98

UNIVERSITY OF SWAZILAND

Faculty of Science
Department of Computer Science

Main Examination, May 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

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).

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

```
smaller (1, [2, 3, 4, 5])  →  (1, [3, 4, 5])
smaller (5, [4, 3, 2, 1])  →  (4, [3, 2, 1])
smaller (3, [1, 5])        →  (1, [5])
smaller (3, [5, 1])        →  (3, [1])
smaller (3, [1, 5, 0])     →  (1, [5, 0])
smaller (3, [0, 5, 1])     →  (0, [5, 1])
smaller (3, [3])           →  (3, [])
smaller (3, [])            →  (3, [])
```

Formally, *smaller* is defined to:

- take a single argument, that is a pair consisting of a number (*small*) and a list of numbers (*nums*), and
- return a pair whose data type is the same as that of the argument. If *nums* is the empty list, the result should be the same as the argument. Otherwise, the result should be the pair (*s*, *n*), where *s* is the smaller between *small* and the first number of *nums;* and where *n* is *nums* without the first number. [12]

b) Write a function named *minimize* that takes a list of numbers as its only argument, and returns the smallest number in the list. You may assume that the given list is never empty.

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

100

Question 2

a) Discuss any two reasons for programming in high-level languages rather than in low-level languages. [4]

b) Distinguish between the following pairs:
   i. Expression and statement.

   ii. Compiler and interpreter.

   iii. Imperative and declarative paradigms. [6]

c) Define the following terms in relation to expression syntax:
   i. Precedence

   ii. Associativity [2]

d) Consider a language having 4 operators: $\wedge$, $\vee$, $>$ and $<$, that take numerical operands. Their syntactic properties are as follows:

Operator	*Precedence*	*Arity*	*Fixity*	*Associativity*
$\vee$	0 (high)	1	Postfix	Left
$<$	1	2	Infix	Left
$\wedge$	2	2	Infix	Right
$>$	3 (low)	1	Prefix	Right

Fully parenthesize the following expressions:

i. $1 < 2 \wedge 3 < 4$ [2]

ii. $> > 1 \vee \vee$ [2]

iii. $(> 1 < (> 2) < (> 3)) \vee < 4$ [3]

iv. $> 1 < 2 \wedge 3 \wedge 4 \vee \wedge (> 5 \vee < 6 \wedge 7)$ [6]

Question 3

a) Briefly describe any 5 kinds of user defined data types. [5]

b)

i. Define operator overloading. [2]

ii. Explain why operator overloading is impossible in untyped languages. [3]

c)

i. Define *type safety*. [2]

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

iii. Present the main arguments for and against dynamic typing. [7]

d) Define overloading polymorphism and parametric polymorphism. [4]

Question 4

a) Consider the following Pascal program. It is unstructured in some respects:

```
01      PROGRAM greet;
02      { This program prompts for a number, then displays a
03        greeting the given number of times. }
04      VAR
05        n: INTEGER;
06      LABEL
07        again;
08      PROCEDURE showGreetings();
09        BEGIN
10          writeln('Hello');
11          n := n - 1
12        END;
13      BEGIN
14        write('How many greetings? ');
15        read(n);
16      again:
17        showGreetings();
18        IF n > 0 THEN GOTO again
19      END.
```

i. Identify the unstructured aspects of this program, citing the line numbers where they appear. In addition, explain why you consider them to be harmful. [6]

ii. Re-write the program in a structured manner. Your program should display the same output as the original. [6]

b) Consider an object-oriented program for recording each student's test marks and calculating continuous assessment marks. Assume that such a program has a class of student objects. Using a single object of this class as an example, *list and describe the three main characteristics of objects*. [5]

c) Explain what the following terms mean in object-oriented programming:
i. Inheritance
ii. Dynamic dispatch
iii. Encapsulation
iv. Interface-implementation separation [8]

Question 5

a) Define the following terms:
   i. Higher-order function.
   ii. Tail-recursive function. [2]

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 -> Char -> [(Integer, Char)]
```

d) Re-write the following Haskell function using pattern matching: [2]

```
describe x =
  if x==0 then "zero"
  else "nonzero"
```

e) Prove that the following Lambda Calculus expression evaluates to 14, showing all steps: [6]

$(\lambda x.\ ((\lambda y.\ x*y)\ 2))\ ((\lambda x.\ x+((\lambda y.\ y)\ 3))\ 4)$

f) Consider a Prolog knowledge base concerning the names of some languages and their translators. Some of the facts in the knowledge base are listed below:

```
language(pascal).
language(python).
translator(pascal, turbo).
translator(pascal, delphi).
translator(python, python).
```

   i. Write a query to find the language for which `delphi` is a translator. [2]

   ii. Write a rule `selfnamed(Lang)` that succeeds when `Lang` is the name of a language having a translator that is also named `Lang` (e.g. `python` in the above knowledge base). [4]

   iii. Write a rule `babel(All)` that binds `All` to a list of all known languages. [3]

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