*2 ", " 11 * 1"]`. You can then use `evaluateMult` to evaluate each expression in the list and then add up the results. You can use the `map` function to do all of this in one line of code. But, feel free to use a for-loop, if you prefer.

(c) Now write the "main program" that reads each expression from a file called `expressions.txt`, evaluates it, and writes the value of the expression into a file called `answers.txt`. The main program should repeatedly call the function `evaluate`.
