Name: _____

Solutions

1. (1 pt.)
   - **Read all material carefully.**
   - *If in doubt whether something is allowed, ask, don't assume.*
   - You may refer to your books, papers, and notes during this test.
   - E-books may be used.
   - Computers are permitted but discouraged.
   - **Electronic and network resources must only be used as a passive library.**
   - Write, and draw, carefully. Ambiguous or cryptic answers receive zero credit.
   - Use class and textbook conventions for notation, algorithmic options, etc.
   - **Do not attach or remove any pages.** Questions must be answered only on the provided pages.

     Write your name in the space provided above.

2. (10 pts.) Using *Standard ML*, define a function `fi` that takes a pair of lists as argument and returns a list whose elements are the elements of the two lists picked alternately in order, starting with the first element of the first list. If the lists are of unequal lengths then the remaining elements from the longer list appear at the end (without any interleaving elements). The following examples illustrate the desired behavior of `fi`.

```
1  fi ([10, 11, 12, 13], [20, 21, 22, 23]) =
       [10,20,11,21,12,22,13,23]
2
3  fi ([10, 11, 12, 13], [20, 21]) = [10,20,11,21,12,13]
4
5  fi ([10, 11], [20, 21, 22, 23]) = [10,20,11,21,22,23]
6
7  fi (["Hello,", "World!"], ["I", "prefer", "to", "be"]) = ["Hello,
       ","I","World!","prefer","to","be"]
8
9  fi (["I", "prefer", "to", "be"], ["Hello,", "World!"]) = ["I","
       Hello,","prefer","World!","to","be"]
```

3. (5 pts.) Explain as precisely as possible why your SML definition of Question 2 is correct.

4. (5 pts.) Trace the operation of your SML code of Question 2 on last example provided there.

Ⓐ

```
fun fi (xs, nil) = xs
  | fi (nil, ys) = ys
  | fi (x :: xs, y :: ys) = x :: y :: fi (xs, ys)
```

Ⓐ *[There are several correct answers, as is typical for questions asking for code.]The SML code above uses pattern matching to first handle the two base cases of at least one of the lists being empty, when the result is simply the other list. When both lists are nonempty, the result is the cons of the head of the first list, the head of the second list and, recursively, the result of applying* `fi` *to the tails of the two lists.*

On the inputs of the last example, the action of the above definition of `fi` *is summarized below.*

Ⓐ

```
fi (["I", "prefer", "to", "be"], ["Hello,", "World!"])
= "I" :: "Hello" :: fi (["prefer", "to", "be"], ["World!"])
= "I" :: "Hello" :: "prefer" :: "World!" :: fi (["to", "be"], [])
= "I" :: "Hello" :: "prefer" :: "World!" :: ["to", "be"]
= ["I","Hello,","prefer","World!","to","be"]
```

5. (15 pts.) Provide a **complete JCoCo assembly language program** that

   - Reads a single newline-terminated string from *standard input*. (Everything up to but excluding the newline is the input string, which may include spaces.)
   - Writes a single integer n followed by a newline to *standard output*, where n is the number of spaces (character count) in the input string.

6. (10 pts.) Explain why your program of Question 5 is correct using a suitable combination of inline comments and separate text.

7. (10 pts.) Trace the operation of your program of Question 5 when the string `I␣Hi!␣␣y␣o␣` (followed by a newline) appears on standard input. (The string has 11 characters and spaces are depicted as ␣ for clarity.)

   Ⓐ *[There are several correct answers, as is typical for questions asking for code.]The code appears below, with material after a* `#` *on each line representing a comment for a human (not part of the program). Within that comment, the state of the operand stack is depicted before the second* `#` *(if any), as a list with the top-of-stack leftmost. The comments use a sample input of "*`f oo `*" as an example. The general plan of action of the program is to keep a count of the number of space characters on the stack (only), at the bottom. This counter, initialized to 0, is put on the stack. For later use, the* `list` *function is also put on the stack. Standard input is read using the* `input` *with the empty string as prompt. The* `list` *function is invoked and then an iterator on it is used in the main loop. In the loop body,* `ROT_TWO` *is used a couple of times to allow the counter to be incremented when the current item is a space character. When the loop ends, the counter value from the stack is printed using the* `print` *function, again using* `ROT_TWO` *to get the items in the proper order on the stack. The* `None` *result of printing on stack is returned from the main function.*

```
Function: main/0
Constants: "", " ", 0, 1
Globals: input, list, print
BEGIN
    LOAD_CONST 2          # [0] # initialize spaces-counter; leave it on stack.
    LOAD_GLOBAL 1         # [list, 0] function invoked later
    LOAD_GLOBAL 0         # [input, list, 0] # read line from stdin with "" as prompt...
    LOAD_CONST  0         # ["", input, list, 0]
    CALL_FUNCTION 1       # ["f oo ", list, 0] # convert string to list of chars...
    CALL_FUNCTION 1       # [["f", " ", "o", "o", " "], 0]
    GET_ITER              # [iter(above list), 0] set up loop
L0: FOR_ITER L1           # ["f", iter(...), 0]
    LOAD_CONST 1          # [" ", "f", iter(...), 0]
    COMPARE_OP 2          # [false, iter(...), 0] # is " " == "f"? etc.
    POP_JUMP_IF_FALSE L0  # [iter(...), 0] # if not, go to loop top to check next char
    ROT_TWO               # [0, iter(...)]
    LOAD_CONST 3          # [1, 0, iter(...)]
    BINARY_ADD            # [1, iter(...)]
    ROT_TWO               # [iter(...), 1]
    JUMP_ABSOLUTE L0      # [iter(...), 1] # ... and then go to loop top...
L1: LOAD_GLOBAL 2         # [print, 2] # out of loop, print and return from main...
    ROT_TWO               # [2, print] # value 2 is for "f oo " example.
    CALL_FUNCTION 1       # [None]
    RETURN_VALUE          # []
END
```

8. (10 pts.) Consider the following grammar (using *yacc/PLY* syntax):

```
S : A S B | c
A : a c a | a c A a
B : B b | b | b S
```

Does the sentence `acacacacbacacb` belong to the language of this grammar? If it does then provide a corresponding **leftmost derivation** and a **parse tree**, both *using class and textbook conventions*; otherwise, provide a **proof** (as precise as possible) that it does not.

Ⓐ *No, the given string does not belong to the language of this grammar. Proof: Consider (for proof by contradiction) a leftmost derivatition of the given string. For reference, number the rules of the grammar 1 through 7 (expanding the vertical-bar short-hand into multiple rules). The first step of the derivation must use rule 1 because if it uses the only other rule for the start symbol S, the derivation would end with just c as the string. In the resulting ASB, if A is expanded using rule 3, we get acaSB. At this point, the only way to get a c following the initial aca is to expand S using rule 2 because using rule 1 gives acaASBB which results in two consecutive as regardless of which rule is used to expand the A (and the desired string does not have any consecutive as). Retreating back to acaSB and using rule 2 to expand S yields acacB. At this point, regardless of which of the rules is used to expand B, the next terminal will be b so that the prefix of the derived string is acacb, which is not a prefix of the desire string. Therefore, no such derivation exists.*

9. (5 pts.) Consider the following regular expression. (The regular expression uses the textbook's conventions. Tokens `a`, `b`, and `c` are character literals.)

$$\texttt{a.b.(c.(a + b))}*$$

   (a) What is the minimum length of a string that matches this regular expression?

   (b) Why is that the minimum length?

   (c) Provide such a string.

   Ⓐ *The minimum length of a matching string is 2. The string* `ab` *matches this regular expression and is of length 2. There can be no shorter matching string because of the prefix* `a.b` *(with semantics* `a` *followed by* `b`*) in the regular expression, which is only matched by the string* `ab` *and nothing shorter (or different).*

10. (5 pts.) Provide two distinct strings of length 3 that match the regular expression of Question 12, or explain as precisely as possible why no such strings exist.

    Ⓐ *There are no strings of length 3 that match the regular expression, for the following reason: The first two letters of any matching string must be* `a` *and* `b`*, as explained in the previous answer, to match the prefix* `a.b` *of the regular expression. Following that prefix, we have* `.(c.(a+b))*` *with the semantics "zero or more instances of* `c` *followed by either* `a` *or* `b`*." The substring of which there must be zero or more instances is thus of length 2, which means only strings of length* $2k$ *for integer* $k > 0$ *can match. So no odd-length string (such as a length 3 string) can match.*

11. (5 pts.) Repeat Question 10 for length 4 instead of 3.

    Ⓐ *Strings* `abca` *and* `abcb` *both match the regular expression (for the reasons explained in the previous two answers).*

12. (10 pts.) Provide a context-free grammar (CFG) for the language defined by the following regular expression. (The set of strings that can be derived using the CFG must be the same as the set of strings matching the regular expression.) **Explain** briefly why your answer is correct.

    Ⓐ *[There are several correct grammars.] In the grammar below, the first two rules ensure that any derivation from the start symbol A must include the prefix* `ab`*. The third line (shorthand for three rules) ensures that the rest of the derivation generates zero or more instances of either* `ca` *or* `cb`*.*

$$\begin{aligned} A &\rightarrow aB \\ B &\rightarrow bC \\ C &\rightarrow caC \mid cbC \mid \epsilon \end{aligned}$$

13. (5 pts.) For each string of Question 10, either provide a leftmost derivation using the grammar of Question 12 or explain as precisely as possible why no such derivation exists.

Ⓐ *Consider any leftmost derivation with the grammar of Q. 12. The first step two steps must be* $A \overset{1}{\Rightarrow} aB \overset{2}{\Rightarrow} abC$ *(numbering rules sequentially 1–5). There are three rules for* C *and they result in either zero or two additional terminals. So it is not possible for a derivation to yield only three nonterminals and no string of length 3 can be generated.*

14. (5 pts.) Repeat Question 13 for the strings of Question 11 instead of those of Question 10.

    Ⓐ $A \overset{1}{\Rightarrow} aB \overset{2}{\Rightarrow} abC \overset{3}{\Rightarrow} abcaC \overset{5}{\Rightarrow} abca$. $A \overset{1}{\Rightarrow} aB \overset{2}{\Rightarrow} abC \overset{4}{\Rightarrow} abcbC \overset{5}{\Rightarrow} abcb$.