HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP3031 (Principles of Programming Languages)

Fall 2014

FINAL EXAMINATION
16:30-19:30
Dec 9, 2014 Tuesday
Room LG1027

Name	
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 12 pages, including this title page.*
    b. *The total number of points is 100, distributed to seven 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. *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	Total
Marks								

Problem 1. SML Programming (20 points)

Note: You might find the following SML built-in functions useful.
`rev = fn : 'a list -> 'a list` (* reverse the order of the elements in the list *)
`length = fn : 'a list -> int` (* return the length of the list, `length [] =0` *)

a) Write a prefix-sum function on an input list A[0..n]. It returns a list B[0..n] where for i=0,..n, B[i] = ∑A[j], j=0,1,2,..,i. An optional helper function `sum` is provided.

```
val presum = fn : int list -> int list
```

Examples:

```
- presum [];
val it = [] : int list
- presum [1,2];
val it = [1,3] : int list
- presum [1,2,3,4];
val it = [1,3,6,10] : int list
```

```
fun sum [] = 0  | sum (h::t) = h + sum t; (* helper function *)

fun presum [] = [] |
    presum L  = (presum (rev (tl (rev L)))) @ [sum L]

sol2:
fun presum [] = []
  | presum L =
        let val reversed = rev L
            val total = sum reversed
            val newTail = rev (tl reversed)
            val recurseStep = presum newTail
         in
            recurseStep @ [total]
         end;
```

b) Given a list of *n* lists, write a function named `diagonal` to retrieve the first element of the first list, the second element of the second list, …, and the *n-th* element of the *n-th* list. Assume the *i-th* list has at least *i* elements, *i=1,2,..n.*

```
val diagonal = fn : 'a list list -> 'a list
```

Examples:

```
- diagonal [];
val it = [] : ?.X1 list
- diagonal [[1],[2,3],[4,5,6]];
val it = [1,3,6] : int list
- diagonal[["a","b"],["c","d"],["e","f","g","h"]];
val it = ["a","d","g"] : string list
```

```
(* helper function you can use *)
fun element(i, L) = if i=1 then hd L else element(i-1, tl L);

fun d2 n L = if n=0
             then []
             else (d2 (n-1) L) @ [element(n, element(n, L))];

fun diagonal L = d2 (length L) L;

sol2:
fun dia [] = []
  | dia L =
     let val reversed = rev L
         val size = length reversed
         val headRev = hd reversed
         val ele = element(size,headRev)
         val newTail = rev (tl reversed)
         val recurseStep = dia newTail
      in recurseStep @ [ele]
      end;
```

base case:2 marks

Problem 2. Prolog Programming (20 points)

Note: You might find the following PROLOG built-in predicates useful:
`length(L, N)` : N is the length of L. `length([],0).`
`append(X,Y,Z)` : Z is X appended by Y.

a) Write a predicate `triangular(X, Y)` in which X is a given *n*-element list, Y is a list containing the first element of X, followed by the first two elements of X, followed by the first three elements of X, ..., followed by the first *n* elements of X. The code skeleton is given. You only need to fill in the missing predicates, one predicate per blank.

Examples:

```
?- triangular([1,2],L).
L = [1, 1, 2].
?- triangular([a,b,c],L).
L = [a, a, b, a, b, c].
?- triangular([a, b, c, d], [a, a, b, a, b, c, a, b, c, d]).
true.
```

```
/*firstN(L1,N,L2): L2 contains the first N elements of the given L1.*/
  firstN([X|_],1,[X]) :- !.
  firstN([X|T],N,[X|L]) :- N > 1,

                           ___ N1 is N-1____,

                           ____ firstN(T,N1,L)_____.

  /*dupN(L1,N,L2): L2 contains the first element, followed by the
  first two elements,..., followed by the first N elements of L1. */

  dupN(L1,0,[]) :- !.
  dupN(L1,N,L) :- N > 0,
                 N1 is N-1,

                 __ dupN(L1,N1,L2)_________,

                 ___firstN(L1,N,L3)_____,

                 ___append(L2,L3,L)___.

  triangular(L1,L) :- length(L1,N), dupN(L1,N,L),!.
```

b) Write a predicate `median(L,M)` such that for a given list L with an odd number of numeric values, M is the median of all the values. Assume all values in the list are different. Hint: The median of a list of 2N+1 distinct numbers has N numbers in the list greater than it and N numbers less than it.

Examples:

```
?- median([-4],M).
M = -4.

?- median([2,3.1,1.2],M).
M = 2.

?- median([6,3,2.5,-7.8,12],M).
M = 3.
```

```
/* split(X, L, L1, L2) :
L1 contains all elements of L less than X and L2 the rest */

split(X, [], [], []).

split(X, [Y | Y1], [ Y | Less ], NotLess)
  :- Y < X,

     ___split(X, Y1, Less, NotLess)___.

split(X, [Y | Y1], Less, [ Y | NotLess ])
  :- Y >= X,

     ____split(X, Y1, Less, NotLess)____.

median(L, M):- member(M, L),
               split(M, L, A, B),

               _length(A, N)____,

               _length(B, N1)________,

               _N1 is N+1___or N1 =:= N+1____,

               !.
Sol2: length(L,N), N1 is (N-1)div 2, length(A,N1).
```

Each blank, 2 marks

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

Given the following Prolog program, write the *first* answer to each query.

```
a(ala).
a(ama).
b(bala).
b(bama).
f(X,Y) :- a(X), b(Y).
f(X,Y) :- b(X), \+(X=Y), b(Y).
f(X,Y) :- b(X), !, a(Y).
```

a)

```
?- f(bama,bala).
```

true.

b)

```
?-f(bama,bama).
```

false.

c)

```
?- f(bama,ama).
```

true.

d)

```
?- f(X, ama).
```

X=bala.

e)

```
?- f(ala,ama).
```

false.

Problem 4. Prolog Search Tree (10 points)

Consider the same program as in Problem 3:

```
/*R1*/ a(ala).
/*R2*/ a(ama).
/*R3*/ b(bala).
/*R4*/ b(bama).
/*R5*/ f(X,Y) :- a(X), b(Y).
/*R6*/ f(X,Y) :- b(X), \+(X=Y), b(Y).
/*R7*/ f(X,Y) :- b(X), !, a(Y).
```

Draw the complete Prolog search tree for the query `f(bama,X)`, giving **all** answers. 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. At each leaf node indicate "succeed" or "fail". The initial step has been done for you.

```
                         f(bama,X)
R5: {_1/bama, _2/X}/          R6\{_3/bama,_4/X}            \R7:{_5/bama,_6/X}
              a(bama), b(X).   b(bama), \+(bama=X),b(X)   b(bama),!,a(X)
             /                 |                       R1: {X/ala}/    \R2: {X/ama}
            Fail               \+(bama=X),b(X)           succeed        succeed
                                |
                               Fail|
```

R5: 2 marks
R6: 4 marks
R7: 4 marks (lack one branch, -2marks)

Problem 5. Flex and Bison (20 points)

Given the following grammar for list expressions:

```
<LINE> ::= <NL>|<LIST> + <LIST><NL>|<LIST> * <LIST><NL>
<LIST> ::= [ <VAL> <VALS>]
<VALS> ::= ,<VAL><VALS>|<empty>
<VAL> ::= <DIGIT>|<DIGIT><VAL>
<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
<NL> ::= \n
```

The Flex file for the tokens in the grammar is given below:

```
%option noyywrap
%{
#include <vector>
#include <stdlib.h>
struct listnode
{
    int val;
    listnode *next;
};
#define YYSTYPE listnode *
#include "list.tab.h"
%}

ws [ \t]+
digits [0-9]
op [+|*]
val {digits}+
%%
{ws} /**/
[\[\],]      return *yytext;
{op}         return *yytext;
[\n]         return NL;
{val}        {listnode *node =
              (listnode *)malloc(sizeof(struct listnode));
              node->val = atoi(yytext);
              node->next = NULL;
              yylval = node;
              return VAL;}
%%
```

Complete the BISON file so that, for each input list expression, the program will display the result of the operation – addition (+) or multiplication (*) of the corresponding elements in the two lists of the same length. Examples (**input** in **bold face**):

[1,2,3]+[4,5,6]
[5,7,9]
[1, 3,   5]* [2, 4,6]
[2,12,30]

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

/**** Helper functions ***/
listnode *push_front(listnode *val, listnode *head);
/* add val to the front of the linked list, and link its next
pointer to the old head. e.g. head=>2->1, after pushing a val(3) to
this linked list: push_front(val, head), the list becomes: head=>3-
>2->1, the return value is the new head.
 */

listnode *listadd(listnode *l1, listnode *l2);
/* Perform list1 + list2. The two linked lists have the same
length. The result is stored in l1, the return value is l1.
 */

listnode *listmultiply(listnode *l1, listnode *l2);
/* Perform list1 * list2. The two linked lists have the same
length. The result is stored in l1, the return value is l1.
 */

void destroy(listnode *head); /*deallocate the list*/

void output(listnode *head); /*display the list*/
/***************************************/
%}

%token VAL
%token NL

%%
input: input line {output($2); destroy($1);}
   | line {output($1); destroy($1);}
   ;
line: NL
   | list '+' list NL {$$ = listadd($1, $3); destroy($3); }
   | list '*' list NL {$$ = listmultiply($1, $3); destroy($3);}
   ;
list: '[' VAL vals ']' {$$ = push_front($2, $3);}
   ;
vals: /*empty*/ {$$ = NULL;}
   | ',' VAL vals  {$$ = push_front($2, $3);}
   ;
%%
int main(){ return yyparse(); }
int yyerror(const char* s){ return 0; }

…
```

For left 4 blanks, 3 marks / blank.
For right 6 blanks, 1 marks / blank.

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 a = 2;
int b = 3;
int x = 4;
void funnyStuff(int b, int a)
{
    a = a+x;
    x = b*x;
    printf("(%d, %d)",a,x);
}
int main(int argc, char **argv)
{
    int x = a+b;
    funnyStuff(a,x);
    printf("(%d,%d,%d)",a,b,x);
}
```

Static scoping, call by value:

(9, 8) (2,3,5)

Static scoping, call by reference:

(9, 8)(2,3,9)

Static scoping, call by value-result:

(9, 8) (2,3,9)

Dynamic scoping, call by name:

(20, 20) (2,3,20)

Each number, 0.5 mark.

Problem 7. Activation Records (10 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:

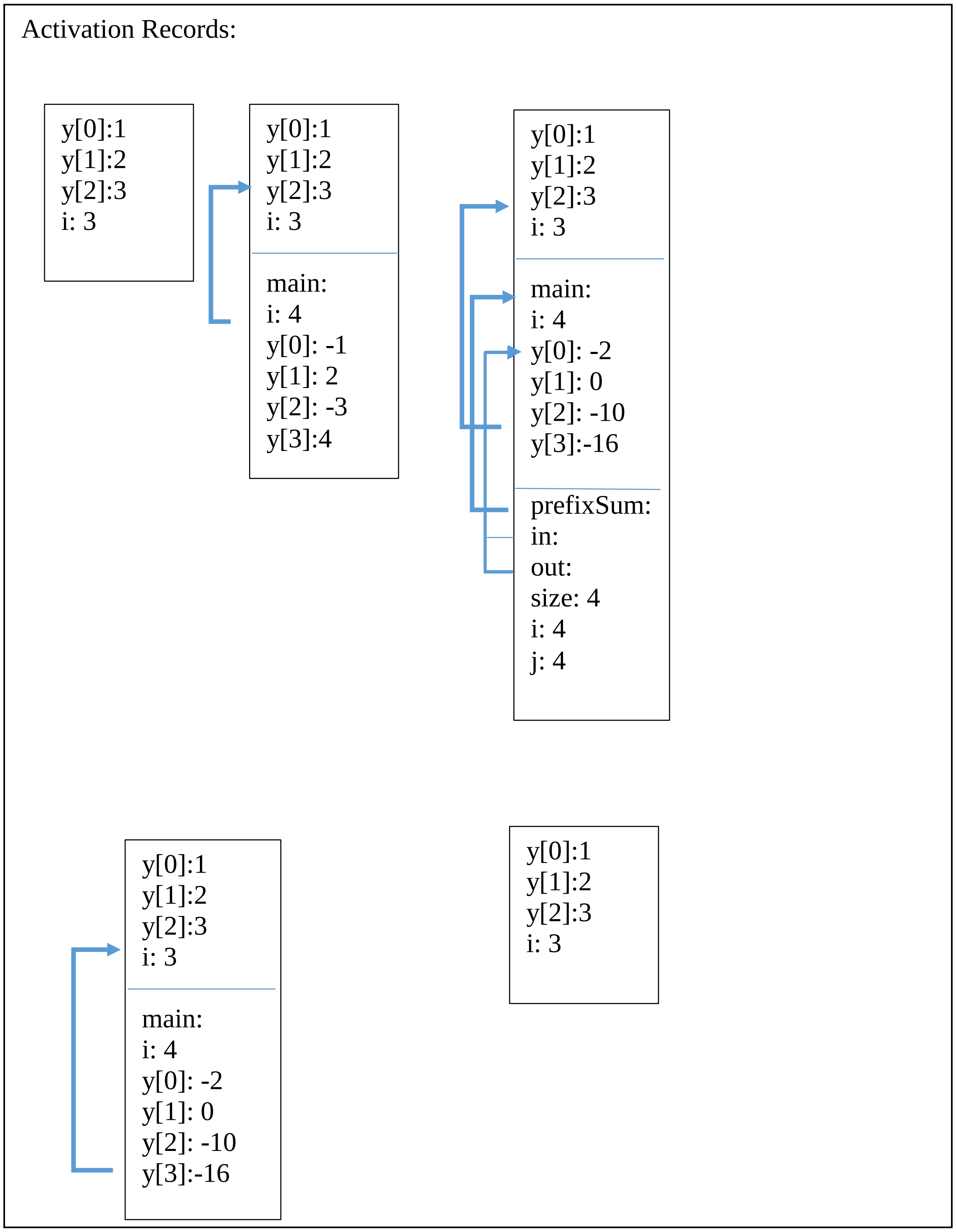
(i) right before calling main;
(ii) right before calling prefixSum;
(iii) right before exiting the call of prefixSum;
(iv) right before exiting main;
(v) right after exiting main.

```
#include <stdio.h>

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

void prefixSum(int in[], int out[], int size)
{
    int i=0, j=0;
    for (i=0; i<size; i++)
         for (j=0; j<=i; j++)
             out[i]+=in[j];
}

int main(int argc, char **argv)
{
    int i = 4;
    int y[4] = {-1,2,-3,4};
    prefixSum(y, y,i);
 }
```



(i) to (v) each step 2 marks