PAPER CODE NO.
COMP505

EXAMINERS : L. Gasieniec, V. Tamma
DEPARTMENT : Computer Science
TEL.NO : 3686, 6797

THE UNIVERSITY *of* LIVERPOOL

JANUARY 2001 EXAMINATIONS

Degree of Master of Science :

ALGORITHM DESIGN AND IMPLEMENTATION

TIME ALLOWED : Two Hours

INSTRUCTIONS TO CANDIDATES

- candidates will be assessed on their best four answers
- if you attempt to answer more than the required number of questions, the marks awarded for the excess questions will be discarded (starting with your lowest mark)

Question 1

1.A What is the *order* of complexity of an algorithm?
What is meant by *polynomial* order and *exponential* order? **[5 marks]**

1.B Consider function $f(n)$ defined on positive integers, s.t. $f(1) = 1$, $f(2) = 3$, and $f(n) = 2f(n-1) - f(n-2)$, for all $n \geq 3$. What is the order of complexity of a recursive procedure finding a value of $f(n)$? Can we compute the value of $f(n)$ more efficiently? **[12 marks]**

1.C What is the main characteristic of the *divide and conquer* method of algorithm design? **[3 marks]**

1.D Which sorting algorithm – *quick-sort*, *merge-sort*, or *bubble-sort* – has been used to sort an input array of integers:

$$A[1..15] = [4, 9, 6, 11, 7, 10, 14, 8, 12, 13, 2, 15, 3, 5, 1]?$$

[4,	9,	6,	11,	7,	10,	14,	8,	15,	13,	2,	12,	3,	5,	1]
[4,	6,	7,	2,	3,	5,	1,	**8**,	9,	11,	10,	14,	15,	13,	12]
[2,	3,	1,	**4**,	6,	7,	5,	**8**,	9,	11,	10,	**12**,	14,	15,	13]
[1,	**2**,	3,	**4**,	5,	**6**,	7,	**8**,	9,	**10**,	11,	**12**,	13,	**14**,	15]

Justify your answer. **[5 marks]**

Question 2

2.A State the Church-Turing Thesis. What evidence justifies the belief in its validity? **[5 marks]**

2.B What is meant by saying that a *class* of problems is *infeasible*? Give an example of an infeasible problem, explaining why it is infeasible.

[5 marks]

2.C Solve the following instance of the *Knapsack problem*

Type	A	B	C
Size	1	2	4
Value	1	3	7

where the capacity of a the knapsack is 7.
What is the principle of the *dynamic programming* method?

[12 marks]

2.D What is the *Halting Problem*? How difficult is this problem?

[3 marks]

Question 3

3.A Give the parse tree for the program:

```
begin x :=5 ; repeat x := x - 2 until x < y; end;
```

which is derived from the production for *program* in grammar below. (With the usual definition of *identifier* and *constant*). **[8 marks]**

```
<program> ::= <block>
<block> ::= begin <statement_sequence> end;
<statement_sequence> ::= <statement> | <statement> <statement_sequence>
<statement> ::= <assignment_statement> | <block> | <repeat_statement>
<assignment_statement> ::= <identifier> := <expression>
<repeat_statement> ::= repeat <statement_sequence> until <boolean_condition>
<expression> ::= <term> | <expression> <adding_operator> <term>
<adding_operator> ::= + | -
<term> ::= <identifier> | <constant> | (<expression>)
<boolean_expression> ::= <expression> <comparison_operator> <expression>
<comparison_operator> ::= < | >| <= | >= | =
```

3.B State in BNF the syntax of the conditional statement of Java and the syntax of the indefinite loop with post-condition of Java. **[5 marks]**

3.C What is the difference between an indefinite loop with *precondition* and an indefinite loop with *postcondition*. **[7 marks]**

3.D Define the syntax and the semantics of a generic programming language. **[3 marks]**

Question 4

4.A Describe the process of *functional decomposition* in program design. **[5 marks]**

4.B In choosing a suitable functional decomposition what properties should be demonstrated by the modules? **[3 marks]**

4.C What is the difference between a procedure and a function? **[5 marks]**

4.D What is meant by *aliasing*? **[5 marks]**

4.E In high-level programming languages four different ways to pass parameters to procedures (calling methods) are made available. Describe them all. **[7 marks]**

Question 5

5.A Give the definition of *data type* and define *hardware data types*, *virtual data types* and *abstract data types* **[7 marks]**

5.B What is a *strongly typed* language? **[5 marks]**

5.C The binding of the name of an array to its required amount of storage is called *name-sized binding*. Describe the two possible types of bindings. **[5 marks]**

5.D A binary tree is represented as a set of nodes. Each node is represented by a structure of `tree_node` which has three fields: a data field and two fields that are, respectively, the pointers to the left hand sub-tree of the node and to the right hand sub-tree of the node. Write a pseudo-code procedure, whose parameter is a pointer to the root node of the tree, that visits each node of such a tree in left to right, depth-first, pre-order. This procedure has to be recursive. **[8 marks]**

Answers to question 1

1.A The order of the Algorithm If the approximate time taken by an algorithm is about $f(n)$ and $f(n)$ becomes a better approximation as n gets larger, we say that that the complexity of the algorithm is of order $f(n)$. We sometimes also apply the concept to the storage space needed.

An algorithm is of polynomial order if its order $f(n)$ is a constant power of n, i.e. $f(n) = O(n^c)$, where c is a constant. For example, the bubble sort method has order $O(n^2)$.

An algorithm is of exponential order if its order $f(n)$ is a linear power of a constant, i.e. $f(n) = O(c^n)$, for some constant c. For example, the recursive algorithm generating Fibonacci numbers has order $O((\frac{1+\sqrt{5}}{2})^n)$.

1.B Function $f(n)$ is defined as follows: $f(1) = 1$, $f(2) = 3$, and $f(n) = 2f(n-1) - f(n-2)$, for all $n \geq 3$. Notice that the definition of function $f(n)$ is very similar to the definition of a sequence of Fibonacci numbers. I.e. the values of $f(1)$ and $f(2)$ are given explicitly, and the value of $f(n)$ (for $n \geq 3$) depends on values of $f(n-1)$ and $f(n-2)$. Thus the number of recursive calls of a procedure that computes a value of $f(n)$ and a procedure that computes F_n is the same, and we know that it is exponential in n.

One can compute a value of $f(n)$ more efficiently using the *dynamic programming* method, i.e. storing all consecutive values of $f(i)$, for $i = 1, 2, .., n$. Using this method we avoid recursive calls (each value of $f(i)$ is computed only once) and the order of the algorithm is $O(n)$.

More careful analysis of the definition of $f(n)$ shows that $f(n+1) - f(n) = f(n) - f(n-1)$, for all $n \geq 3$. This means that the consecutive values of function $f(n)$ form an arithmetic progress with a step of size $f(2) - f(1) = 3 - 1 = 2$. And knowing that we can conclude that $f(n) = 1 + 2(n-1) = 2n - 1$.

1.C Divide and conquer In this recursive method the strategy is to split the problem into smaller parts, which are usually of the same type as the original but involve less data. Each of the parts is treated separately and then the solutions one the two (or more) parts are combined in some way to get a solution to the complete problem.

1.D An input array of numbers has been sorted by the quick-sort algorithm. In the first round element 8 has been selected to split the sequence of all numbers into one subsequence of all numbers that are smaller than 8 (stored to the left of 8) and another subsequence of all numbers that are greater than 8 (stored to the right of 8). Later, both subsequences are split similarly by numbers 4 and 12, when the next level of recursive calls is applied, etc.

Answers to question 2

2.A The Church-Turing thesis states that all reasonable definitions of algorithm that anyone will ever make will turn out to be equivalent to the definitions we already know, which include the algorithms that can be specified by computer programs.

The justification is that many systems for specifying programs such as Turing machines, Computer programs and Markov algorithms have been proved to be equivalent, and that when people have tried to express any algorithm in computer code it has always been possible to do so if the algorithm appeared to be possible on the other considerations.

2.B The infeasible problems are those for which the computing time (or the data space required) increases so quickly with the problem size, that it cannot be envisaged that there will ever be computers sufficiently powerful to solve the problem for large problem size n.

The Travelling salesman problem is a famous problem which is very time consuming to solve. Given a set of n cities and the distances between them, the problem is to find a route of minimum length which visits each city exactly once and returns to the starting point. There are $(n-1)!$ different routes to check, so the work involved in an exhaustive search of all routes is of order of $(n-1)!$ operations. This increases so fast that any such method is completely infeasible for all but very small values of n. Despite a very large research effort no algorithm is known which is significantly better than this.

2.C

Type&Size	1	2	3	4	5	6	7
A	1^{1A}	2^{2A}	3^{3A}	4^{4A}	5^{5A}	6^{6A}	7^{7A}
A|B	1^{1A}	3^{1B}	4^{1A1B}	6^{2B}	7^{1A2B}	9^{3B}	10^{1A3B}
A|B|C	1^{1A}	3^{1B}	4^{1A1B}	7^{1C}	8^{1A1C}	10^{1B1C}	11^{1A1B1C}

The principle of dynamic programming method is similar to the one of divide and conquer, in which a large problem is split up into smaller problems which are solved independently. In dynamic programming this principle is carried to an extreme, i.e. when we do not know exactly which smaller problem to solve, we simply solve them all, then store the answers away to be used later in solving

larger problems.

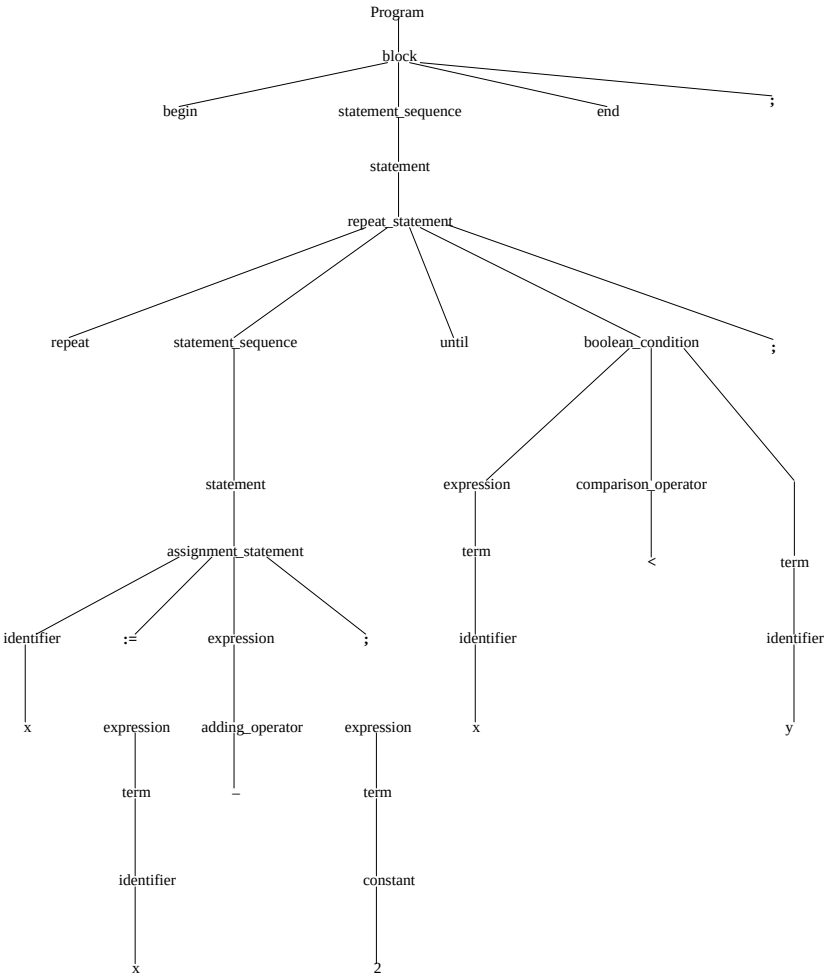
2.D The Halting Problem is the problem of taking a student's program (or indeed any other program) and its data as input, and analysing them to say whether the program given that data will loop for ever or would eventually terminate. Unfortunately it can be proved that it is impossible to write such a useful program.

Answers to question 3

3.A A parse tree is obtained by replacing a syntactic category by the right hand side of one of its production and showing the derivation in hierarchical form. The parse tree for the expression:

```
begin x :=5 ; repeat x := x - 2 until x < y; end;
```

in the given language is given below:



3.B The Java conditional statement is of the form:

```
<conditional_statement> := if (<Boolean_expression>) then <statement_sequence>;
                                                    [else <statement_sequence>;]
```

The Java indefinite loop with post-condition is of the form:

```
<indefinite_loop_with_post_condition> := do <statement_sequence> while
                                         (<Boolean_expression>);
```

3.C The indefinite loop with *precondition* has the pseudo-code form:
while *condition* **do** *statement sequence* while in Java is expressed by the statement:

```
<indefinite_loop_with_pre_condition> := while (<Boolean_expression>) do
                                          <statement_sequence>;
```

It first checks whether the condition is true, then if this is the case the statement sequence is executed. So, if the condition is never verified then the statement sequence is never executed.
The indefinite loop with *postcondition* has the pseudo-code form: **do** *condition* **while** *statement sequence* while in Java is expressed by the statement:

```
<indefinite_loop_with_post_condition> := do <statement_sequence> while
                                           (<Boolean_expression>);
```

In this case the statement sequence is executed before the termination condition is checked. Therefore it has to be used only in those cases such that it is desirable that the statement sequence is executed at least once.

3.C *Syntax*: The syntax of a programming language specifies the form in which a program (and all its sub-units) should be specified. *Semantics*: Semantics denotes the meaning which is to be given to the various syntactic units of a program, and to the program itself.

Answers to question 4

4.A The process known as *functional decomposition* decomposes a large program into a hierarchy of smaller modules, each of which should carry out its own well specified function. Some software programs can be made of a a considerable number of lines, thus making the design of the program itself quite difficult. If the problem to be solved is large it is impossible (or at least inadvisable) to start to write down the detailed steps needed without first decomposing the problem into a number of smaller subproblems which are so small that the complete solution can be easily comprehended. The whole solution is then constructed by combining together these small modules into a tree-like structure. Some modules will do little but organise the calling into operation of other modules. At the upper level the modules will be performing large steps in the computation (most of the detail being done by their submodules) but lower down in the decomposition the modules will be carrying out simple detailed steps.

4.B The modules obtained by applying correctly the design rule of functional decomposition should demonstrate a *high degree of cohesion* and a low *degree of coupling*. The concepts of *cohesion* and *coupling* describe conflicting principles that lead to program design compromise.

Cohesion relates to the functional independence of the module and to the performance of a single ”task” within the module. The overall objective is to separate the problem in to parts such that each part becomes a module that will be designed to complete an independent task. It allows a programmer to write a module of a larger program and test it independently. This module become a ”black box” in a large program.

Coupling allows modules to be connected by an interface, which enables the programmer to transfer data from one module to another. Three coupling techniques are parameters used in functions, function names and data that all the modules can access. Coupling allows for communication between modules. It allows programmers to use necessary variable names without loosing the cohesion in the module.

4.C Both *procedures* and *functions* are comprised of a sets of instructions that perform specific tasks. A *Procedure*, in its generic sense, is a program unit written independently of the main program and associated with it through a transfer/return

process. Once the procedure is invoked or called the control is transferred to the procedure that carries out its task and then returns the control to the main program. Data is transferred to and from the main program and the procedure via *parameters list*. A *function* is a program unit similar to a procedure except that a value is always transferred back to the main program as "the value of the function" rather than via the parameter list. That is the value computed by the function is associated to the function's name in a way similar to the one that a value is associated with a variable.

4.D The following four calling methods are made available:

- *Call by value*: When this method is being used, on obeying a call to the procedure, the actual parameter is evaluated and its value is used to initialise the corresponding formal parameter. Inside the procedure it may or may not be possible to assign a new value to the formal parameter.

- *Call by result:* Result parameters are used for passing values out of a procedure. On leaving the procedure the value of its formal parameter, as calculated by the procedure, is assigned to the actual parameter. The actual parameter must thus be a variable; its address is normally calculated at procedure entry. The corresponding formal parameter must be a variable of the same type, which is not initialised.

- *Call by Value-Result*: These parameters can feed a value into a procedure and pass one out of it. They are like result parameters, but the formal parameter is initialised form the value of the actual parameter.

- *Call by reference*: When a reference formal parameter is referred to in the procedure body, the variable which is accessed is that of the corresponding actual parameter; there is no storage space in the procedure which is allocated to the storage of the value of the formal parameter. Thus the formal parameter is an alias for the actual parameter. The way this is implemented is by passing the address of the actual parameter into the procedure; this address is then used as the address of the formal parameter.

4.E The problem known as *aliasing* occurs when the same variable can be referred to by two different names at the same place in the program. This can happen if a procedure has a formal parameter x and also uses a global variable p of the same type. If now the procedure is called with the variable p as the actual parameter corresponding to x then we have aliasing because within the procedure, both pand x refer to the same variable.

Answers to question 5

5.A Data type refer to both the interpretation given to a bit pattern and the operations that can be performed on the pattern. We distinguish the following data types:

- *Hardware data types*: are the types provided directly by the CPU. They consist of: integers, floating points and addresses;
- *Virtual data types*: are the types provided directly by a high level computer language (or its compiler). They consist of: hardware integers, hardware floating point types, hardware addresses types, Boolean types, character types, enumerated types, arrays, records;
- *Abstract data types*: are the types which have a hidden local state and an interface of named operations. They are usually specified by the programmer.

5.B Strongly typed languages are those in which the compiler can check that all expressions are type consistent (possibly without it knowing the type of everything). If a language is strongly typed its compiler can guarantee that the programs that it accept will execute without type errors.

5.C The binding of the name of an array to its required amount of storage is called name-size binding. There are two main possibilities:

- *Static array*: In a static array bounds are fixed when the array variable is declared, and it cannot change at run time. This method is simple to compile and is fast at run time, but it is very inflexible. It is used in Pascal, C and Java.
- *Dynamic array*: The size of a dynamic array is not determined until the time when the declaration is obeyed at block entry. This method is slightly less efficient (only at the time of memory allocation) than static arrays but much more convenient, as the bounds can be calculated or taken from data values read by the program. It is used in Java and C.

5.D A binary tree will have at each node an informal field, of type `int` say and two fields or instance variables, each pointing to a subtree or to nothing in which case they would be null. The procedure below visits the tree in left to right, depth-first pre-order:

```
Procedure tree_walk(tree: T)
    begin
        if T is not null
        then
            visit T.rood_node;
            for i to t.no_of_child_nodes
            loop
                tree_walk(T.child_node(i));
            end loop;
        end if;
    end tree_walk;
```