Homework 3: Introductory SML Programming

Due:

Since the purpose of this homework is to ensure basic skills in functional programming, this is an individual homework. That is, for this homework, you are to do the work on your own, not in groups.

For all SML programs, you must run your code with the SML interpreter. You must also provide evidence that your program is correct (for example, test cases). Hand in a printout of your code and your testing. Some questions just require an English answer.

Be sure to clearly label what problem each function solves, as some of the problems don't state the names required of the functions.

Following the suggestions of the programming track: most of the problems below exhibit polymorphism, there is a problem to illustrate a bit of the difference between functional and OO programming, and several that describe lazy evaluation. Finally, I have included a problem comparing SML to more conventional languages, and the problems as a whole illustrate how functional languages work without commands or variables, hence the following instruction.

You should program in a purely functional style (for example, don't use **`ref`**).

I don't expect you to do all of the suggested practice problems; but consider them as you read along in the book, give a moment of thought to each, and if you can't see how to do it, take the time to think about it. These may also be good for questions in discussion section.

1. (suggested practice) Read a tutorial on the concepts of functional programming. For example, chapter 13 in Watt's *Programming Language Concepts and Paradigms*. Or look at some of the books recommended in the "Introduction to the Literature" handout.

2. (40 points) Read an article in a journal or conference proceedings that is related to functional programming languages (such as SML) from the "Introduction to the Literature" handout. This time I want you to find an article that is either a survey article or one that describes research in programming languages. Don't report on a chapter from a book, or an article on software engineering or hardware. See Gary if you have questions about the suitability of an article. Write a short (1 or 2 page) review of the article.

   If you read a survey article, your review should clearly state: (a) the subarea surveyed and the major problems covered, (b) the main points (ideas) in the work surveyed, and (c) what you learned from it.

   If you read a research article, your review should clearly state: (a) what the problem is that the article claims to solve, (b) the main points including any claimed contribution vs. related work, (c) your opinion as to how good the solution to the problem was, and (d) what you learned from it.

   Please be nice to the class by not checking out a proceedings or journal volume from the library, but only making a copy of a relevant article, or reading it in the library.

3. (10 points extra credit only; programming, semantics) Write an answer to exercise 2.2 in Paulson's book.

4. (suggested practice) Do one or more of exercises 2.3, 2.4, 2.5 in Paulson's book.

5. (5 points) Do exercise 2.6 in Paulson's book (tuples).

6. (suggested practice) Do one or more of exercises 2.7, 2.9, 2.10 in Paulson's book

7. (5 points) Answer exercise 2.14 in Paulson's book (call-by-name and lazy eval).

8. (5 points) Answer exercise 2.19 in Paulson's book (local vs. nesting).

9. (5 points) Answer exercise 2.22 in Paulson's book (types).
10. (suggested practice) Do exercise 2.23 in Paulson's book.
11. (5 points) Answer exercise 2.24 in Paulson's book (types).
12. (5 points) (a) Do exercise 3.2 in Paulson's book (last element in list). (b) Is your code polymorphic?
13. (suggested practice) Do exercise 3.3 in Paulson's book.
14. (5 points) Do exercise 3.4 in Paulson's book (nth).
15. (suggested practice) Do one or more of exercises 3.6 and 3.7 in Paulson's book.
16. (10 points) Do problem 3.11 in Paulson's book (binary sum and product).
17. (suggested practice) Do one or more of exercises 3.12, 3.13, 3.16, 3.17 in Paulson's book.
18. (10 points) Do problem 3.18 in Paulson's book (determinant).
19. (suggested practice) Do one or more of exercises 3.20, 3.21, 3.22, 3.23 in Paulson's book.
20. (a, 10 points) Do problem 3.25 in Paulson's book (choose). (b, 5 points) Solve this same problem is Smalltalk.
21. (suggested practice) Do one or more of problems 4.1, 4.2 in Paulson's book.
22. (10 points) Do problem 4.3 in Paulson's book (shapes).
23. (suggested practice) Do one or more of problems 4.4, 4.5 in Paulson's book.
24. (5 points) Do problem 4.6 in Paulson's book (types).
25. (suggested practice) Do one or more of problems 4.8 in Paulson's book.
26. (5 points) Do problem 4.9 in Paulson's book (case).
27. (5 points) Do problem 4.10 in Paulson's book (eliminating case).
28. (suggested practice) Do one or more of problems 4.11, 4.12, 4.13, 4.14 in Paulson's book.
29. (5 points) Do problem 4.16 in Paulson's book.
30. (5 points) Do problem 4.18 in Paulson's book.
31. (suggested practice) Do one or more of problems 4.23, 4.24, 4.25, 4.26, 4.27 in Paulson's book.
32. (5 points) Do problem 4.28 in Paulson's book (sparse arrays).
33. (suggested practice) Do one or more of problems 4.29, 4.30, 4.31, 4.32, 4.33, 4.34, 4.35, 4.36, 4.37, 4.38 in Paulson's book.
34. (3 points) Do problem 5.1 in Paulson's book (fn notation).
35. (suggested practice) Do problem 5.2 in Paulson's book.
36. (4 points) Do problem 5.3 in Paulson's book (currying).
37. (suggested practice) Do one or more of problems 5.4, 5.5, 5.6 in Paulson's book.
38. (10 points) Do problem 5.7 in Paulson's book (minimum of a function).
39. (5 points) Do problem 5.8 in Paulson's book (sections and currying).

40. (suggested practice) Do one or more of problems 5.9, 5.10, 5.11 in Paulson's book.

41. (10 points) Do problem 5.12 in Paulson's book (map).

42. (10 points) Do problem 5.13 in Paulson's book (andf).

43. (10 points) Do problem 5.14 in Paulson's book (union).

44. (suggested practice) Do one or more of problems 5.15, 5.16, 5.17, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, 5.24 in Paulson's book.

45. (10 points) Do problem 5.25 in Paulson's book (α lseq lazy lists).

46. (10 points) Do problem 5.27 in Paulson's book (repeating lazy lists).

47. (10 points extra credit) Do problem 5.28 in Paulson's book.

48. (10 points; programming, semantics) Do problem 5.29 in Paulson's book (generalizing functionals to lazy lists).

49. (a, 10 points) Do problem 5.30 in Paulson's book (exponential). (b, 5 points) Briefly compare this to how you would solve the problem in a more conventional language such as C or Pascal.

50. (10 points extra credit each; programming) Do one or more of problems 5.31, 5.32, 5.33, 5.34, 5.35, 5.36, 5.37, 5.38, 5.39 in Paulson's book.

51. (suggested practice) Do one or more of problems 6.1, 6.2, 6.3, 6.4 in Paulson's book.

52. (10 points; programming) Do problem 6.5 in Paulson's book.

53. (10 points; programming) Do problem 6.6 in Paulson's book.

54. (10 points; programming) Do problem 6.7 in Paulson's book.

55. (10 points extra credit each; programming) Do one or more of problems 6.8, 6.9, 6.10, 6.11, 6.12, 6.13 in Paulson's book.

56. (15 points, extra credit only; programming, semantics) A set (or multi-set) can be described by a "characteristic function" (whose range is the booleans) that determines whether an element occurs in the set. For example, the function ϕ such that $\phi(\texttt{coke}) = \phi(\texttt{pepsi}) = \mathit{true}$ and for all other arguments x, $\phi(x) = \mathit{false}$ is the characteristic function for a set containing a `coke` and a `pepsi` and nothing else. Allowing the user to construct a set from a characteristic function gives one the power to construct sets that may "contain" an infinite number of elements (such as the set of all prime numbers).

    The elements of a set are of some equality type, `''elem` (see page 86 of Paulson's book). Let the polymorphic type constructor `set` be some polymorphic type that you decide on (you can declare this with something like the following).

    ```
    type ''elem set = ...;
    (* or perhaps something like *)
    datatype ''elem set = ...;
    ```

    The operations on sets are described informally as follows.

    - The function

      `with_characteristic: (''elem -> bool) -> (''elem set)`

      is such that `(with_characteristic f)`, returns a set that contains each x of type `''elem` if and only if `f(`x`)` is true.

    - The function

```
union: ''elem set * ''elem set -> ''elem set
```

is such that `union(b1,b2)`, returns a set containing all the elements of `b1` and `b2`.

- The function

  ```
  intersect: ''elem set * ''elem set -> ''elem set
  ```

  is such that `intersect(b1,b2)`, returns a set containing all x such that x is both an element of `b1` and of `b2`.

- The function

  ```
  member: ''elem set * ''elem -> bool
  ```

  is such that `member(b,e)`, returns true if `e` is in `b`, and false otherwise.