Higher Level Course Examination August 1999

PROGRAMMING AND PROGRAMMING LANGUAGES (Oct 98)

13 August 1999  Time Allowed: 3 hours  18:30 - 21:30

Examination Number

Student Number

There are TWO parts to this paper and you should attempt BOTH PARTS. You should answer ALL questions in Part I and THREE MORE questions in Part II, in which Question 13 is compulsory.

Part I carries 55% of the total examination marks and Part II carries 45%. Marks for each part of each question are shown in brackets, such as [4].

Instructions
1. This examination paper should be answered in ENGLISH.
2. Write your answers to Part I questions in the spaces provided inside this examination paper and answers to Part II questions in the answer book provided. Begin each answer on a new page.
3. Write the question number at the top of each page you have worked on.
4. Do all your rough work in the answer book but remember to cross it out afterwards.
5. You may ask the invigilator for supplementary answer books if you need them.
6. You should write clearly. Marks may be deducted where the writing is very difficult to read.
7. At the end of the examination, check that you have written your examination number and student number on the cover page of this examination paper and on each answer book used. Also check that you have written your course code and completed the examination number and student number column on the answer book flap. Failure to do so will mean that your work cannot be identified.
8. Attach together your examination paper, answer book and any supplementary answer books by the treasury tag provided. The examination paper MUST NOT be removed from the examination room when you leave.
9. Do NOT open this examination paper until you are told to do so, otherwise you may be disqualified.

Admissible materials in this examination:
1. English language and/or English-Chinese language dictionary without additional handwritten or printed notes is allowed. Electronic dictionary is NOT allowed. Violation of the above may lead to disqualification from the examination.
PART I (55 marks)
1 You should attempt ALL of the questions in this part.
2 Write your answers in the spaces provided inside this examination paper.
3 We suggest you spend about 1 hour and 15 minutes completing this part.

Question 1

TWOLESS is an operation that takes two integers as source data and returns the integer which is two less than the smaller of the two. If the source integers have the same value, two less than that value is returned. For example, TWOLESS(7, 8) is 5, and TWOLESS(9, 9) is 7.

Complete the formal specification of the semantics using the constructive approach.

\[
\text{pre-TWOLESS}(x, y) \quad ::= \\
\text{post-TWOLESS}(x, y, r) \quad ::= \\
\]

[4]

Question 2

The following diagram shows a representation of a queue using a 'circular' array.

![Queue Diagram]

Complete the diagram next page to show the representation following the sequence of queue operations: add Choi, add Chui, delete.
Question 3

A BSTree operation COUNT is defined by the axioms:

\[
\text{COUNT(CREATETREE)} = 0 \hspace{1cm} (\text{Axiom 1})
\]
\[
\text{COUNT(MAKETREE}(l, i, r)) = 1 + \text{COUNT}(l) + \text{COUNT}(r) \hspace{1cm} (\text{Axiom 2})
\]

Where \( l \) and \( r \) are BSTrees and \( i \) is an item.

Fill in the boxes in the following explanation of how COUNT can be applied to the given BSTree \( t \):

24

17

The given BSTree \( t \), may be written as: MAKETREE(\( l \), \( i \), \( r \)), where:

\[
l = \hspace{1cm}
\]
\[
r = \hspace{1cm}
\]
\[
i = \hspace{1cm}
\]

Therefore, by Axiom 2, \( \text{COUNT}(t) = 1 + \text{COUNT}(l) + \text{COUNT}(r) \),
And by Axiom 1,

\[ \text{COUNT}(i) = 1 + \quad + \quad \]  

\[ [1] \]

Now, \( l = \text{MAKETREE}(l', i', r') \) where \( l' = r' = \text{CREATETREE} \). By Axiom 2,

\[ \text{COUNT}(l) = 1 + \quad + \quad \]  

\[ [1] \]

And so, by Axiom 1,

\[ \text{COUNT}(i) = 1 + \quad + \quad \]  

\[ [1] \]

Therefore,

\[ \text{COUNT}(i) = \quad \]  

\[ [1] \]

**Question 4**

A Pascal Unit which provides an implementation of the abstract data type *stack* has the form:

```pascal
unit stack;
implementation
  type item_type = integer;
  stack = record
    {representation details}
    end;
  var s: stack;
  {stack procedures and functions}
end.
```

Given that there is no interface part, what are the two main disadvantages of this unit?

(a) \[ \quad \]  

\[ \quad \]

(b) \[ \quad \]  

\[ \quad \]  

\[ [4] \]
Questions 5 and 6 both use the two classes, **POINT,** and **POLYGON** the interface features of which are as follows:

```plaintext
class POINT;
  feature(interface)
    x: REAL;  {x value}
    y: REAL;  {y value}
  translate(x_dir, y_dir; REAL);
    {moves current object by x_dir and y_dir}
  set_coords(x_val, y_val: REAL);
    {updates x and y}
end. {class POINT}

class POLYGON;
  feature(hidden)
    vertices: LIST[POINT];
  feature(interface)
    createpolygon(v: LIST[POINT]);
      {creates polygon with vertice v}
    area: REAL;
      {the area enclosed by polygon}
end. {class POLYGON}
```

Note that a polygon is a many-sided figure like a triangle or rectangle. In this case the vertices of a polygon are defined by points (see the figure below) stored in a list.

```
(3, 4)  
  |
  |
(4, 3)  |
  |
(1, 2)  |
  |
(4, 2)  |
```

Question 5

Complete the following root class which creates a three-sided polygon with vertices (0.0, 0.0), (2.0, 2.0) and (-3.0, 3.0) and prints out its area.

class THREE_SIDED_POLY
feature(hidden)
  poly: POLYGON;
  alist: LIST[POINT];
  p1, p2, p3: POINT;
feature(interface)
  use_poly;
  {procedure providing top-level thread of control}
begin

alist.createlist;
alist.addtoend(p1);
alist.addtoend(p2);
alist.addtoend(p3);

end; {use_poly}
end. {class THREE_SIDED_POLY}
Question 6

A class RECTANGLE (a four sided figure) is defined as inheriting from POLYGON. This question is based on the following root class, not all statements of which have been shown:

class POLYQUEUE

feature (hidden)
    queue: QUEUE[POLYGON];
    poly: POLYGON;
    rect: RECTANGLE;

feature(interface)
    createobjects; {applies creation procedures to hidden attributes}

    begin
        end; {createobjects}
    execute; {provide top-level thread of control}

    begin
        createobjects
        queue.addtoqueue(poly); (1)
        poly := rect; (2)
    .

    end; {execute}

end. {class POLYQUEUE}

(i) The two variables in statement (2) are of different types but this will not cause a compilation error. Explain briefly why not.

(ii) Suppose that the routine addtoqueue(poly), called in statement (1), contains the statement alist.addtoend(i) and that an external error occurs during the execution of the latter statement.

Complete the table below which shows the history of the failure. (For the purpose of this question we are not interested in which statement in addtoend suffers the external error.)

<table>
<thead>
<tr>
<th>class</th>
<th>Routine</th>
<th>Failed statement or assertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST[I]</td>
<td>addtoend</td>
<td>unknown</td>
</tr>
</tbody>
</table>
Question 7

From the list of words and phrases given below complete the given sentences by entering the appropriate number in the spaces provided.

In Ada, the select statement works in the following manner:

(i) The ______ in the statement are evaluated in some ________ order, to determine which are open. [1]

(ii) The ________ whose ________ are open are now considered: if a ________ is possible in one of these, the ________ is made by executing its accept statement. This situation occurs if a client task has already issued the appropriate ________ call. [1]

(iii) If more than one of the ________ is open, one of the associated ________ is chosen arbitrarily. [1]

(iv) If no ________ is possible, the server process is ________ until a ________ is possible, as in Step (ii). [1]

(v) If no ________ are open, the program is in error.

Words and phrases

1 accept statements
2 blocked
3 caretakeprocess
4 synchronization
5 critical section
6 deadlock
7 determinism
8 entry
9 guards
10 guarded commands
11 message passing
12 monitor
13 non-determinism
14 mutual exclusion
15 rendezvous
16 select statement
17 task
18 undefined
19 well defined
20 when-part
**Question 8**

Given the Prolog program:

\[
\begin{align*}
\text{sum_list([], 0),} \\
\text{sum_list([A | B], T) :- sum_list(B, T1), T is A + T1.}
\end{align*}
\]

What response do you expect the interpreter to give to each of the following queries?

(a) \(?- \text{sum_list([2], Z).}\)

(b) \(?- \text{sum_list([a, b, c], Z).}\)

(c) \(?- \text{sum_list([3, 4, 2], Z).}\)

(d) \(?- \text{sum_list([], Z).}\)

**Question 9**

(i) Given the following database (in which \(\ne\) means not equals)

\[
\begin{align*}
\text{works(chan, morning).} \\
\text{works(lee, evening).} \\
\text{works(cheung, afternoon).} \\
\text{works(wong, evening).} \\
\text{works(ho, afternoon).}
\end{align*}
\]

\[
\begin{align*}
\text{knows(X, Y) :- works(X, Shift), works(Y, Shift), X \ne Y.}
\end{align*}
\]

What response would Prolog interpreter give to the following query?

\(?- \text{knows(X, Y).}\)

(ii) What response would our Prolog interpreter give if asked to resatisfy the query in part (i)?
Question 10

(i) Given the following grammar:

\[
S \rightarrow X \\
X \rightarrow Y \mid X \ast Y \\
Y \rightarrow Z \mid Z + Y \\
Z \rightarrow a \mid b \mid c \mid d \mid e \mid f
\]

Tick the appropriate box to indicate which of the following trees represents the parse tree derived from the expression: \( e + b \ast c + f \)

```
  *       +       +       +
 / \     / \     / \     / \ 
 +   +   *   f   *   f   e   *
 / \ / \   / \   / \   / \ 
 e   b   c   f   +   c   e   +   b   +
 / \     / \     / \     / \ 
 e   b     b   c     c   f
```

(ii) Assuming that \( e = 3, b = 2, c = 7, \) and \( f = 5 \), mark in circles at appropriate locations on the tree of your choice in (i), the synthesized attributes which would result when the expression in (i) is evaluated.

[2]

[3]
Question 11

This question concerns the following program written in SL:

< n
< m
p : n
?q n m < ; p : m
?q p 1 < ;.
s : 0
*k p ; s : s k +
>s
.

(i) Give an informal explanation of the meaning of the program.

................................................................................................................................................
................................................................................................................................................
................................................................................................................................................
................................................................................................................................................
................................................................................................................................................
................................................................................................................................................
................................................................................................................................................

[4]

(ii) Fill in the missing words in the following explanation of the meaning of an SL program:

    We translate from SL to ______________________ and then execute the
    __________________ code on _________________________________.

[2]
Question 12

Write down a syntax checking procedure that will check the syntax of an `<if statement>` given by the syntax diagram:

```
<if statement>
  if <expression>
  then <statement>
  else <statement>
```

You may assume the existence of routines for checking the syntax of each non-terminal and that the procedures:

```
insymbol(var sy: symbol)
```

and

```
accept(var sy: symbol; expectedsymbol: symbol)
```

are also available.

Do not include general error recovery in your solution but indicate where `stest` and/or `ftest` might be included. Give reasons for including or excluding these tests.

[END OF PART I]
PART II  (45 marks)

1  You should attempt the compulsory Question 13 and not more than TWO OTHER questions in this part.

2  Each question is worth 15 marks.

3  Write your answers in the answer book provided. Answers to Part II written on the examination paper would not be marked.

4  You are advised to spend about 30 minutes on each of your three Part II questions, leaving about 15 minutes for checking.

Question 13 (15 marks) (Compulsory)

Given the following specification of the ADT queue:

NAME
  QUEUE(I)

SETS
  Q  the set of all queues
  I  the set of all items
  B  the set of Boolean values consisting of true and false
  E  the set of errors consisting of the single member error-empty queue

SYNTAX OF OPERATIONS
  CREATEQUEUE → Q
  FRONT:  Q → I UNION E
  DELETEFROMQUEUE: Q → Q UNION E
  ADDTOQUEUE: I, Q → Q
  IEMPTYQUEUE: Q → B

SEMANTICS OF OPERATIONS
  i is an instance of I; b is an instance of B, q is an instance of Q and the underlying model is a list.
  pre-CREATEQUEUE( ) ::= true
  post-CREATEQUEUE(q) ::= isemptylist(q)
  pre-FRONT(q) ::= true
  post-FRONT(q; r) ::= IF not isemptylist(q)
    THEN
      r = first(q)
    ELSE
      r = error-empty queue

\texttt{pre-DELETEFROMQUEUE}(q) ::= \texttt{true} \\
\texttt{post-DELETEFROMQUEUE}(q; r) ::= \texttt{IF not isempty}(q) \\
THEN \\
r = \texttt{trailer}(q) \\
ELSE \\
r = \texttt{error-empty queue} \\
\texttt{pre-ADDTOQUEUE}(i, q) ::= \texttt{true} \\
\texttt{post-ADDTOQUEUE}(i, q; r) ::= r = (q \texttt{concatenate make}(i)) \\
\texttt{pre-ISEMPTYQUEUE}(q) ::= \texttt{true} \\
\texttt{post-ISEMPTYQUEUE}(q; b) ::= b = \texttt{isempty}(q)

(i) Write down the changes that would have to be made to this specification to avoid dealing with error situations and to show more clearly the generic qualities of those ADTs used in the specification. \hfill [4]

(ii) Draw a diagram of the relationships between the client and supplier ADTs involved in your revised specification of the \textit{queue} ADT. Indicate, on your diagram, which ADT(s) is(are) client(s) and which is(are) supplier(s). \hfill [2]

(iii) Write down the changes you would make to your revised specification of a queue in part (i) to modify it so that a queue may have no more than a specified maximum number of items. \hfill [2]

(iv) Draw up a table describing the obligations and benefits of the client-supplier contract in relation to the queue operation \textit{FRONT}. \hfill [2]

(v) \textbf{Briefly} describe what is meant by the client-supplier relationship and the subtype relationship. Your answer should clearly indicate the circumstances in which each of the relationships is appropriate. \hfill [5]
Question 14 (15 marks)

The question involves the following Prolog database in which the clauses have been numbered for ease of reference.

(1) `arel([], []).`

(2) `arel([H \ [ ]], []).`

(3) `arel([A, B], A).`

(4) `arel([H | T], L) :- arel(T, L).`

and the query

`?- arel([d, e, f, g], Y).`

(i) (a) With which of the clauses in the database will the query initially match? Write down the resulting instantiations.  

[4]

(b) A formal parameter in a Prolog procedure can behave either as a constant or as an input/output parameter. Describe, for the first unification of the above query, how the relevant formal parameters behave.  

[2]

(c) Write down, in your own words, an informal description of `arel`. What answer will the Prolog interpreter finally give to the query?  

[4]

(ii) Write a Prolog relation called `sublist` which has two list arguments. The first argument is a `sublist` of the second provided its elements occur in the second argument in the same order but not necessarily consecutively. For example,

`sublist([b, e], [a, b, d, e, g]).`

is an instance of the relation.  

[5]
Question 15 (15 marks)

In the course, the syntax of the SL language is given by:

\[
<\text{SL-program}> ::= \{ <\text{statement}> \ <\text{newline}> \}\n\]

\[
<\text{statement}> ::= <\text{read}> \ | \ <\text{write}> \ | \ <\text{halt}> \ | \ <\text{assignment}> \ | \ <\text{conditional}> \ | \ <\text{loop}>
\]

\[
<\text{read}> ::= \langle <\text{variable}> \rangle
\]

\[
<\text{write}> ::= \rangle <\text{expression}>
\]

\[
<\text{halt}> ::= .
\]

\[
<\text{assignment}>::= <\text{variable}> :<\text{expression}>
\]

\[
<\text{conditional}>::= ? <\text{expression}> ; <\text{statement}>
\]

\[
<\text{loop}>::= * <\text{variable}> <\text{value}> ; <\text{statement}>
\]

\[
<\text{expression}>::=<\text{value}> | <\text{expression}> <\text{expression}> <\text{operator}>
\]

\[
<\text{value}>::=<\text{variable}|<\text{constant}>
\]

\[
<\text{variable}>::= a \ | \ b \ | \ c \ | \ d \ | \ . . . \ | \ y \ | \ z
\]

\[
<\text{constant}>::= 0 \ | \ 1 \ | \ 2 \ | \ 3 \ | \ . . . \ | \ 9
\]

\[
<\text{operator}>::= + \ | \ - \ | \ * \ | \ / \ | \ < \ | \ > \ | \ =
\]

(i) Is the SL grammar an example of a context-free or context-sensitive grammar? Give a reason for your answer. [2]

(ii) Draw the phrase structure (parse tree) for the following SL program:

\[
< \ n
\]

\[
n::=n2*
\]

\[
>\ n
\]

(iii) Extend the SL grammar to a grammar ESL so that you can declare subroutines (see below) at the beginning of a program (before any other statements) and then use them anywhere within the program that a statement may be used. Your answer should use EBNF syntax and need only show the rules that have changed from the SL grammar and any new rules.

Subroutines are named by a single uppercase letter (A, B, C, ... Z) and they do not take any parameters, nor do they have scope for local variables (all variables are global). Subroutine declarations begin with the carat character (^), followed by the name of the subroutine, followed by some number of statements, followed by an exclamation mark (!).
For example, here is a complete ESL program having a single subroutine named F, and a main program that reads in a value into the variable n, calls the subroutine and finally writes out the value of the variable r:

```
^ F
 r: 0
 * k n; r : r k +
 !
 < n
 F
 > r
 .
```

F is a subroutine containing two statements but their meaning is not important in answering the question.

[8]
Question 16 (15 marks)

(i) Copy down and complete the symbol table for the following program fragment.

```
program prog 1;
const
  max = 10;
type
  name = array[1 .. max] of char;
var
  a: integer;
  b: real;
  n: name;
procedure check(var c: boolean; n: name);
var
  a: real;
  p: name;
begin
```

<table>
<thead>
<tr>
<th>identifier</th>
<th>type</th>
<th>class</th>
<th>link</th>
<th>level</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(ii) Here are two assignment statements:

```
x := a;
y := x > 3;
```

which are constructed according to the following syntax:

```
<variable> := <expression>
```

where

```
<expression>
```

```
<simple expression> → <expression>
```

```
= <> < > >= <=
```

```
<simple expression> → <simple expression>
```

[4]
(a) Outline, briefly, the semantic checks that a compiler takes to ensure that strong typing is adhered to in assignment statements such as those given above.  

(b) At what point is widening used by a compiler?

(iii) The three main components of the Pascal compiler discussed in the course are: the lexical analyser, the syntax checker and the code generator. Describe briefly the function of each component in the compiling process.

If a new control structure were to be added to the language, which component(s) would require modification? Give reasons for each component requiring modification.

[END OF EXAMINATION PAPER]