104

UNIVERSITY OF SWAZILAND

Faculty of Science
Department of Computer Science

Supplementary Examination, July 2011

Title of paper: **PROGRAMMING LANGUAGES**

Course numbers: **CS343**

Time allowed: 3 hours

Special requirements: Computer with Haskell interpreter.

Instructions:

- Answer question 1 (compulsory) and any 3 of the remaining 4 questions. Each question carries 25 marks.
- Save answers to question 1 in a file named *id*.hs with *id* substituted by your student ID.
- Questions 2 to 5 are to be answered in the provided examination folder.

THIS EXAMINATION PAPER SHOULD NOT BE OPENED UNTIL PERMISSION HAS BEEN GRANTED BY THE INVIGILATOR

105

Question 1 (compulsory)

Write Haskell code to answer the following 2 questions. Save both answers in your answer file. In the file, clearly indicate the line separating the code for (a) from the code for (b). the beginning of the second.

a) Write a function named *join* that works as shown in the following examples:

```
join ("Mba", ["ba", "ne"]) → ("Mbaba", ["ne"])
join ("a", ["b", "c", "dd"]) → ("ab", ["c", "dd"])
join ("now", ["here"]) → ("nowhere", [])
join ("Hello World", []) → ("Hello World", [])
```

Formally, *join* is defined to:

- take a single argument, that is a pair consisting of a string (*initial*) and a list of strings (*strings*), and
- return a pair whose data type is the same as that of the argument. If *strings* is the empty list, the result should be the same as the argument. Otherwise, the result should be the pair (*i*, *s*), where *i* is the string formed by appending *initial* and the first string of *strings*; and where *s* is *strings* without the first string. [12]

b) Write a function named *cat* that takes a list of strings as its only argument, and returns a string that is the concatenation all the given list's elements. (Concatenation means putting together the characters of a sequence of strings, creating one big string.) Some instances of how it should work are:

```
cat ["Mba", "ba", "ne"] → "Mbabane"
cat ["a", "b", "c", "dd"] → "abcdd"
cat ["now", "here"] → "nowhere"
cat ["Hello World"] → "Hello World"
cat [] → ""
```

The *cat* function is allowed, though not required, to call the *join* function defined in (a) above. [13]

Question 2

a) What *semantic gap* exists in programming? [1]

b) Discuss any three reasons for programming in high-level languages rather than in low-level languages. [6]

c) Distinguish between the following pairs:
   i. High-level and low-level languages.

   ii. Syntax and semantics.

   iii. Imperative and declarative paradigms. [6]

d) What is the meaning of operator fixity? [1]

e) Consider a language having 4 operators: ∧, ∨, > and <, that take numerical operands. Their syntactic properties are as follows:

Operator	*Precedence*	*Arity*	*Fixity*	*Associativity*
∧	0 (high)	1	Prefix	Right
<	1	2	Infix	Left
>	2	2	Infix	Right
∨	3 (low)	1	Postfix	Left

Fully parenthesize the following expressions:

i. 1 > 2 < 3 ∨ [2]

ii. ∧ ∧ 1 ∨ ∨ [2]

iii. (1 > 2 ∨) > ∧ 3 > ∧ 4 [3]

iv. 1 > ∧ (2 ∨) < 3 < ∧ 4 > 5 [4]

Question 3

a) Distinguish between primitive and user-defined data types, giving examples from a programming language you know. [2]

b)

i. Define *type safety*. [2]

ii. Distinguish between statically and dynamically typed languages. [2]

iii. Give an example of an unsafe operation that will be rejected by a typed language. [2]

iv. Although Haskell is a statically typed language, it does not require data types to be declared in most cases. Why not? [1]

c) Explain the 4 kinds of polymorphic operators. Your answer should include examples of polymorphism in languages you know. [16]

Question 4

a) Why do structured languages encourage the breaking up programs into short routines? [1]

b)

i. Describe what is done by the *goto* statement. In addition, explain why it is generally considered to be harmful. [4]

ii. Imperative programming languages offer control structures for *selection* and *repetition*. Give a complete example of each one, in a programming language you know. [4]

c)

i. Explain what the terms *inheritance, overriding* and *dynamic dispatch* mean in object-oriented languages. [6]

ii. Explain the problem of repeated inheritance . [6]

iii. What information does a C++ *vtable* contain, and how is it put to use? [4]

Question 5

a) Consider a Haskell function, `fn`, of 2 numeric arguments:

i. Re-write the following function call using infix notation: [1]

```
fn 10 20
```

ii. Supposing that function `gn` is defined by *currying* `fn` as follows, how many arguments does `gn` take and of what type? [1]

```
gn = fn 30
```

b) Define referential transparency and explain one of its benefits. [4]

c) Based on the following type signature, describe the Haskell function `fun` as completely as possible: [2]

```
fun :: [Integer] -> (Integer -> Bool) -> Bool
```

d) Define *tail recursion* and explain whether or not the following Haskell function is tail-recursive: [4]

```
numChars str =
  if str=="" then 0
  else 1 + numChars (tail str)
```

e) Prove that the following Lambda Calculus expression evaluates to 200, showing all steps: [6]

$$(\lambda x.\ ((\lambda y.\ x\ y)\ 10))\ ((\lambda x.\ (\lambda y.\ x*y))\ 20)$$

f) Consider a Prolog knowledge base concerning the towns of Swaziland and the regions in which they are located. Some of the facts in the knowledge base are listed below:

```
town(mbabane, hhohho).
town(manzini, manzini).
town(matsapha, manzini).
```

i. Write a query to find the region in which `matsapha` is located. [2]

ii. Write a rule `alltowns(Town)` that binds `Towns` to a list of all towns. [5]

***** END OF PAPER *****