

HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY
COMP3031 (Principles of Programming Languages)

Fall 2015

FINAL EXAMINATION

12:30PM - 3:30PM
Dec 10, 2015 Thursday
LG5 Multi-function Room

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 14 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)

Given the following SML datatype:

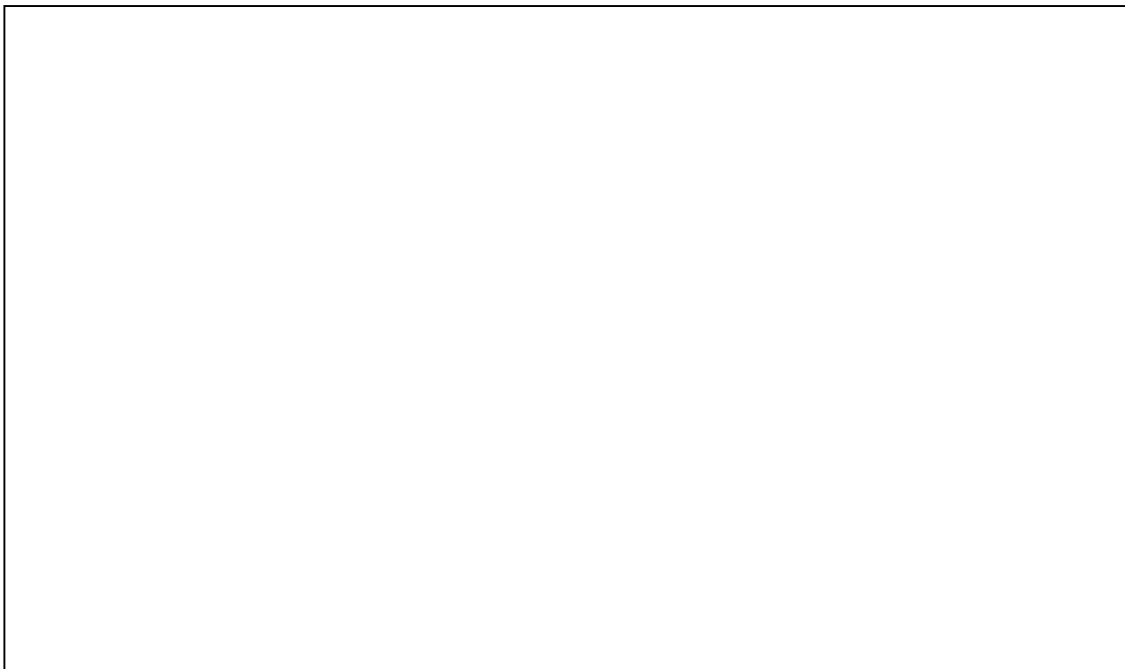
```
datatype 'a tree = empty_tree |  
                  leaf of 'a |  
                  node of 'a * 'a tree * 'a tree;
```

- a) Write a function `path` to return a list of labels of the tree nodes that form the path from the root to the specified label, if there is such a path in the tree; otherwise, return `nil`. The list is generated through pre-order traversal, and only the first path found is returned.

```
val path = fn : 'a -> 'a tree -> 'a list
```

Examples:

```
-val x = node(0, node(1, leaf(2), leaf(3)), node(2, leaf(3), empty_tree));  
val x = node (0,node (1,leaf #,leaf #),node (2,leaf #,empty_tree)) :  
int tree  
- path 0 x;  
val it = [0] : int list  
- path 1 x;  
val it = [0,1] : int list  
- path 2 x;  
val it = [0,1,2] : int list  
- path 3 x;  
val it = [0,1,3] : int list  
- path 4 x;  
val it = [] : int list  
- path true (leaf false);  
val it = [] : bool list
```

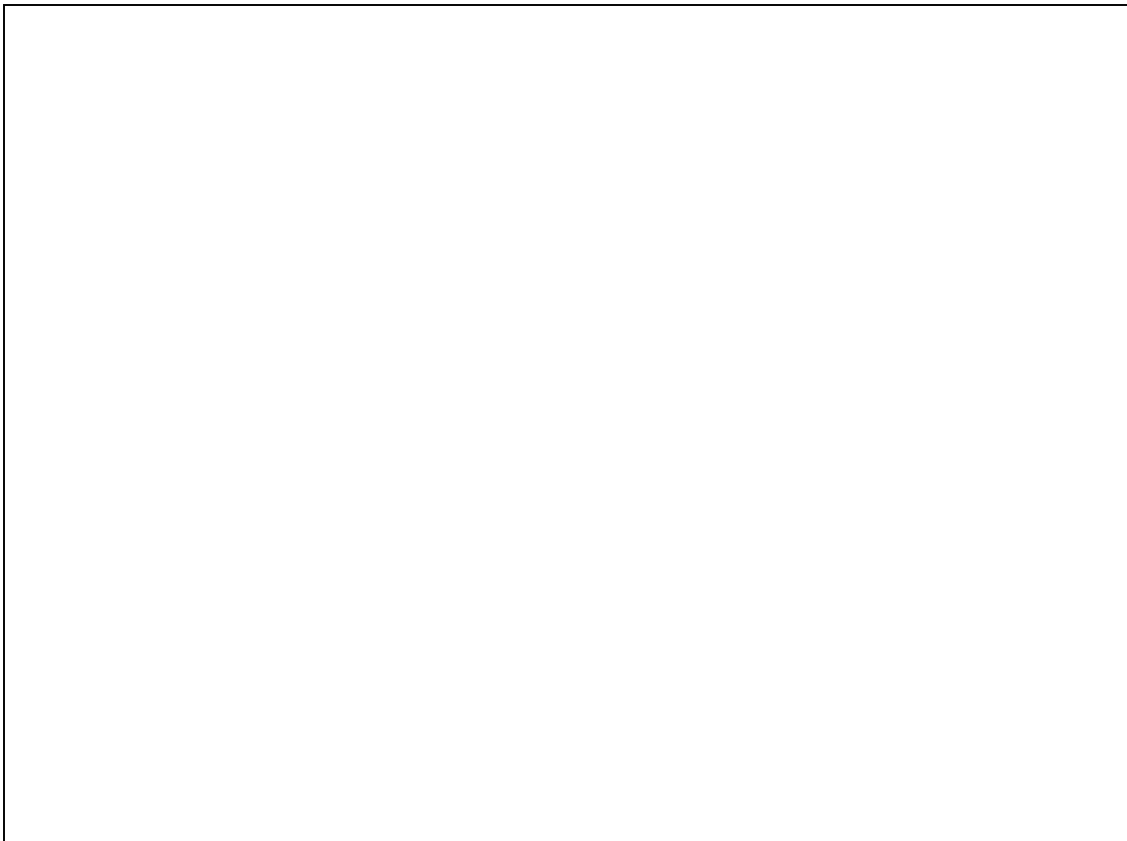


- b) Write a function `height` to return the maximum height of a specified label in the tree. If the label is not in the tree, return `~1`.

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

Examples:

```
- val x = node("a", node("b", leaf("c"), leaf("d")),
node("c",leaf("d"), empty_tree));
val x = node ("a",node ("b",leaf #,leaf #),node ("c",leaf
#,empty_tree)) : string tree
- height "a" x;
val it = 3 : int
- height "b" x;
val it = 2 : int
- height "c" x;
val it = 2 : int
- height "d" x;
val it = 1 : int
- height "e" x;
val it = ~1 : int
- height true empty_tree;
val it = ~1 : int
- height ~1 (leaf ~1);
val it = 1 : int
```



Problem 2. Prolog Programming (20 points)

Given a knowledge base of facts in the form of `node(X, L, R)`, where `X` is the label of a tree node, and `L` and `R` labels of the left and right children of the node. The set of facts represents a binary tree where all labels of the tree nodes are unique:

```
node(a,b,c).
node(b,d,e).
node(e,f,g).
```

- a) Define a predicate `path(X, L)` in which `X` is the label of a tree node, and `L` is a list of labels that form the path from the root node of the tree to `X`, if `X` is in the tree; otherwise, `L` is an empty list. The code skeleton is given. You only need to fill in the missing predicates, one predicate per blank.

Examples:

```
?- path(c,X).
X = [a, c].
?- path(X,[a,c]).
X = c.
?- path(g,X).
X = [a, b, e, g].
?- path(h,X).
X = [].
```

<pre>path(X, []) :- _____, _____, _____, !.</pre>
<pre>path(X, [X]) :- _____, _____, _____, !.</pre>
<pre>path(X, L) :- node(P, X, _), _____, _____, !.</pre>
<pre>path(X, L) :- node(P, _, X), _____, _____, !.</pre>

- b) Define a predicate `height(X,H)` such that for a given node `X`, `H` is the maximum height of the node in the tree, if the node is in the tree; otherwise, the predicate returns false. The code skeleton is given. You only need to fill in the missing predicates, one predicate per blank.

Examples:

```
?- height(a,X).
X = 4.
?- height(b,X).
X = 3.
?- height(c,X).
X = 1.
?- height(d,X).
X = 1.
?- height(e,X).
X = 2.
?- height(f,X).
X = 1.
?- height(h,X).
false.
```

```
max(A,B,A) :- A>B, !.
max(A,B,B).

height(A,1) :- node(_,A,_), \+ node(A,_,_), !.
height(A,1) :- node(_,_,A), \+ node(A,_,_), !.

height(X, H) :- _____,
                  _____,
                  _____,
                  _____,
                  _____, !.
```

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

Given each of the Prolog programs a) – e), write the **first** answer to the query:

?- likes(X,a).

a)

```
bear(a).
animal(a).
likes(X, Y) :- Y=b, !, X=d.
likes(c, Y) :- animal(Y).
```

b)

```
bear(a).
animal(a).
likes(X, Y) :- Y=a, !, X=d.
likes(X,Y) :- animal(Y).
```

c)

```
bear(a).
animal(a).
likes(c, X) :- \+ X=a.
likes(c, X) :- \+ animal(X).
```

d)

```
bear(a).
animal(a).
likes(c, X) :- bear(X), !, fail.
likes(c, X) :- animal(X).
```

e)

```
bear(a).
animal(a).
likes(c, X) :- \+ bear(X).
likes(c, X) :- animal(X).
```

Problem 4. Prolog Search Tree (10 points)

Consider the following program:

```
/*R1*/ basic(enq).  
/*R2*/ basic(ref).  
/*R3*/ over(john).  
/*R4*/ over(walter).  
/*R5*/ gen(book).  
/*R6*/ add(walter, enq).  
/*R7*/ fac(P, F) :- add(P, F).  
/*R8*/ fac(P, F) :- over(P), !, basic(F).  
/*R9*/ fac(P, F) :- gen(F).
```

Draw the complete Prolog search tree for the query `fac(X,Y)`, giving **all** answers. At each **tree edge**, whenever applicable, **i)** indicate the rule number R_i , $i=1,\dots,9$, 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.

```
                fac(X,Y)  
R7: {_1/X, _2/Y}/  
      add(X,Y)
```

Problem 5. Flex and Bison (20 points)

Given the following grammar for expressions on a binary tree where every non-empty tree node has a numeric label:

```
<expression> ::= sum(<tree>) | product(<tree>)
<tree> ::= empty | node(<num>,<tree>,<tree>)
<num> ::= <D1><N> | <N>
<D1> ::= <D1><N> | <D2>
<D2> ::= [1-9]
<N> ::= [0-9]
```

An expression is either a *sum* or a *product*. A *sum* operation adds up the numeric label values of all nodes in the tree. A *product* operation, multiplies the numeric label values of all nodes in the tree. An empty node has a numeric value of 1 in a *product* and 0 in a *sum*. Some examples:

$\text{sum}(\text{node}(4,\text{empty},\text{empty})) = 4+0+0 = 4$

$\text{sum}(\text{node}(9,\text{node}(5,\text{empty},\text{node}(2,\text{empty},\text{empty})),\text{node}(1,\text{empty},\text{empty})))$
 $= 9+(5+0+(2+0+0))+(1+0+0) = 17$

$\text{product}(\text{node}(12,\text{empty},\text{empty})) = 12*1*1 = 12$

$\text{product}(\text{node}(20,\text{node}(6,\text{empty},\text{node}(4,\text{empty},\text{empty})),\text{node}(7,\text{empty},\text{empty})))$
 $= 20*(6*1*(4*1*1))*(7*1*1) = 3360$

Complete the Flex and Bison files so that when they are compiled and run, it will give the following output on the **input**:

```
sum(node(4,empty,empty))
4
sum(node(9,node(5,empty,node(2,empty,empty)),node(1,empty,empty)))
17
product(node(12,empty,empty))
12
product(node(20,node(6,empty,node(4,empty,empty)),node(7,empty,empty)))
3360
```


Flex file “tree.lex”:

```
%option noyywrap

%{
struct treenode
{
    int sum;
    int product;
};

#define YYSTYPE treenode
#include "tree.tab.h"

%}

num _____
op [(),\n]
ws [ \t]+

%%
{num}      { _____

                return NUM;}
{op}        return *yytext;
{ws}
empty       return EMPTY;
sum         return SUM;
product     return PRODUCT;
node
%%
```

Bison file “tree.y”:

```
%{
#include <iostream>
using namespace std;

struct treenode
{
    int sum;
    int product;
};

#define YYSTYPE treenode
int yylex(void);
int yyerror(const char*);

}%

%token NUM
%token EMPTY
%token SUM
%token PRODUCT

%%

/* Fill in the blanks in the grammar rules and actions*/

input: /* empty */ | input line ;

line: '\n'

|SUM '(' tree ')' '\n' { cout << _____ << endl;}

| PRODUCT '(' tree ')' '\n' { cout << _____ << endl;};

tree:

_____

_____

%%

int main() { return yyparse();}

int yyerror(const char* s) {
    cout << "error" << endl;
    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 x = 3;
int y = 2;
void final(int a, int b)
{
    a++;
    b = x - 2;
    printf("(%d,%d)",x,y);
    x--;
    y = b + 1;
    printf("(%d,%d)",x,y);
}
int main(int argc, char **argv)
{
    int x = 4;
    final(y, x);
    printf("(%d,%d)",x,y);
}
```

Static scoping, call by value:

Static scoping, call by reference:

Static scoping, call by value-result:

Dynamic scoping, call by name:

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 mul;
- (iii) right before exiting the call of mul;
- (iv) right before exiting main;
- (v) right after exiting main and before the program terminates.

```
#include <stdio.h>

int x = 3;
int y = 4;
int a[2] = {5, 6};

void mul(int x, int b, int z[ ])
{
    int i=0, size=2;
    for (i=0; i < size; i++) {
        a[i] *= x++;
        z[i] *= b++;
    }
    y=x*b;
}

int main()
{
    int y = 5;
    int b[2] = {2, 3};
    mul(y, x, b);
}
```

Activation Records:

Additional paper for draft work