UNIVERSITY OF MORATUWA

Faculty of Engineering

Department of Computer Science and Engineering

B.Sc. Engineering Honours Degree

Semester 5 Examination (2019 Intake)

CS 3512 Programming Languages

Time allowed: 2 Hours December 2022

Instructions to candidates

1. This paper consists of 4 questions in 10 pages.
2. Answer ALL 4 questions.
3. Start answering each of the 4 main questions on a new page.
4. The maximum attainable mark for each question is given in brackets.
5. This examination accounts for 60% of the module assessment.
6. This is a closed book examination.

   It is an offence to be in possession of unauthorized material during the examination.
7. Only calculators approved and labelled by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper or Appendix. Clearly state such assumptions made on the script.
9. Appendices in page 7-10 contains RPL Grammar rules, CSE machine rules, lambda calculus axioms and Roman Numerals from 1 to 100.
10. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
11. This paper should be answered only in English.

Q1. [25 marks]

(a) For L_1 = {a, b,cc} and L_2 = {ac,ca}, calculate L_1L_2 , $L_1 \cup L_2$, and L_1^3.

[2 Marks]

(b) Consider the following grammar.

Sentence	→ NP VP
NP	→ N
NP	→ Adj NP
N	→ boy
N	→ girl
Adj	→ the
Adj	→ tall
Adj	→ jealous
VP	→ V NP
V	→ hit
V	→ bit

produce the derivation of the sentence "tall the jealous boy hit the tall boy".

[3 Marks]

(c) Construct a context-free grammar that can parse all the lowercase roman numerals from 1 to 99. The terminal symbols for roman numerals are $\{c, l, x, v, i\}$ where $c = 100, l = 50, x = 10, v = 5$ and $i = 1$. For your reference roman numerals from 1 to 99 are given in the appendix E. [7 Marks]

(d) Draw a parse tree for 97: "xcvii", using above grammar. [3 Marks]

(e) Below is the definition of the Integer constants in C++.

"An integer constant consisting of a sequence of digits is taken to be decimal (base ten) unless it begins with 0 (digit zero). A sequence of digits starting with 0 is taken to be an octal integer (base eight). The digits 8 and 9 are not octal digits. A sequence of digits preceded by 0x or 0X is taken to be a hexadecimal integer (base sixteen). The hexadecimal digits include a or A through f or F. The type of an integer depends on its suffix. [An integer may have no suffix]. If the integer is suffixed by u or U, its type is unsigned int . If it is suffixed by l or L, its type is long int . [Either or both suffixes may appear at most once each in either order.]"

Give a regular expression for C++ integer constants. Use only the operations +, *, and concatenation in your expression.

[5 Marks]

(f) Briefly explain the following phases in a compiler.

I. Scanner
II. Screener
III. Parser
IV. Constrainer
V. Code Generator [1 x 5 marks]

Q2. [25 marks]

Consider the following RPAL program which takes two tuples as arguments and returns the vector sum of the two tuples. As an example, Vec_sum ((1,-2,3),(4,5,6)) will produce (5,3,9) as the result.

```
let Vec_sum (A,B) =
    Psum(A,B,Order A) where
        rec Psum(A,B,N) =
            N eq 0 ->  nil
            |  (Psum(A,B,N-1) aug  A N + B N)
in Print (Vec_sum ( (1,-2,3),  (4,5,6) ))
```

(a) Construct the Abstract Syntax Tree (AST) for the above Program. [5 marks]

(b) Construct the Standardize Tree (ST) for the above Program using the AST constructed in above (a). Transformational Rules are provided in the appendix. [5 marks]

(c) List down the Control Structures of the above program. [5 marks]

(d) Show the Control Stack Environment (CSE) machine evaluation for the above program. [10 marks]

Q3. [25 marks]

(a) The Tower of Hanoi is a game consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The game begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the game is to move the entire stack to the last rod, obeying the following rules:

i. Only one disk may be moved at a time.

ii. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.

iii. No disk may be placed on top of a disk that is smaller than it.

Write a RPAL program that defines and uses a recursive function THanoi which takes four arguments, i.e. ‘A’, ‘B’, ‘C’ as rod names and ‘n’ which is the number of rods, and print the sequence of movements to complete the game. For example,

(THanoi ‘A’ ‘B’ ‘C’ 3) should return the below output.

Move A to B

Move A to C

Move B to C

Move A to B

Move C to A

Move C to B

Move A to B

[6 marks]

(b) Construct the Abstract Syntax Tree (AST) for the program you wrote for part (a).

[6 marks]

(c) Show the output of the following program segment (written in C-like syntax) for each of the following parameter-passing mechanisms:

I. pass by value-result, [2 marks]
II. pass by value, and [2 marks]
III. pass by reference. [2 marks]

```
int a;
int b;
void f(int c, int d)  {
     print (c);
     print (d);
     c = 4;
     print(a+b+c+d);
}
main ()  {
     a = 6, b=3;
     f(a,b) ;
     print (a);
     print (b);
}
```

(d) For the program shown below, draw a picture of the run-time environment, at the point marked "HERE", i.e., **at the point when the value of ‘b’ has been placed on the stack**. Show all temporaries, local variables, parameters, locations reserved for return values, and return addresses, as well as the base pointer, frame pointer(s), and the stack pointer. [7 marks]

```
function gcd (int a, int b) {
        int q;
        if a = 0 then return b // HERE
        else {
                q= gcd (a, b);
                return q;
        }
}
main( )
begin
        write(gcd ( 42, 28))
end;
```

Q4. [25 marks]

Consider the following regular expression: (ab*c + de*)*

(a) Transform the above regular expression to an NFA, from there to a right-linear regular grammar. [2 Marks]

(b) Transform the NFA from Part (a), to a DFA. [2 Marks]

(c) Minimize the DFA obtained in Part (b). [2 Marks]

(d) Write (in pseudo-code) a lexical analyzer for the language given by the above regular expression.
Write two versions of the lexical analyzer:
   I. Table-driven. [2 Marks]
   II. Hard-coded. [3 Marks]

(e) Underline the free variable occurrences in the following Applicative Expressions (AE's): You should copy the AE's to the answer script and underline free variable occurrences.
   I. (fn x. fn y. x y) x
   II. (fn x. ((fn y. fn x. x+x) (x+1) (y+1)) +1) 1
[1 x 2 marks]

(f) Reduce the following Applicative Expressions (AE's)
   I. (fn x. fn y. x y) (fn z. z+2) 1
   II. (fn x. fn y. x y) (fn y. y+2) 1
   III. (fn x. fn y. x y) (fn x. fn y. x y) (fn z. z+1) 1
   IV. (fn x. x+2) ((fn y. y+1) 3)
   V. (fn x. x+2) (fn y. y+1) 3 [1 x 5 marks]

(g) Consider the following clauses where plays(A,B) denotes person A plays Instrument B:

```
plays(flagTwirlers, flags).
plays(plankton, keyboard).
plays(squidward, clarinet).
plays(patrick, mayo).
plays(patrick, drums).
plays(spongebob, drums).
plays(spongebob, spatula).

twoPlayers(X) :- plays(Y, X), plays(Z, X), not (Y = Z).
talented(X) :- plays(X, Y), plays(X, Z), not (Y = Z).

band1([ ], [ ]).
band1([H | T], [X | Y]) :- talented(H), plays(H, X), band1(T, Y).
band2([ ], [ ]).
band2([H | T], [X | Y]) :-
plays(H, X), band2(T, Y), not (member(H, T)).
```

Note: Prolog lists have the following syntax. Starts and ends with square brackets, and each of the items they contain is separated by a comma. Ex: [apple, orange, grapes]. Also, Prolog provides a way to split the first part of the list (called the head) away from the rest of the list (known as the tail) by placing the symbol " | " (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list. Ex: [first,second,third] = [A | B] where A = first and B=[second,third]

Write the prolog answers for each of the following query.

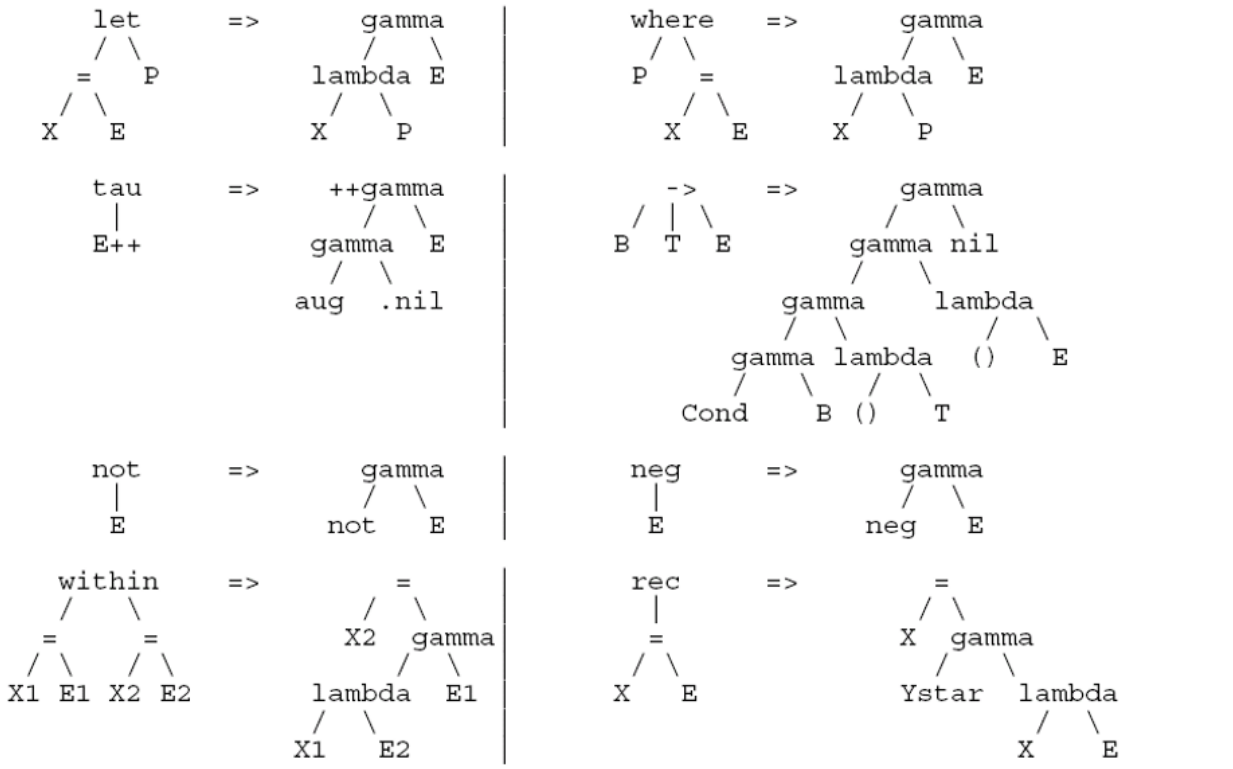
i. ?- plays (patrick,X).
ii. ?- twoPlayers (X).
iii. ?- talented (squidward).
iv. ?- band1 ([patrick, flagTwirlers], X).
v. ?- band1 (X, [drums, mayo]).
vi. ?- band2 (X, [drums, drums]).
vii. ?- band2 ([spongebob, plankton, flagTwirlers], Y).

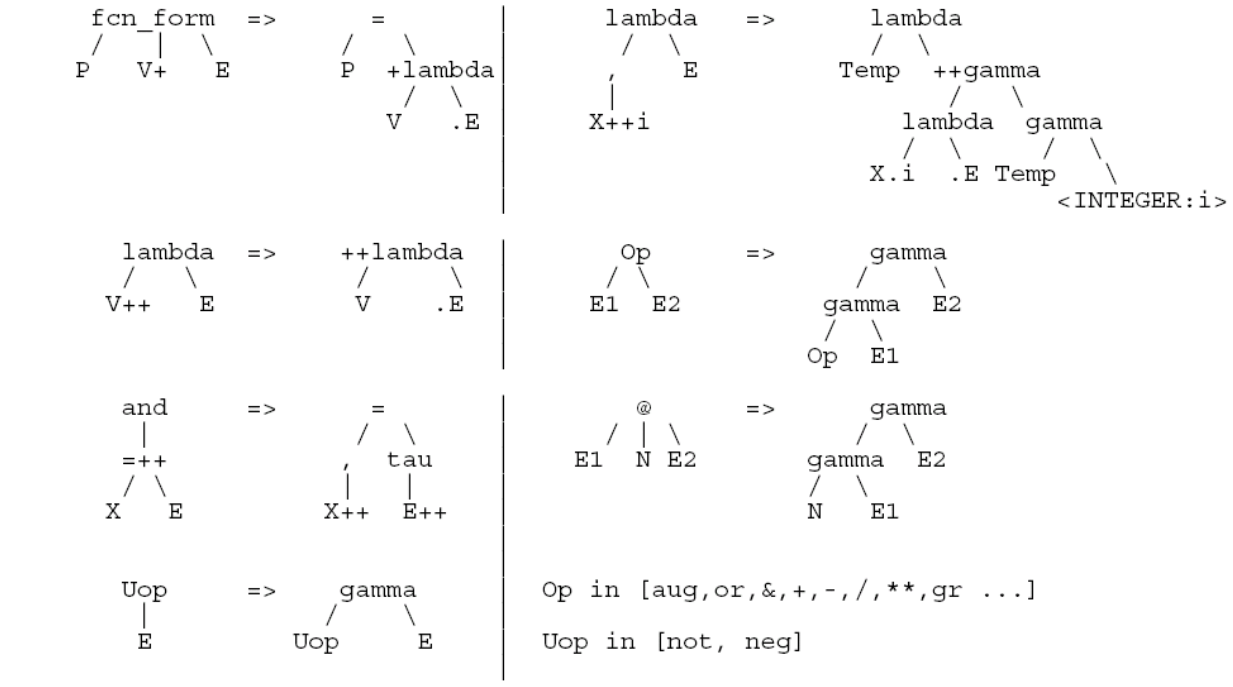
[1 x 7 marks]

---------- End of the Examination Paper ----------

Appendix

A. RPAL Subtree Transformational Gramma





B. CSE Machine Rules

	CONTROL	STACK	ENV
Initial State	$e_0\ \delta_0$	e_0	e_0 = PE
CSE Rule 1 (stack a name)	.... Name	.... Ob	Ob=Lookup(Name,e_c) e_c:current environment
CSE Rule 2 (stack λ)	.... λ_k^x	.... $^c\lambda_k^x$	e_c:current environment
CSE Rule 3 (apply rator)	.... γ	Rator Rand .... Result	Result=Apply[Rator,Rand]
CSE Rule 4 (apply λ)	.... γ .... $e_n\ \delta_k$	$^c\lambda_k^x$ Rand .... e_n	$e_n = [Rand/x]e_c$
CSE Rule 5 (exit env.)	.... e_n	value e_n .... value	
CSE Rule 6 (binop)	.... binop	Rand Rand .... Result	Result=Apply[binop,Rand,Rand]
CSE Rule 7 (unop)	.... unop	Rand .... Result	Result=Apply[unop,Rand]
CSE Rule 8 (Conditional)	.... $\delta_{then}\ \delta_{else}\ \beta$	true	
CSE Rule 9 (tuple formation)	.... τ_n	V_1 ... V_n .... $(V_1,...,V_n)$	
CSE Rule 10 (tuple selection)	.... γ	$(V_1,...,V_n)$ I .... V_I	
CSE Rule 11 (n-ary function)	.... γ .... $e_m\ \delta_k$	$^c\lambda_k^{V_1,...,V_n}$ Rand .... e_m	e_m=[Rand 1/V_1]... [Rand n/V_n]e_c
CSE Rule 12 (applying Y)	.... γ	Y $^c\lambda_i^v$ .... $^c\eta_i^v$	
CSE Rule 13 (applying f.p.)	.... γ .... $\gamma\ \gamma$	$^c\eta_i^v$ R .... $^c\lambda_i^v$ $^c\eta_i^v$ R	

C. RPAL's Phrase Structure Grammar

```
# Expressions ##############################################

E    -> 'let' D 'in' E                          => 'let'
     -> 'fn'  Vb+ '.' E                         => 'lambda'
     ->  Ew;
Ew   -> T  'where' Dr                           => 'where'
     -> T;

# Tuple Expressions ########################################

T    -> Ta ( ',' Ta )+                          => 'tau'
     -> Ta ;
Ta   -> Ta 'aug' Tc                             => 'aug'
     -> Tc ;
Tc   -> B '->' Tc '|' Tc                        => '->'
     -> B ;

# Boolean Expressions ######################################

B    -> B 'or' Bt                               => 'or'
     -> Bt ;
Bt   -> Bt '&' Bs                               => '&'
     -> Bs ;
Bs   -> 'not' Bp                                => 'not'
     -> Bp ;
Bp   -> A ('gr' | '>' ) A                       => 'gr'
     -> A ('ge' | '>=') A                       => 'ge'
     -> A ('ls' | '<' ) A                       => 'ls'
     -> A ('le' | '<=') A                       => 'le'
     -> A 'eq' A                                => 'eq'
     -> A 'ne' A                                => 'ne'
     -> A ;

# Arithmetic Expressions ###################################

A    -> A '+' At                                => '+'
     -> A '-' At                                => '-'
     ->   '+' At
     ->   '-' At                                => 'neg'
     -> At ;
At   -> At '*' Af                               => '*'
     -> At '/' Af                               => '/'
     -> Af ;
Af   -> Ap '**' Af                              => '**'
     -> Ap ;
Ap   -> Ap '@' '<IDENTIFIER>' R                 => '@'
     -> R ;

# Rators And Rands #########################################

R    -> R Rn                                    => 'gamma'
     -> Rn ;
Rn   -> '<IDENTIFIER>'
     -> '<INTEGER>'
     -> '<STRING>'
     -> 'true'                                  => 'true'
     -> 'false'                                 => 'false'
     -> 'nil'                                   => 'nil'
     -> '(' E ')'
     -> 'dummy'                                 => 'dummy' ;

# Definitions ##############################################

D    -> Da 'within' D                           => 'within'
     -> Da ;
Da   -> Dr ( 'and' Dr )+                        => 'and'
     -> Dr ;
Dr   -> 'rec' Db                                => 'rec'
     -> Db ;
Db   -> Vl '=' E                                => '='
     -> '<IDENTIFIER>' Vb+ '=' E                => 'fcn_form'
     -> '(' D ')' ;

# Variables ################################################

Vb   -> '<IDENTIFIER>'
     -> '(' Vl ')'
     -> '(' ')'                                 => '()';
Vl   -> '<IDENTIFIER>' list ','                 => ','?;
```

D. Lambda Calculus Axioms

Axiom Delta:

Let M and N be AE's that do not contain λ-expressions.
Then $M =>_{\delta} N$ if Val(M) = Val(N).

Axiom Alpha:

Let x and y be names, and M be an AE with no free occurrences of y. Then, in any context,
$\lambda\ x.M =>_{\alpha} \lambda\ y.subst[y,x,M]$

Axiom Beta:

Let x be a name, and M and N be AE's. Then, in any context,
$(\lambda x.M)\ N =>_{\beta} subst[N,x,M]$.

Axiom ρ (Fixed point identity):

$Y\ F =>_{\rho} F\ (Y\ F)$.

E. Roman numerals from 1 to 100.

1	I	**21**	XXI	**41**	XLI	**61**	LXI	**81**	LXXXI
2	II	**22**	XXII	**42**	XLII	**62**	LXII	**82**	LXXXII
3	III	**23**	XXIII	**43**	XLIII	**63**	LXIII	**83**	LXXXIII
4	IV	**24**	XXIV	**44**	XLIV	**64**	LXIV	**84**	LXXXIV
5	V	**25**	XXV	**45**	XLV	**65**	LXV	**85**	LXXXV
6	VI	**26**	XXVI	**46**	XLVI	**66**	LXVI	**86**	LXXXVI
7	VII	**27**	XXVII	**47**	XLVII	**67**	LXVII	**87**	LXXXVII
8	VIII	**28**	XXVIII	**48**	XLVIII	**68**	LXVIII	**88**	LXXXVIII
9	IX	**29**	XXIX	**49**	XLIX	**69**	LXIX	**89**	LXXXIX
10	X	**30**	XXX	**50**	L	**70**	LXX	**90**	XC
11	XI	**31**	XXXI	**51**	LI	**71**	LXXI	**91**	XCI
12	XII	**32**	XXXII	**52**	LII	**72**	LXXII	**92**	XCII
13	XIII	**33**	XXXIII	**53**	LIII	**73**	LXXIII	**93**	XCIII
14	XIV	**34**	XXXIV	**54**	LIV	**74**	LXXIV	**94**	XCIV
15	XV	**35**	XXXV	**55**	LV	**75**	LXXV	**95**	XCV
16	XVI	**36**	XXXVI	**56**	LVI	**76**	LXXVI	**96**	XCVI
17	XVII	**37**	XXXVII	**57**	LVII	**77**	LXXVII	**97**	XCVII
18	XVIII	**38**	XXXVIII	**58**	LVIII	**78**	LXXVIII	**98**	XCVIII
19	XIX	**39**	XXXIX	**59**	LIX	**79**	LXXIX	**99**	XCIX
20	XX	**40**	XL	**60**	LX	**80**	LXXX	**100**	C