HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP3031 (Principles of Programming Languages)

Fall 2017

FINAL EXAMINATION

16:30PM - 19:30PM
Dec 12, 2017 Tuesday
LG5 Conference Room

Name SAMPLE SOLUTION	
Student ID	**ITSC Account**

1. *About the exam:*
    a. *This is a closed-book, closed-note examination.*
    b. *You CANNOT use any electronic devices including calculators 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. *This paper contains 17 pages, including this title page.*
    b. *The total number of points is 100, distributed to eight problems.*
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. *Attempt all questions.*

Problem	1	2	3	4	5	6	7	8	Total
Marks									

Problem 1. SML Programming (10 points)

The following SML datatype defines an encoded string:

```
datatype code = One of string | Many of int * string;
```

Example:

```
val t = [Many (3,"b"),Many (2,"a"),One "c",Many (2,"b")] : code list
```

`t` represents the code for the following list of strings

```
["b", "b", "b", "a", "a", "c", "b", "b"]
```

If the number of consecutive occurrences of string `entry` is equal to 1, we encode `entry` with `One entry`; if the number of consecutive occurrences is greater than 1, we encode the consecutive `entry's` with `Many (cnt, entry)`, where `cnt` is the number of consecutive occurrences of string `entry`.

a) Write a function `encode` to encode a list of strings into a `code list`.
`val encode = fn : string list -> code list`
The SML code skeleton is given. You only need to fill in the blanks, one expression per blank.

Examples:

```
- encode [];
val it = [] : code list
- encode ["a"];
val it = [One "a"] : code list
- encode ["a", "a"];
val it = [Many (2,"a")] : code list
- encode ["a", "a", "b", "a"];
val it = [Many (2,"a"),One "b",One "a"] : code list
- encode ["b", "b", "b", "a", "a", "c", "b", "b"];
val it = [Many (3,"b"),Many (2,"a"),One "c",Many (2,"b")] : code list
```

```
fun encode [] = []
| encode (h::t) =
let
     fun count (elem, [], cnt) = cnt
     | count (elem, h::t, cnt) =
     if elem=h then
     ___count(elem, t, 1+cnt)____ else cnt;
     fun skip (elem, []) = []
     | skip (elem, h::t) = if elem=h then _skip (elem, t)_
     else h::t;
in
     if __count (h, t, 1) = 1____
     then (One h)::(encode t)
     else (Many (count(h, t, 1), h))::(encode(skip(h, t)))
end;
```

b) Write a function decode to decode a code list into a list of strings.
`val decode = fn : code list -> string list`
The code skeleton is given. You only need to fill in the blanks, one expression per blank.

Examples:

```
- decode [];
val it = [] : string list
- decode [One "a"];
val it = ["a"] : string list
- decode [Many (2,"a")];
val it = ["a","a"] : string list
- decode [Many (2,"a"),One "b",One "a"];
val it = ["a","a","b","a"] : string list
- decode [Many (3,"b"),Many (2,"a"),One "c",Many (2,"b")];
val it = ["b","b","b","a","a","c","b","b"] : string list
```

```
fun decode [] = []
| decode ((One h)::t) = ____h::(decode t)____
| decode (__(Many (cnt, elem))::t__) =
let
     fun gen (0, elem) = []
     | gen (cnt, elem) = __elem::(gen(cnt-1, elem))__;
in
     __gen(cnt, elem)__ @ __(decode t)__
end;
```

Grading criteria: each blank 1pt, no partial points

Problem 2. Prolog Programming (10 points)

Given a knowledge base of facts `prerequisite(C, L),` where `C` is a course, `L` is a list of prerequisite courses of `C`. For example:

```
prerequisite(c01, []).
prerequisite(c10, []).
prerequisite(c11, [c01, m03]).
prerequisite(c21, [c11]).
prerequisite(c23, [c11, m12]).
```

a) Define a predicate `common_pres(X, Y, L),` which specifies that a list `L` of courses are prerequisites of two different courses `X` and `Y`. The code skeleton is given. You only need to fill in the missing predicates, one predicate per blank.

Examples:

```
?- common_pres(c10, c01, L).
L = [].

?- common_pres(c11, c21, L).
L = [].

?- common_pres(c23, c21, L).
L = [c11].

?- common_pres(c23, c21, [c01]).
false.

?- common_pres(c21, c23, [c01, c11]).
false.

?- common_pres(c21, c23, [c11]).
true.

?- common_pres(c21, Y, L).
Y = c01,
L = [] ;
Y = c10,
L = [] ;
Y = c11,
L = [] ;
Y = c23,
L = [c11].
```

```
common_pres(X, Y, L) :- prerequisite(X, Px),
                        prerequisite(Y, Py),
                        X \== Y,
                        comm(Px, Py, L, []).

  comm(Px, Py, [H|T], L) :- ________member(H, Px)_____,
                            ________member(H, Py)________,
                            ________\+member(H, L)_______,
                            ____comm(Px, Py, T, [H|L])_,
                            !.

  ___ comm(_, _, [], _)___ .
```

b) Define a predicate `not_prerequisite(C),` which specifies that a course `C` is not any other courses' prerequisites. The code skeleton is given. You only need to fill in the missing predicates, one predicate per blank.

Examples:

```
?- not_prerequisite(C).
C = c10 ;
C = c21 ;
C = c23.

?- not_prerequisite(c01).
false.

?- not_prerequisite(c11).
false.

?- not_prerequisite(c10).
true.

?- not_prerequisite(c23).
true.
```

```
not_in_pres(C, Cs) :- prerequisite(Course, Ps),
                      \+member(Course, Cs),
                      \+member(C, Ps),
                      __not_in_pres(C, [Course|Cs])_,

                      _______________!_______________.

not_in_pres(C, Cs) :- prerequisite(Course, Ps),
                     \+member(Course, Cs),
                      _______member(C, Ps) _________,

                      _______________!_______________,

                      _______________fail___________.
not_in_pres(_, _).
not_prerequisite(C) :- prerequisite(C, _),
                       not_in_pres(C, []).
```

Grading criteria: each blank 1pt, no partial points

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

Given the following Prolog database, write *all* the answers to each of the query a) – e):

```
p(a, 3).
p(b, 0).
p(c, 2).
s(1).
s(0) :- !.
s(2).
```

a)

```
?- s(Y), p(X,Y).
```

Y = 0,
X = b. (2pts)

b)

```
?- s(Y), \+p(X, Y).
```

Y = 1 ;
false. (2pts)

c)

```
?- p(X, Y), !, s(Y).
```

false. (2pts)

d)

```
?- p(X, Y), !, \+s(Y).
```

X = a,
Y = 3. (2pts)

e)

```
?- s(X), !, s(Y).
```

X = Y, Y = 1 ;
X = 1,
Y = 0. (2pts)

Problem 4. Prolog Search Tree (10 points)

Consider the following program:

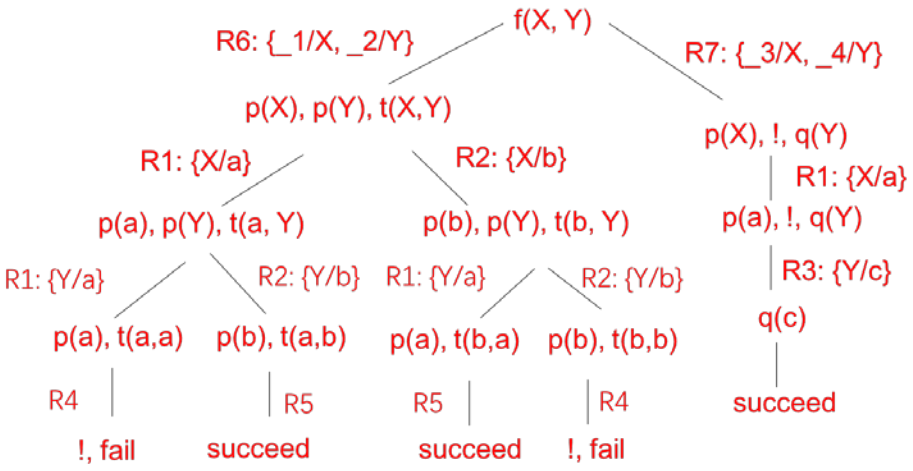
```
/*R1*/ p(a).
/*R2*/ p(b).
/*R3*/ q(c).
/*R4*/ t(X, X) :- !, fail.
/*R5*/ t(_, _).
/*R6*/ f(X, Y) :- p(X), p(Y), t(X, Y).
/*R7*/ f(X, Y) :- p(X), !, q(Y).
```

Draw the complete Prolog search tree for the query `f(X,Y)`, giving **all** answers. At each **tree edge**, whenever applicable, **i)** indicate the rule number Ri, i=1,..,7, of the rule being applied, and **ii)** the unification(s) being made. At each **tree node** indicate the goal to be satisfied. At each leaf node indicate "succeed" or "fail". The initial step is given.

f(X, Y)

R6: {_1/X, _2/Y}/

Solution:



Grading criteria: R6 result 1pt
R7 result 1pt
subtree from p(X), p(Y), t(X,Y) with R1 3pt
subtree from p(X), p(Y), t(X,Y) with R2 3pt
subtree from p(X), !, q(Y) with R1 2pt
Extra subtree -2pt
Missing rule -1pt

Problem 5. Flex and Bison (20 points)

Given the following grammar for FindA expressions:

```
<S> ::= <S> + <S1> | <S> - <S1> | <S1>
<S1> ::= <S1> * <S2> | <S1> / <S2> | <S2>
<S2> ::= <S3> <S2> | <S2>
<S3> ::= a | b | c
```

Complete the following Flex and Bison files to evaluate a FindA expression. The program will count the number of 'a's in each string and calculate the number of 'a's based on the operators in the expression. Some examples:

Input : aaaa+bbaa
Output : 6
Input : abac-bac
Output : 1
Input : aaa-ab*aaa
Output : 0
Input : abacaa/baa
Output : 2
Input : abaa+bac-aa
Output : 2
Input : aaa*aaa
Output : 9

Flex file "finda.lex": **Grade criterion: 1pt each blank**

```
%option noyywrap
%{
#define YYSTYPE int
#include "finda.tab.h"
#include <stdio.h>
%}
ws [ \t]+
%%
[abc] return *yytext;
"+"     return ADD  ;

"-"     return MINUS  ;

"*"     return DOT  ;

"/"     return DIV  ;

"\n"  return *yytext;
{ws}
%%
```

Bison file “finda.y”: **Grade criterion: 2pts each blank**

```
%{
#include <stdio.h>
int yylex(void);
int yyerror(const char*);
%}
%token ADD MINUS DIV DOT
%%
input: input line
      | line
      ;
line: '\n'
      | expr '\n' { printf("%d\n", $1);};
expr:   expr   ADD    expr_md {$$=$1+$3;}

      | expr   MINUS    expr_md {$$=$1-$3;}

      | expr_md {$$=$1;}
      ;
expr_md:  expr_md DOT exprh {   $$=$1*$3    ;}

      | expr_md DIV exprh {   $$=$1/$3    ;}

      | exprh {$$=$1;}
      ;
exprh:
      exprh atom  {   $$ = $1 + $2    ;}

      | atom            {$$ = $1;}
      ;
atom:   'a'              {$$ = 1;}

      |   'b'              {$$ = 0;}

      |   'c'              {$$ = 0;}

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

Problem 6. Parameter Passing Methods (10 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 i, j;
i = j = 1;
void comp(int x, int y)
{
      int i = 3;
      x = x * y;
      y++;
      y = x + i * 2;
      i++;
      printf("(%d,%d) ", i,j);
}
int main(int argc, char **argv)
{
      comp(i, j);
      printf("(%d,%d)", i,j);
      int j = 2;
      comp(i, j);
}
```

Static scoping, call by value:

(4,1) (1,1) (4,1)

Static scoping, call by reference:

(4,7) (1,7) (4,7)

Static scoping, call by value-result:

(4,1) (1,7) (4,7)

Dynamic scoping, call by name:

(4,9) (1,9) (7,18)

2.5 pts for each result

Problem 7. Activation Records (10 points)

Recall that the C language by default uses static scoping on variable names and call-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 the specified point in time:

(i) right before calling main;
(ii) right before calling calc;
(iii) right before exiting the call of calc;
(iv) right before exiting main;

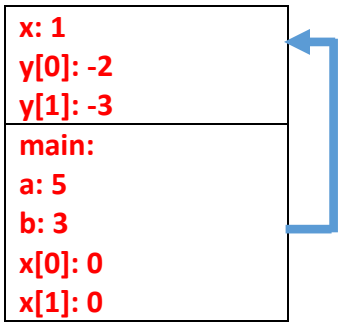
```
#include <stdio.h>

int x = 1;
int y[2] = {-2,-3};

void calc(int x, int a, int b[]){
      int i, len = 2;
      for(i=0;i<len;i++){
            y[i] = x * a;
            b[i] = y[i] + x;
      }
      return;
}

int main(int argc, char const *argv[])
{
      int a = 5, b = 3;
      int x[2] = {0,0};
      calc (a,b,x);
      return 0;
}
```

Activation Records:

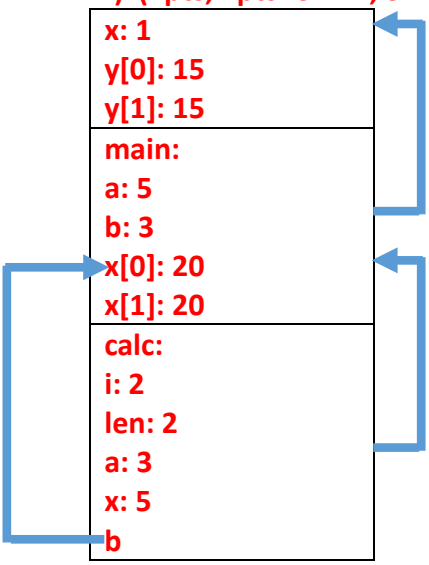
i) (1pt)

x: 1
y[0]: -2
y[1]: -3

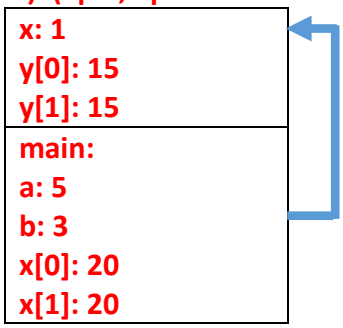
ii) (3pts, 1pt for each blank, 1pt for control link)



iii) (4pts, 2pts for 2nd, 3rd blank, 2pts for the control link)



iv) (2pts, 1pt for all blanks, 1pt for control link)



Problem 8. CUDA Parallel Programming (20 points)

(a) Complete the following CUDA program that counts the number of students with a grade greater than 70.

Example:
The input array A[NUM_OF_STUDENTS] is the grades of all students:

81.5	67	92	70.5	89	61.5	73

The output is 5, which is the number of entries in A that is greater than 70.

Fill in the blanks in the code line in bold face:

```
#define GRID_SIZE 8
#define BLOCK_SIZE 1024

/*Count the number of students with grades */
__global__ void count1 (float * d_grades, float grade, int * block_sum, int num)
{
   int global_tid = blockDim.x * blockIdx.x + threadIdx.x;

   int element_skip = _ blockDim.x * gridDim.x_; /*or BLOCK_SIZE*GRID_SIZE*/

   __shared__ int count[BLOCK_SIZE];
   int my_sum = 0;

   for (int i = ___global_tid___; i < num; i+= element_skip) {
         if (d_grades[i] > grade)
         my_sum += 1;
   }

   count[___threadIdx.x_____] = my_sum;

   __syncthreads();
   if(threadIdx.x == 0){
      int sum = 0;
      for (int i = 0 ; i < blockDim.x; i++){
         sum += count[i];
      }
      block_sum[___blockIdx.x____] = sum;
   }
}
```

```
__global__ void count2 (int * block_sum, int * count){
   if(threadIdx.x == 0 && blockIdx.x == 0){
      int all_sum = 0;

      //sum all block's result up to get the global summation
      for(int i = 0; i < __ gridDim.x_____; i++) { /*or GRID_SIZE*/
         all_sum += block_sum[i];
      }
      *count = all_sum;
   }
}

int main(int argc, char **argv){
   int n = 1000;              // size of input
   float *h_input_data; // host input grades
   float *d_input_data; // device input grades
   int *d_block_sum; // device count within a thread block
   int *d_cnt;           //  overall device count
   int h_cnt;           //  host result

   /* Allocate memory for host and device, code omitted*/
   /* Initialize host input grades, code omitted */

   /* Copy host input device to device */
   cudaMemcpy(d_input_data,h_input_data,sizeof(float)*n,cudaMemcpyHostToDevice);

   /* Configure grid and block */
   dim3 blocks(GRID_SIZE);
   dim3 threads(BLOCK_SIZE);

   /* Launch kernels */
   count1 <<<blocks, threads>>> (d_input_data, 70, d_block_sum, n);
   count2 <<<blocks, threads>>> (d_block_sum, d_cnt);

   /* Copy result back and check result */
   cudaMemcpy(&h_cnt, d_cnt, sizeof(int), cudaMemcpyDeviceToHost);
   printf("Number of students with grades > 70: %d\n", h_cnt);

   /* Free memory for host and device, code omitted */

   return 0;
}
```

(b) Complete the following CUDA program that rotates a matrix of size N*N to 90 degrees anticlockwise. The input matrix A and the output matrix B are both of size 200*200.

An example of a 3*3 matrix:

Input (before rotation):

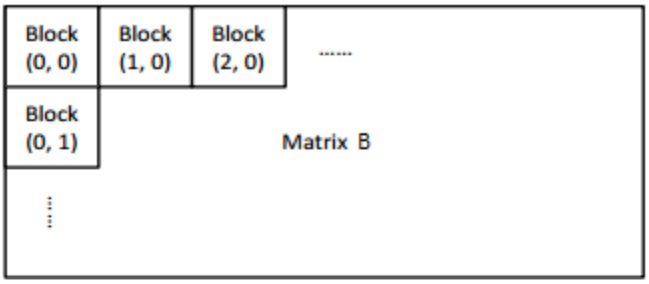
7	8	9
4	5	6
1	2	3

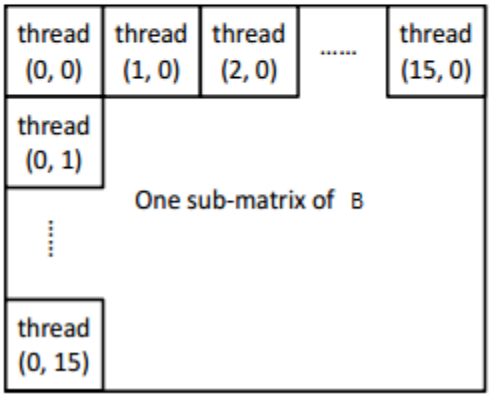
Output (after rotation):

9	6	3
8	5	2
7	4	1

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 of the result matrix B; 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 B and a sub-matrix of B, respectively, as shown in the following figure.





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

```
#define width_A 200
#define height_A 200

__global__ void Rotate(int *A, int *B)
{
        int row = blockIdx.y*blockDim.y+threadIdx.y;
        int col = blockIdx.x*blockDim.x+threadIdx.x;

        if (___row < width_A) && (col < height_A)_____) {

                B[row * height_A + col] = A[col * height_A + (width_A – row - 1)];
                /* or */
                B[(width_A - col - 1) * height_A + row] = A[row * width_A + col];
                /* fill in the above blanks for matrix rotation */
        }
}

int main()
{
        int size = width_A*height_A*sizeof(int);       // size of the matrices
        int *h_A = (int *)malloc(size);                // host pointer for matrix A
        int *h_B = (int *)malloc(size);                // host pointer for matrix B
        int *d_A, *d_B;                                // device pointer for matrix A and B

       /*Initialization of host matrix A and B, code omitted*/

        cudaMalloc((void **)&d_A,size);
        cudaMalloc((void **)&d_B,size);

        cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
        cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

        dim3 dimBlock( 16,  16 );
        dim3 dimGrid(__13, 13___);  /*numbers larger than 13 are correct*/
       /*fill in the above blank to set the thread grid size */

        Rotate<<<dimGrid,dimBlock>>>(___ d_A, d_B___);
       /*fill in the above blank to pass parameters to the kernel function */

        cudaMemcpy(h_B, d_B, size, cudaMemcpyDeviceToHost);
       /* data transmission from GPU to CPU */
```

```
        /*free memory; code omitted*/
}
```

Grading criteria: each blank 2pt, no partial points