NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1, 2002/2003
CS2104 - PROGRAMMING LANGUAGE CONCEPTS

November 2002 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four(4)** long questions and comprises **fifteen(15)** pages.
2. Answer **ALL** questions.
3. **ALL** answers must come with the correct explanations. *There is no credit for blind guessing.*
4. Answer **ALL** questions within the space provided in this booklet.
5. This is an **OPEN BOOK** examination.
6. **Please write your Matriculation Number below.**

MATRICULATION NO.:

(This portion is reserved for the examiner's use only)

Question	Marks	Remark
Question A 11		
Question B 14		
Question C 13		
Question D 12		
Total 50		

A. **Language Syntax and Semantics** $(3 + 4 + 4) = 11$ marks

1. Consider the imperative language IMP introduced in class. This language has boolean and natural number datatypes and allows arithmetic expressions containing the addition (+) and multiplication operator (*) (refer lecture notes). Now suppose, I also allow the subtraction operator in arithmetic expressions *i.e.* I add the rule $A \to A - A$ to the BNF grammar of arithmetic expressions.

   (a) How will this change the set of values to which any arithmetic expression can evaluate to ?

   (b) Will this change any of the existing operational semantics rules of the IMP programming language (*i.e.* the operational semantics rules discussed in class) ?

2. Consider the following two grammars for expressions.

$$
\begin{array}{ll}
E \rightarrow E + T \mid T & \quad E \rightarrow T + E \mid T \\
T \rightarrow T * P \mid P & \quad T \rightarrow P * T \mid P \\
P \rightarrow (E) \mid i & \quad P \rightarrow (E) \mid i
\end{array}
$$

(a) Are the two grammars equivalent (*i.e.* represent the same set of expressions) ? Explain your answer.

(b) Will the two grammars remain equivalent after the addition ('+') operator is replaced by the subtraction ('-') operator in the first production rule of E in both the grammars ? Explain.

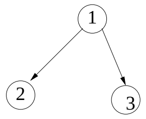
3. Prove the following Hoare triple. <> stands for *not-equal-to*. You will need to construct a nontrivial loop invariant first.

```
{n > 0}
i := n;
sum := 0;
while (i <> 0) do
    sum := sum + 2;
    i := i - 1
endwhile
{ sum = 2n}
```

B. **Logic Programming** $(3 + 3 + 3 + 5) = 14$ marks
Consider the following Prolog predicate which defines the reachability of nodes in a graph. Thus `reach(X, Y)` is true if and only if `Y` is reachable from `X`.

```
reach(X, Y) :- arc(X, Y).
reach(X,Y) :- arc(X, Z), reach(Z, Y).
```

The graph is encoded as facts, *e.g.* the graph given below



is encoded via the two facts (one each for the two edges in the graph)

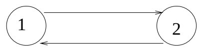
```
arc(1, 2).          arc(1,3).
```

1. Construct the Prolog search tree of the query `reach(1, X)` for the above graph.

2. Suppose the definition of the `reach` predicate is changed to the following (equivalent) definition. Will there be any change in the search tree constructed by Prolog for the query `reach(1, X)` ?

```
reach(X, Y) :- arc(X, Y).
reach(X,Y) :- reach(X,Z), arc(Z,Y).
```

3. Consider the following graph



What can you say about the search tree of the query `reach(1, X)` in this graph using the definition of `reach` given in question B.1 ?

4. Consider the following predicate `mystery`

```
mystery(X, [X|Xs], Xs) :- !.
mystery(X, [Y|Ys], [Y|Zs]) :- mystery(X, Ys, Zs).
```

(a) If the above program is used for queries of the form mystery(3, [1,2,3,4], Y) (*i.e.* queries where the first argument is a ground term, the second argument is a list of ground terms and the third argument is a variable) then what is the outcome ? Explain what the `mystery` predicate is doing for such queries, rather than simply constructing the search tree for some example queries.

(b) Will your answer to question (a) change if we remove the cut in the first clause ?

C. **Expressions, Functions, Functional Programming**
$(4 + 2 + 3 + 4) = 13$ marks

1. Infer the types of the functions `new1` and `new2`

   - `fun new1 g f x = g x (f x)`
   - ```
     fun new2 [] = []
     | new2 (x::xs) = 1 :: new2 xs ;
     ```
     ```

2. Consider the following higher-order function `reduce`.

```
fun reduce(f, nil, a) = a
|   reduce(f, x::xs, a) = f(x, reduce(f, xs, a));
```

This defines `reduce(f, [3,2,1], 0)` as `f(3,f(2, f(1, 0)))`. Construct a different function called `opp_reduce` which reduces the elements in the opposite order. Thus `opp_reduce(f, [1], 0)` should be same as `f(0,1)`, `opp_reduce(f, [2,1], 0)` should be same as `f(f(0,1), 2)`, `opp_reduce(f, [3,2,1], 0)` should be same as `f(f(f(0, 1), 2), 3)`, and so on. Use SML syntax.

3. As you probably know, the series of Fibonacci numbers are defined as:

   fib(0) = 1
   fib(1) = 1
   fib(n) = fib(n-1) + fib(n-2) if $n > 1$

   Consider the following function definition which correctly computes the nth Fibonacci number, given a number $n > 0$.

```
fun fib(n) = if n < 2 then 1 else fib(n-1) + fib(n-2);
```

   This function is very inefficient for computing fib(n) for large positive values of n, such as computing `fib(100)`. Explain why.

4. Consider the following expression with side-effects. What is the value obtained by its left-to-right and right-to-left evaluations ? Assume a= 1, b = 2, c =3 before the evaluation of the expression.

$$a + b * (a := a + 1) - (b := c)$$

D. **Imperative and Object Oriented Programming**
(4 + 4 + 4 = 12 marks)

1. Consider the following pseudocode of class definitions using virtual functions.

```
Class  A{
Public:
Virtual void f() { print  'a'}
}

Class B:A{
Public:
       Void  f() {print 'b'}
}
```

```
Class C:B{
Public:
      Void  g(){ print 'c'}
}

main()
{   B x;
    C y;
    x.f(); y.f();
}
```

What are the two characters printed ? Explain your answer.

2. You are familiar with the column-major order of array layout for 2-dimensional arrays (*i.e.* in the linear layout of the array, the first dimension increases rapidly and wraps around, whereas the second dimension increases slowly.) Thus in column major of an array A[1..M,1..M] of integers we will have a layout

$$A[1,1], A[2,1], \ldots, A[M,1], A[1,2], A[2,2], \ldots, A[M,2], \ldots$$

(a) How can we extend this notion to obtain a similar layout for k-dimensional arrays ?

(b) Now consider a k-dimensional array A of integers. The lower and upper bounds of array index in each dimension are -5 and 25 respectively. Each integer occupies 4 bytes, the address of the first element $A[1, 1, \ldots, 1]$ is 0. What is the address of the array element $A[i_1, i_2, \ldots, i_k]$?

3. Consider the following pseudocode where all parameters are passed by name.

```
function mystery_proc(f, i)
begin
   integer tmp;
   tmp := 0;
   for i:= 1 to 10 do  tmp := tmp + f
   return tmp;
end
```

Assume that A is an array of integers. Explain the computation performed by invoking the above function as `mystery_proc(A[K],K)`. What value will be returned ?