PAPER CODE NO.	EXAMINER : Grant Malcolm	TEL. NO. 794 6794
COMP205	DEPARTMENT : Computer Science	

THE UNIVERSITY
of LIVERPOOL

JANUARY 2003 EXAMINATIONS

Bachelor of Arts : Year 2
Bachelor of Arts : Year 3
Bachelor of Engineering : Year 2
Bachelor of Science : Year 1
Bachelor of Science : Year 2

COMPARATIVE PROGRAMMING LANGUAGES

TIME ALLOWED : Two hours

INSTRUCTIONS TO CANDIDATES

Answer **all** questions in Section A and **four** questions only in Section B.

If you attempt to answer more questions than the required number of questions (in any section), the marks awarded for the excess questions will be discarded (starting with your lowest mark).

THE UNIVERSITY
of LIVERPOOL

Section A

Each question in this section is worth 4 marks. Answer all questions in this section.

1. For each of the following programming paradigms, name one example of a programming language from that paradigm:
   (a) the object-oriented paradigm
   (b) the imperative paradigm
   (c) the functional paradigm
   (d) the logic paradigm

   (i.e., name an object-oriented language, an imperative language, etc.).

2. Give three major characteristics of the imperative paradigm.

3. Give three major characteristics of the declarative paradigm.

4. Consider the following fragment of C code:

```
int *p, n=0;
p = &n;
printf("%d, ", ++(*p));
printf("%d",n++);
```

   What would you expect the output to be?

5. What is an Abstract Data Type?

6. What is meant by 'referential transparency'?

7. Why can `goto`-statements be considered harmful?

8. Consider the following Haskell definitions:

```
sumAll [] = 0
sumAll (x:xs) = x + sumAll xs
```

   Define an equivalent function to `sumAll` using `foldl`.

9. What are the potential benefits of enumerated types?

10. Consider the following Horn-clause declarations:

```
works(gregor,biscuit_factory).
works(joseph,car_plant).
works(grace,biscuit_factory).
colleague(X,Y) :- works(X,Z), works(Y,Z).
```

    Give *all* possible solutions to the query `?- colleague(A,B)`?

THE UNIVERSITY
of LIVERPOOL

Section B

Each question in this section is worth 15 marks. Answer four questions from this section.

1. (a) What is the difference between a function and a procedure? **[3 marks]**

   (b) Briefly describe the difference between a *formal* and an *actual* parameter. **[3 marks]**

   (c) In Ada, what are the differences between `in`-parameters, `out`-parameters, and (`in out`)-parameters? **[4 marks]**

   (d) What is meant by 'call by value' parameter-passing, and how is it different from 'call by constant value' parameter-passing? **[5 marks]**

2. Give C functions for each of the following operations on one-dimensional arrays; in each case, the array (or arrays) and its length (or their lengths) should be formal parameters:

   (a) Add `1` to each element in an array of integers **[5 marks]**

   (b) Given an array of integers, return the array consisting of all elements less than `50`. **[5 marks]**

   (c) Concatenate two arrays: for example, given an array with elements `1`, `2` and `3`, and an array with elements `4` and `5`, return the array with elements `1`, `2`, `3`, `4` and `5`. **[5 marks]**

3. (a) The colours in a rainbow are: red, orange, yellow, green, blue, indigo and violet. Give a set of data type definitions in C that would allow you to construct a linked list of all the colours in a rainbow. **[7 marks]**

   (b) Bearing in mind the advantages of functional decomposition, give C code that constructs a linked list containing all the colours of the rainbow, in the order given above. **[8 marks]**

4. (a) What is meant by a *pattern* in Haskell? **[3 marks]**

   (b) What is *pattern-matching*? Illustrate your answer by saying whether the following expressions match the pattern `(a : b : x, c : [])`:

   - `( [(2,3), (4,0)], [1] )`
   - `( [2,3,4], [1,2] )`

   If the expression does match the pattern, say which subexpressions are matched to the variables; if the expression does not match the pattern, explain what is different. **[7 marks]**

   (c) The following Haskell definition uses pattern-matching to define a function that adds all the elements in a list of numbers:

```
sumAll []     =  0
sumAll (x:xs) =  x + sumAll xs
```

   Give an equivalent definition that does not use pattern-matching but does use `head` and `tail`. **[5 marks]**

5. (a) Give a Haskell definition of a polymorphic data type `BTree a` of binary trees, where a binary tree is either a 'leaf' containing an element of the parameter type `a`, or a 'fork' consisting of two binary trees. **[5 marks]**

   (b) Give a Haskell definition of a polymorphic map function that takes a function and a binary tree, and returns the tree formed by applying the function to each leaf in the tree. Include a type declaration of the function in your definition. **[4 marks]**

   (c) Define a function `BTreeFold` that captures the general form of recursive functions on binary trees (in the same way that `foldl` does for lists). The function should have polymorphic type

```
(b -> b -> b) -> (a -> b) -> (BTree a) -> b
```

   [6 marks]

6. (a) Briefly indicate how the λ-calculus can provide an operational semantics for Haskell programs. **[3 marks]**

   (b) What is meant by α-conversion in the λ-calculus? **[3 marks]**

   (c) What is meant by β-reduction in the λ-calculus? **[4 marks]**

   (d) Give a step-by-step reduction of the λ-term

$$\lambda y.((\lambda x.(\lambda y.(x\ y)))\ (\lambda z.(z\ y)))$$

   indicating whether each step is an instance of α-conversion or of β-reduction. **[5 marks]**