HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP3031 (Principles of Programming Languages)

Fall 2013

FINAL EXAMINATION

Dec 9, 2013 (Mon) 8:30AM-11:30AM, Room LG4

Name	**ITSC Account**
Student ID	**Lecture Section (tick one)** **[ ] L1 (T/R) [ ] L2 (M/W)**

1. *About the exam:*
    a. *This is a closed-book, closed-note examination.*
    b. *You CANNOT use any electronic devices including calculator during the examination. Please TURN OFF all of your electronic devices (e.g., mobile phone) and put them into your bag.*
    c. *You CANNOT leave during the last 15 minutes of the examination.*
2. *About this paper:*
    a. *There are a total of 14 pages, including this page.*
    b. *There are a total of eight questions accounting for 100 points in total.*
3. *About your answers:*
    a. *Write your answers in the designated space following each question.*
    b. *Make sure your final answers are clearly recognizable.*
    c. *Rough work can be done in the provided "additional blank paper for draft work". Do not, however, write answers there as they will NOT be graded*
    d. *Attempt all questions.*

Problem	1	2	3	4	5	6	7	8	Total
Marks									

Problem 1. SML Programming (10 points)

a) Given the following datatype `tree` representing a binary tree of integer labels. Write a function *height* to return the height of the tree.

```
datatype tree = nil | node of int*tree*tree;

val height = fn : tree -> int
```

Examples:

```
- height nil;
val it = 0 : int
- height (node(1,nil,nil));
val it = 1 : int
- height (node(1, node(2,nil,node(3,nil,nil)),nil));
val it = 3 : int
```

Answer:

```
fun height nil = 0 | height (node(I,L,R)) =
let val height_L = 1+ height L
val height_R = 1+ height R
      in if height_L > height_R then height_L else height_R
      end;
```

Marking:

1) Empty tree case: 2 marks
2) Non-empty tree case: 3 marks
3) Syntax Error: deduct 1 mark for each kind of syntax errors

b) Given a list L, write a function named *rotate* to circularly shift the list N places to the left. When N is positive, the direction of shifting is to the left; when N is negative, the direction is to the right.

```
val rotate = fn : 'a list * int -> 'a list
```

Examples:

```
- rotate([1,2,3,4,5,6,7,8],3);
val it = [4,5,6,7,8,1,2,3] : int list
- rotate([1,2,3,4,5,6,7,8],~2);
val it = [7,8,1,2,3,4,5,6] : int list
- rotate([1,2,3,4,5,6,7,8],9);
val it = [2,3,4,5,6,7,8,1] : int list
- rotate([1,2,3,4,5,6,7,8],~9);
val it = [8,1,2,3,4,5,6,7] : int list
```

Answer:

```
fun rotate_h([], N) = []| rotate_h(L, N) = if N=0 then L else
rotate_h((tl(L)@[hd(L)]), N-1);
fun rotate([],N) = [] | rotate(L, N) = rotate_h(L, N mod length(L));
```

Marking:
1) Empty list case: 1 mark
2) Non-empty list case: when N>=0, 2 marks; when N<0, 2 marks
3) Syntax Error: deduct 1 mark for each kind of syntax errors

Problem 2. Prolog Programming (10 points)

a) Write a predicate `drop(L,N,X)` such that for a given list L, X is the list after dropping **EVERY** N'th element from the list L. Assume N is a given natural number.

Examples:

```
?- drop([],5,X).
X = [].
?- drop([1,2],3,X).
X = [1, 2].
?- drop([a,b,c,d,e,f,g,h,i,k],3,X).
X = [a,b,d,e,g,h,k]
```

Answer:

```
drop(L1,N,L2) :- drops(L1,N,L2,N).
drops([], _, [], _) :- !.
drops([_|Xs], N, Ys, 1) :- !, drops(Xs,N,Ys,N).
drops([X|Xs], N, [X|Ys], K) :- K1 is K - 1, drops(Xs,N,Ys,K1).
```

Marking:
1 point for base case (empty list)
2 points for dropping element and resetting counter
2 points for changing counter
-1 point for syntax or any other error

b) Write a predicate `rotate(L,N,X)` such that for a given list L, X is the list after circularly shifting the list L N places to the left. When N is positive, the direction of shifting is to the left; when N is negative, the direction is to the right.

Examples:

```
?- rotate([a,b,c,d,e,f,g,h],3,X).
X = [d,e,f,g,h,a,b,c]

?- rotate([a,b,c,d,e,f,g,h],-2,X).
X = [g,h,a,b,c,d,e,f]
```

Answer:

```
split(L, 0, [], L) :- !.
split([X|Xs],N,[X|Ys],Zs) :- N1 is N - 1, split(Xs,N1,Ys,Zs).

rotate([], _, []) :- !.
rotate(L1,N,L2)   :-   length(L1,NL1),   N1   is   N   mod   NL1,
split(L1,N1,S1,S2), append(S2,S1,L2).
```

Marking:
1 point for base case (empty list)
2 points for split
2 points for mod and append
-1 point for syntax or any other error

OR

```
rotate([],_,[]) :- !.
rotate(L,N,R) :- length(L,NL),N1 is N mod NL, rotateN(L,N1,R).

rotateN(L,0,L) :- !.
rotateN([X|Xs],N,R):-append(Xs,[X],Ys),N1 is N - 1,rotate(Ys,N1,R).
```

Marking:
1 point for base case (empty list)
2 points for mod, rotateN and its base case N=0
2 points for append (no points for [Xs|X] or [Xs|[X]])
-1 point for syntax or any other error

Problem 3. Cuts and Negation in Prolog (10 points)

Given the following Prolog program, answer the queries.

```
    num_parent(X,0) :- adam_eve(X), !.
    num_parent(X,2) :- \+ adam_eve(X).
    adam_eve(adam).
    adam_eve(eve).
```

a)

```
?- num_parent(a,X).
```

X=2

b)

```
?- num_parent(X,0).
```

X=adam

c)

```
?- num_parent(X,2).
```

false

d)

```
?- num_parent(eve,0).
```

true

e)

```
?- num_parent(X,Y).
```

X=adam
Y=0

Marking:
2 marks for each answer (total 10).

Mark deductions:
-1 for partially correct answer
-1 for addition "true", "false" returned
-1 to -4 for other mistakes

Problem 4. Prolog Search Tree (15 points)

Consider the following Prolog program. Recall build-in predicates `var/1` is true for any variable and false otherwise; `nonvar/1` is true for any non-variable and false otherwise.

```
/*R1*/ helper1(true).

/*R2*/ helper2(A):-var(A),helper1(A),A,!.
/*R3*/ helper2(A):-nonvar(A),A.

/*R4*/ pred1(A,B):-helper2(A),helper2(B).

/*R5*/ pred2(A,B):-helper2(A).
/*R6*/ pred2(A,B):-helper2(B).
```

a) Draw the complete Prolog search tree for the query `pred1(A,B)`. At each **tree edge**, whenever applicable, **i)** indicate the rule number Ri, i=1,..,6, of the rule being applied, and **ii)** the unification(s) being made. At each **tree node** indicate the goal to be satisfied. The initial step has been done for you.

```
                    pred1(A,B)
          R4:{_1/A , _2/B} |
```

Answer:

```
          helper2(A) , helper2(B)
            R2:{_3/A }|
           var(A) ,helper1(A),A, helper2(B)
           R1:{A/true} |
                 helper1(true),true, helper2(B)
           R2:{_4/B}    |
           var(B) ,helper1(B),B
           R1:{B/true}  |
                 helper1(true),true
                        |
                     success
```

Marking:

2 marks for correct helper2(A) goal
2 marks for correct helper2(B) goal
2 marks for correct trace up to the "success" leave

mark deductions:
-2 for incorrect tree*
-2 for missing steps.

* (one should be able to trace back directly the answer of the query from the "success" leave through its "direct" ancestors but **not siblings or ancestors' siblings'**, otherwise it is incorrect.)

b) Draw the complete Prolog search tree for the query `pred2(pred1(A,pred2(B,C)),D).`

Answer:

```
                                          pred2(pred1(A,pred2(B,C)),D)
            R5:{_1/pred1(A,pred2(B,C))}                                    R6: {_13/D}
                                              /                       \
                             helper2(pred1(A,pred2(B,C))                  SUBTREE2
  R2:{_2/pred1(A,pred2(B,C))  /               \ R3:{_3/pred1(A,pred2(B,C))}
     Var(pred1…),helper1(pred1…),pred1…    SUBTREE1
                             /
                            fail
```

SUBTREE1:

```
         Nonvar(pred1(A,pred2(B,C)) , pred1(A,pred2(B,C))
                                       |
                            pred1(A,pred2(B,C))
         R4:{_4/A,_5/pred2(B,C)}       |
                         helper2(A),helper2(pred2(B,C))
                             R2:{_6/A}  |
                               var(A),helper1(A),A, helper2(pred2(B,C))
                            R1:{A/true}  |
                                    Helper1(true),true, helper2(pred2(B,C))
      R2:{_7/pred2(B,C)}  /                        \ R3:{_8/pred2(B,C)}
var(pred2(B,C)),helper1(pred2(B,C)),pred2(B,C)     nonvar(pred2(B,C)),pred2(B,C)
                           |                                 |
                          fail                           pred2(B,C)
                                          R5:{_9/B}  /              \ R6:{_11/C}
                                                 helper2(B)        helper2(C)
                                         R2:{_10/B}  |                   | R2:{_12/C}
                                          var(B),helper1(B),B        var(C),helper1(C),C
                                          R1:{B/true}|                   | R1:{C/true}
                                              helper1(true),true         helper1(true),true
                                                    |                        |
                                                 success                  success
                                               Solution AB              Solution AC
```

SUBTREE2:

```
                  helper2(D)
                     | R2:{_14/D}
                  var(D),helper1(D),D
                     | R1:{D/true}
                    helper1(true),true
                     |
                   success
                   Solution D
```

Marking:

3 marks for each correct branch leading to the solutions: AB, AC, D (total 9 marks)

mark deductions:
-2 for incorrect tree
-2 for missing entire branch that leading to fail
-1.5 for missing steps in the branch of **each** solution (the solution must be correct).
Other mistakes depending on the case -1 to -6

Problem 5. Flex and Bison (15 points)

Given the following grammar for polynomial expressions:

```
<polyn>            ::= <polyn><term> |<term>
<term>             ::= <signed_integer>  <var> <pow>  <signed_integer>
<signed_integer>   ::= <plus_minus> <digit_seq>
<digit_seq>        ::= <digit>|<digit><digit-seq>
<plus_minus>       ::= "+"|"-"|""
<pow>              ::= "^"
<var>              ::= "X"
<digit>            ::= 0|1|2|3|4|5|6|7|8|9
```

The Flex file for the grammar is given below:

```
%option noyywrap

%{
#define YYSTYPE int
#include "final.tab.h"
%}

ws [ \t]+
digits [0-9]
var [X]
plus_minus [+|-]
signed_integer [+|-]?(0|[1-9]+{digits}*)

%%
{ws}    /**/
{signed_integer} {yylval=atoi(yytext); return SIGNED_INTEGER;}
{var} return VAR;
"\n" return NEWLINE;
"^" return POW;
%%
```

Complete the following BISON file so that for each polynomial expression, the program will display the *degree* of the polynomial as follows (i.e. the highest power of the terms).

```
./run
+9X^-5-5X^-5
The degree of the polynomial expression is -5

./run
+9X^-5+9X^-2+15X^11
The degree of the polynomial expression is 11
```

```
%{
#define YYSTYPE int
#include <stdio.h>
int yylex(void);
int yyerror(const char*);
%}

/* Add your Bison tokens below*/
ANSWER:
%token SIGNED_INTEGER
%token NEWLINE
%token POW
%token VAR

/* Add your Bison tokens above*/
%%
/* Add your grammar rules and actions below */

input: input line
     | line
     ;
ANSWER:
line:  NEWLINE
     | polyn NEWLINE {printf("The degree of the polynomial
expression is %d\n",$1);}
     ;

polyn: polyn term {if ($$<$1) $$=$1; if ($$<$2) $$=$2;}
     | term     {if ($$<$1) $$=$1;}
     ;

term:  SIGNED_INTEGER VAR POW SIGNED_INTEGER {$$=$4;}
     ;

/* Add your grammar rules and actions above */

%%

int main(){    return yyparse();        }
int yyerror(const char* s){
    printf("%s\n", s);
    return 0;
}
```

Marking:

1 mark for each correct token (total 4 marks).
3 marks for the first line of grammar rule
4 marks for the second line of grammer rule (picking the biggest power)
4 marks for the third line of the grammar rule.

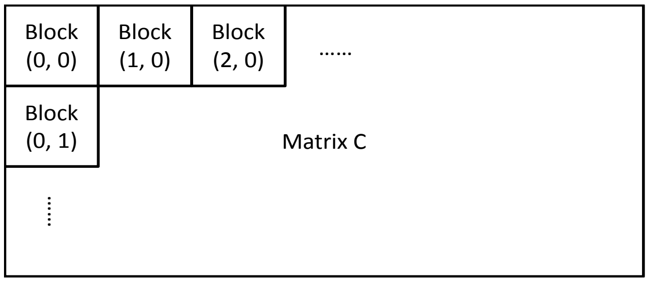
mark deductions:
-1 to -4 for different mistakes

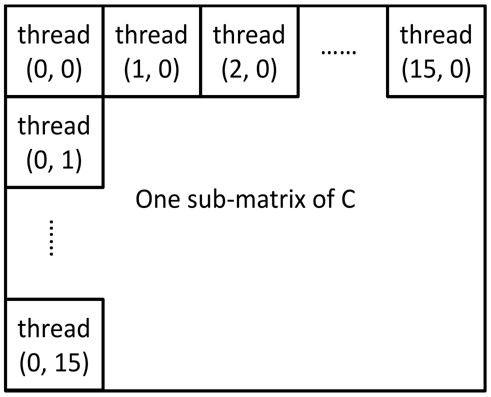
Problem 6. CUDA Parallel Programming (10 Points)

Complete the following CUDA program that performs matrix multiplication C = A×B. The input matrixes are A (with size 300×200) and B (with size 200×400), and the output matrix is C.

The program uses a two-dimensional thread grid. Each thread block in the grid is also two-dimensional. The number of threads in a thread block is 16×16.

Each thread block is responsible for the calculation of a 16×16 sub-matrix block of the result matrix C; each thread in a thread block is responsible for the calculation of one element in the corresponding sub-matrix. The thread blocks and threads in a block are mapped to the result matrix C and a sub-matrix of C, respectively, as shown in the following figure.





Fill in the following seven blanks in the code line in boldface:

```
__global__ void simpleMatrixMul
(int *d_A, int *d_B, int *d_C, int width_A, int height_A, int width_B)
{
      int row = blockIdx.y*blockDim.y+threadIdx.y;

      int col = blockIdx.x*blockDim.x+threadIdx.x;

      if((row < height_A) && (col < width_B))
      {
            int sum = 0;

            for(int k=0; k < width_A; k++)

                  sum += d_A[row*width_A+k_] * d_B[_k*width_B+col_];

                /* fill in the above blanks for multiplication */

            /* fill in the following blank to write the value to C */
            d_C[_row*width_B+col _] = sum;
      }
}


int main(int argc, char** argv)
{
      int height_A = 300; /* number of rows in matrix A */
      int width_A = 200;  /* #columns in A; also #rows in B*/
      int width_B = 400;  /* number of columns in B*/

      /*device pointer for matrix A, B and C */
      int *d_A, *d_B, *d_C;


      /*initialization and memory allocation; code omitted*/


      /*fill out the blanks to set the thread grid and block size */
      dim3 DimGrid (25, 19); (Any values larger than them are also
correct)
      dim3 DimBlock(_16_,_16_);

      simpleMatrixMul<<<DimGrid, DimBlock>>>(d_A, d_B, d_C, width_A,
      height_A, width_B);
      /*free memory; code omitted*/
}
```

Marking:

2 marks for the first three answers.

1 mark for the last four answers

mark deductions

-1 to -2 for the same wrong format in the first three blanks

Problem 7: Parameter Passing Methods (14 Points)

The following program is in an imaginary D language, which has a syntax similar to the C language, but can apply static or dynamic scoping as well as various parameter passing methods as we specify. Determine the output of the following D program with each specified scoping and parameter passing method:

```
int a = 1;
int b = 2;
int x = 3;
void funnyStuff(int c, int d)
{
    a = 5;
    b = 6;
    c = 7;
    d = 8;
    printf("(%d, %d, %d, %d)",a,b,c,d);
}
int main(int argc, char **argv)
{
    int x = b+2;
    funnyStuff(a,x);
    printf("(%d,%d,%d)",a,b,x);
}
```

Static scoping, call by value:

(5,6,7,8) (5,6,4)

Static scoping, call by reference:

(7,6,7,8) (7,6,8)

Static scoping, call by value-result:

(5,6,7,8) (7,6,8)

Dynamic scoping, call by name:

(7,6,7,8) (7,6,8)

Marking:
Each of the blank worth 3 points. And for each blank, one either gets 3 points if correct, or 0 otherwise. No partial credits given.

Problem 8. Activation Records (16 points)

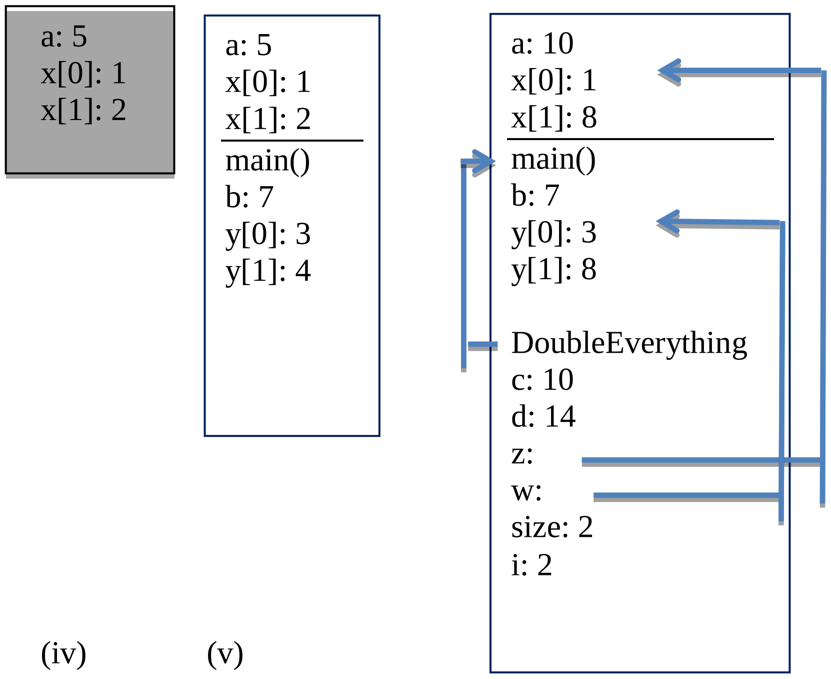
Recall that the C language by default uses static scoping on variable names and passing-by-value for parameter passing in procedure calls. Complete the activation records, including the variables and their values if known, the parameters and their values if known, and the control links for the following C program at specified point in time:

(i) right before calling main;
(ii) right before calling DoubleEverything
(iii) right before exiting the call of DoubleEverything
(iv) right before exiting main
(v) right after exiting main

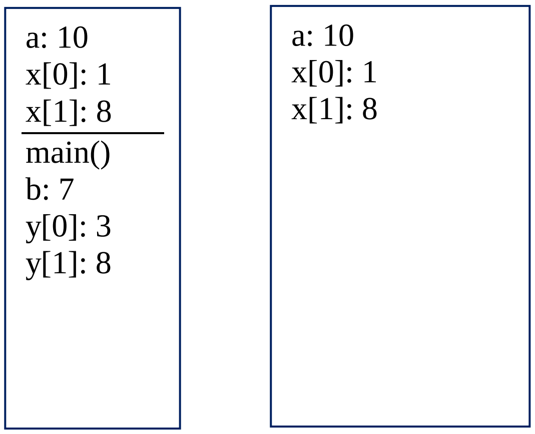
```
#include <stdio.h>
int a = 5;
int x[2] = {1,2};
void DoubleEverything(int c, int d, int z[], int w[], int size)
{
    int i=0;
    a = a*2;
    c = c*2;
    d = d*2;
    while (i++<size){
       x[i] = x[i]*2;
       z[i] = z[i]*2;
       w[i] = w[i]*2;
    }
}
int main(int argc, char **argv)
{
    int b = 7;
    int y[2] = {3,4};
    DoubleEverything(a,b,x,y,2);
}
```

Answer:
Activation Records:

(i) (ii) (iii)



(iv) (v)



Marking:
For (i),(ii),(iv),(v) worth 1,2,2,1 point(s) respectively
For (iii):
control links 1 point, reference links 3 points, proper changes to variable value in main and global 3 points, size, i, and c together with d, 1 point each