UNIVERSITY OF SWAZILAND

Faculty of Science

Department of Computer Science

SUPPLEMENTARY EXAMINATION 2007

Title of paper: PROGRAMMING LANGUAGES

Course number: CS343

Time allowed: Three (3) hours

Instructions: Answer any five (5) of the six (6) questions.

This examination paper should not be opened until permission has been granted by the invigilator.

Question 1

a) In a certain language, the following expression: a←↑b→c↓→d←e
when fully parenthesized, becomes: (a←((↑b)→((c↓)→d)))←e
Note: In this language, arrows represent operators while letters represent operands.

What, if anything, can be concluded from the above expression about the fixity, associativity and relative precedence of the 4 operators in this language?

[7]

b) Give a brief explanation of *axiomatic semantics*.

[3]

c) Describe any 5 kinds of user defined data types. For each kind, include an example of how such a type is defined in Pascal or C++.

[10]

Question 2

a) Describe the components that make up an *activation record.*

[4]

b) Draw a series of labelled diagrams showing the state of the program stack and associated registers *immediately after* each occurrence of *exit from function* `f` over the lifetime of the following C++ program:

```
int f(int x) {
  int y = x - 1;
  if (y == 0) return 0;
  else return 1 + f(y);
}

int main() {
  int x = f(3);
  return 0;
}
```

[16]

Question 3

a) What are the main prescriptions of structured programming, and why are they recommended?

[8]

b) Mention 2 features of Pascal that permit unstructured programming practices.

[2]

c) Explain the following statement: "In object oriented languages, classes not only act as ADTs, they also provide encapsulation and interface-implementation separation."

[6]

d) Explain the problem of *repeated inheritance* in object oriented languages.

[4]

Question 4

a) Explain each of the following terms and their relevance to functional programming: *referential transparency*, *type inference* and *tail recursion.*

[6]

b) Name the 2 main components of any logic programming system, and state the purpose of each.

[2]

c) For each of the following unification queries in Prolog, state whether or not the pair of terms can be unified. If yes, give the values that will be bound to the variables.

- `p(X) = 2.`
- `p(X, Y) = p(5, 6).`
- `p(X, X) = p(5, 6).`
- `p(p(X)) = p(q(1)).`
- `p(X, q(Y)) = p(q(1), q(2)).`

[5]

d) Draw the search tree for the following Prolog query:

```
b(X).
```

Assume that the following predicates have already been entered into Prolog:

```
a(1).
a(2).
a(3).
b(X) :- a(X), X > 1.
```

[7]

Question 5

a) State the meaning of the following Haskell type signature:

```
q5a :: String -> [[Integer]] -> Bool
```

[3]

b) Function `q5b`, defined below in Haskell, takes a list parameter and returns an integer. In *5 words or fewer* state what this returned integer means.

```
q5b a =
  let b c = not (null c);
      d e = if b e then tail e
            else e
  in length (takeWhile b (iterate d a))
```

[3]

c) Write a recursive Haskell function named `factorial` that takes a non-negative integer parameter `n` and returns the factorial of `n`.

[4]

d) Write a Haskell function named `pairs` that takes a string parameter and returns a list of all adjacent pairs of characters found in the given string. Assume that the given string will have a length of at least 2 characters.

For example, the value of the expression (`pairs "UniSwa"`) must be the following:

```
["Un", "ni", "iS", "Sw", "wa"]
```

[8]

e) Write the type signature of the `pairs` function mentioned in question d) above.

[2]

Question 6

a) Write a recursive Prolog predicate `fib(N, X)` that binds `X` to the N^{th} member of the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, ... (where the 1st and 2nd members are by definition 1, and each remaining member is the sum of its 2 predecessors). Assume that `N` will always be given as a positive integer.

[8]

b) Write a recursive Prolog predicate `rev(L, R)` that binds `R` to a list that is the reverse of the given list `L`. *Do not* use the built-in `reverse` predicate.

[7]

c) Write a recursive Prolog predicate `double(L, R)` where `L` is assumed to be a list of integers. The predicate must bind `R` to a list consisting of each element of `L` multiplied by 2.

For example, if `L` is `[3, -7, 0]` then `R` must be bound to `[6, -14, 0]`.

[5]