HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP3031 (Principles of Programming Languages)

Fall 2012

FINAL EXAMINATION

Dec 21, 2012 (Fri) 12:30PM - 2:30PM, LG1 Table Tennis Room

Name **SAMPLE SOLUTION**	**ITSC Account**
Student ID	**Lecture Section** L1

1. *<u>About the exam:</u>*
    a. *This is a closed-book, closed-notes examination.*
    b. *You CANNOT use any electronic device including calculator during the examination. Please TURN OFF all of your electronic devices (e.g., mobile phone) and put them in your bag.*
    c. *You cannot leave during the last 15 minutes of the examination.*
2. *<u>About this paper:</u>*
    a. *There are a total of 11 pages, including this page.*
    b. *There are a total of 8 questions counting for 100 points in total.*
3. *<u>About your answers:</u>*
    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". However, do not write answers there as they will NOT be graded.*
    d. *Attempt all questions. Partial credit will be given as applicable.*

Problem	1	2	3	4	5	6	7	8	Total
Points									

Problem 1 (10 points) SML Programming

A CSE enrollment record is represented as a 2-integer tuple (CID, SID) where CID is the course ID, e.g., 3031, and SID the student ID. Given a list of enrollment records, the department needs to group them by courses, and an instructor needs to know the student IDs in a specific course. Order in the list does not matter.

(a) Write a function *group*: (int * int) list -> (int * int list) list to group enrollment records by CIDs. E.g.,
- group [(3031, 111), (3311,111), (3031, 222), (6111,333)];
val it = [(6111,[333]),(3031,[111,222]),(3311,[111])] : (int * int list) list

```
fun addStudent((Cid,Sid),[]) = [(Cid,[Sid])] |
   addStudent((Cid,Sid),(CidH,Ss)::Cs) =
      if Cid = CidH
         then (CidH,(Sid::Ss))::Cs
      else (CidH,Ss)::(addStudent((Cid,Sid),Cs));

fun group([])=[]|
   group((((Cid: int),(Sid:int))::Ts)) = addStudent((Cid,Sid),group(Ts));
```

(b) Write a function *roster*: (int * int) list -> int -> int list to return the SIDs of the enrolled students in a specific course. E.g.,
- roster [(3031, 111), (3311,111), (3031, 222), (6111,333)] 3031;
val it = [111,222] : int list

```
fun roster [] (Cid:int) = [] |
   roster ((CidH,(Sid:int))::Cs) Cid =
      if Cid = CidH
         then Sid::(roster Cs Cid)
      else roster Cs Cid;
```

Problem 2 (10 points) Prolog Programming

A CSE enrollment record is represented as a 2-element list [CID, SID] where CID is the course ID, e.g., 3031, and SID the student ID. Given a list of enrollment records, the department needs to group them by CIDs as well as count the number of students in each course. Order in the list does not matter.

(a) Write a predicate *group(X,Y)* such that for a given list of enrollment records in X, Y is a list of 2-element lists where each 2-element list is a CID followed by a list of enrolled SIDs. Example:

```
?- group([[3031, 111], [3311,111], [3031, 222], [6111,333]],Y).
Y = [[6111, [333]], [3031, [111, 222]], [3311, [111]]].
```

```
group([],[]).
group([[Cid,Sid]|Ts],RList) :-
    group(Ts,RTail),addStudent([Cid,Sid],RTail,RList).

addStudent([Cid,Sid],[],[[Cid,[Sid]]]) :- !.
addStudent([Cid,Sid],[[Cid,Ss]|Cs],[[Cid,[Sid|Ss]]|Cs]) :- !.
addStudent([Cid,Sid],[[CidH,Ss]|Cs],[[CidH,Ss]|CsNew]) :-
addStudent([Cid,Sid],Cs,CsNew).
```

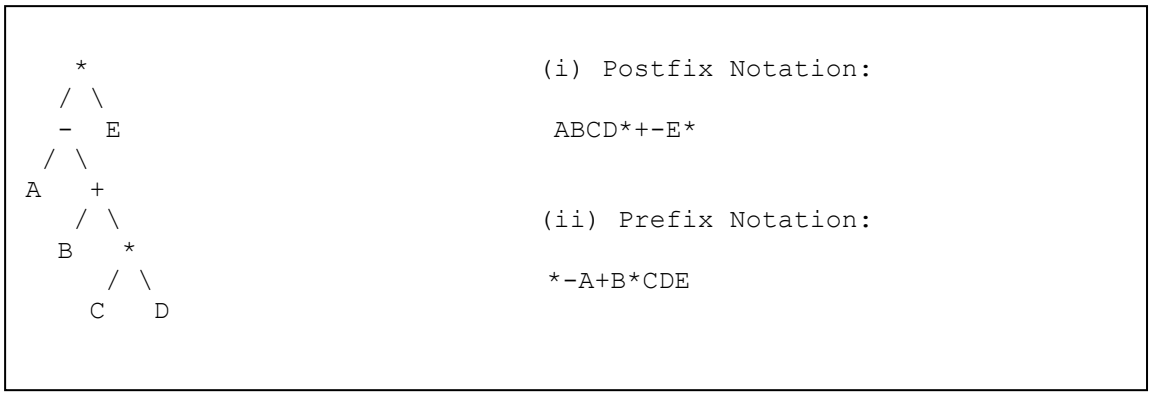
(b) Write a predicate *count(X,Y)* such that for a given list of enrollment records in X, Y is a list of 2-element lists where each 2-element list is a CID followed by the number of enrolled SIDs. Example:

```
?- count([[3031, 111], [3311,111], [3031, 222], [6111,333]],Y).
Y = [[6111, 1], [3031, 2], [3311, 1]].
```

```
count([],[]).
count([[Cid,Sid]|Ts],Count) :-
    count(Ts,CTail),countIn([Cid,Sid],CTail,Count).

countIn([Cid,_],[],[[Cid,1]]) :- !.
countIn([Cid,_],[[Cid,Count]|Cs],[[Cid,CountNew]|Cs]) :- CountNew is Count+1, !.
countIn([Cid,_],[Head|Cs],[Head|CsNew]) :-
    countIn([Cid,_],Cs,CsNew).
```

Problem 3 (12 points) Grammars and Regular Expressions

(a) For the following tree representation of an arithmetic expression, write the expression in (i) postfix notation and (ii) prefix notation.



(b) Write an **unambiguous** context-free grammar to describe a language in which both 'a' and 'b' are valid strings and all other valid strings are formed by the following rules with the **precedence** from high to low:
(i) Given a valid string X, '('X')' also belongs to the language;
(ii) Given a valid string X, X'*', X'+', and X'?' also belong to the language;
(iii) Given two valid strings X and Y, XY also belongs to the language.
(iv) Given two valid strings X and Y, X'|'Y also belongs to the language.

e.g., `b+(a|a)*a?` belongs to the language, but `+ba|)` does not.

Make sure the grammar observe the precedence rules.

<S> ::= <D> '|' <S> |<D>
<D>::= <C><D> | <C>
<C>::= <C>'*' | <C>'+' |<C>'?' | <T>
<T>::= (<S>) | 'a' | 'b'

Problem 4 (15 points) Flex and Bison

Given the following grammar:

<Sentence> ::= <Word> | <Word> <Sentence>
<Word> ::= <Letter> ' ' | <Letter> <Symbol> | <Letter> <Word>
<Letter> ::= <Vowel> | <Nonvowel>
<Symbol> ::= . | ? | ! | ;
<Vowel> ::= a | e | i | o | u
<Nonvowel> ::= b | c | d | f | ... | z

The Flex file for Words specified by this grammar is also given:

```
%option noyywrap

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

op  [.?!;]
ws  [\t]+

%%
[aeiou]   return VOWEL;
[b-df-hj-np-tv-z] return NONVOWEL;
{op}    return SYMBOL;
[ ]     return SPACE;
\n      return *yytext;
{ws}
%%
```

Complete the following BISON file to count the total number of occurrences of the vowels in a string that satisfies the grammar:

```
%{
#define YYSTYPE int
#include <stdio.h>
%}

%token VOWEL
%token NONVOWEL
%token SYMBOL
%token SPACE

%%
input: /* empty */
 | input line
 ;

line: '\n'
 | sentence '\n' { printf("\t%d\n", $1); }
 ;

/* ADD BISON PRODUCTION RULES AND ACTIONS HERE FOR THE GIVEN CFG */

/* Sample answer */
sentence: word   {$$ = $1;}
      | word sentence  {$$ = $1+$2;}
       ;

word: letter SPACE { $$ = $1; }
      | letter SYMBOL { $$ = $1; }
      | letter word { $$ = $1+$2; }
       ;

letter: VOWEL { $$ = 1; }
      | NONVOWEL { $$ = 0; }
       ;
/* End of answer */

%%

int main()
{
    return yyparse();
}

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

Problem 5 (10 points) Cuts in Prolog

Consider each of the following Prolog queries and write the first answer (*true* or *false*) for the query:
?- f(q).

(a)
```
f(X) :- !, X = p.
f(X) :- !, X = q.
f(X) :- X = r.
```

false.

(b)
```
f(X) :- X = p, !.
f(X) :- X = q, !.
f(X) :- X = r.
```

true.

(c)
```
f(X) :- X = p, !.
f(X) :- !, X = q.
f(X) :- X = r.
```

true.

(d)
```
f(X) :- !, X = p.
f(X) :- X = q, !.
f(X) :- X = r.
```

false.

(e)
```
f(p) :- !.
f(q) :- !.
f(r).
```

true.

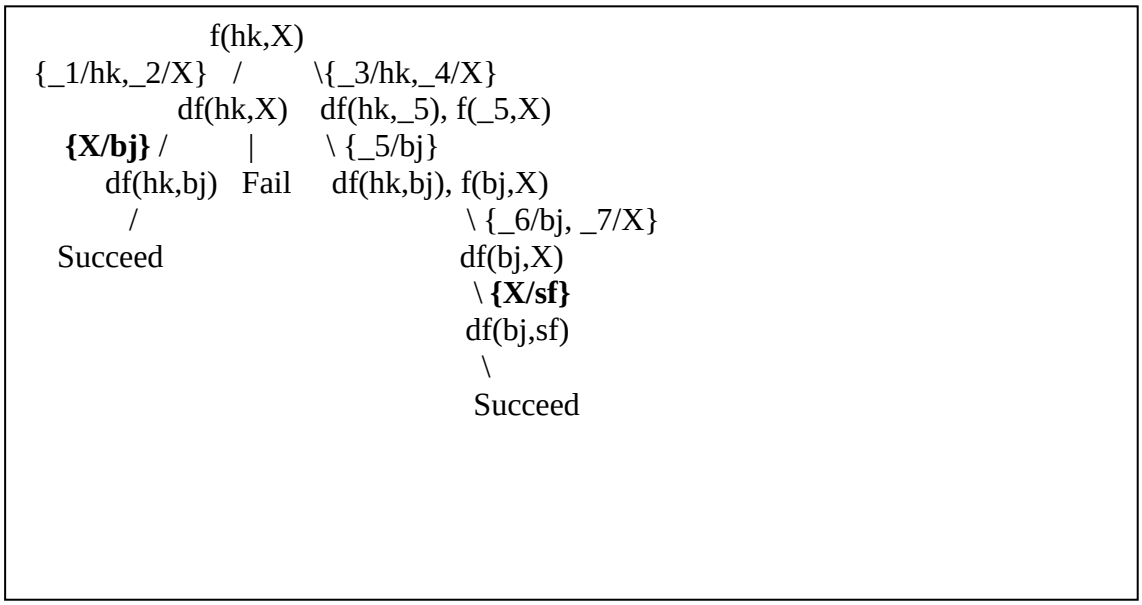
Problem 6 (10 points) Prolog Search Trees

Given the following Prolog program:

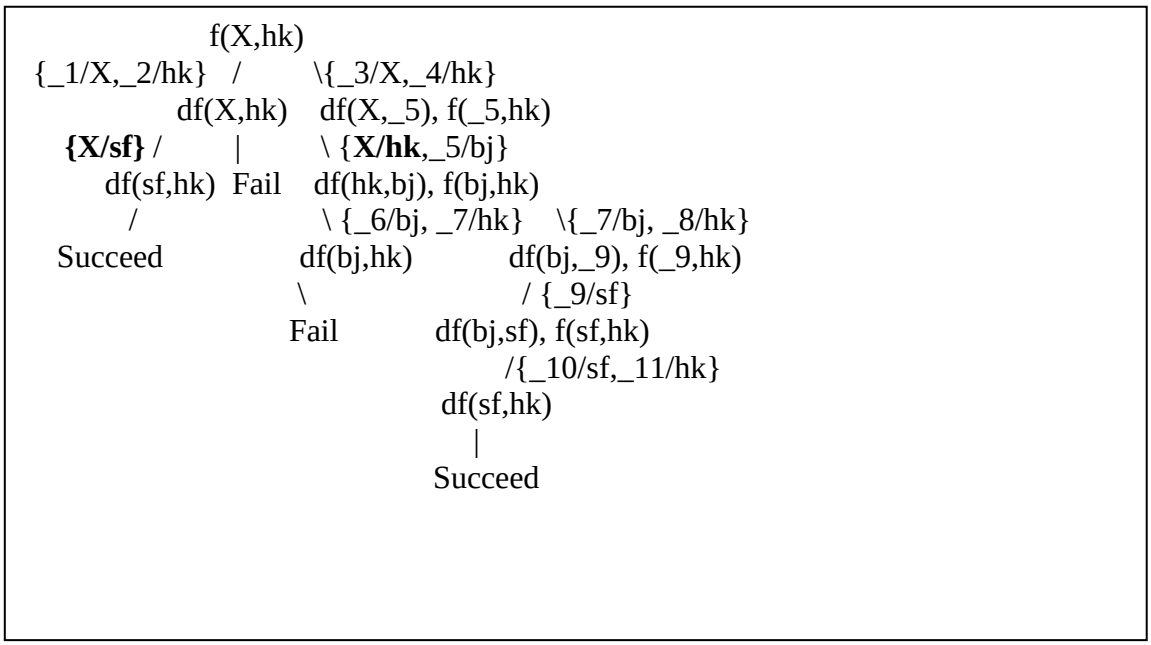
```
df(hk,bj).
df(bj,sf).
df(sf,hk).

f(X,Y):-df(X,Y).
f(X,Z):-df(X,Y),f(Y,Z).
```

Draw a Prolog search tree for the first **two** answers for each of the following queries:

(a) f(hk,X).



(b) f(X,hk).



Problem 7 (18 points) Parameter Passing Methods

Suppose an imaginary D language has syntax similar to the C language, but can apply a parameter passing method as specified. Determine the output of the following D program with each specified parameter passing method:

```
int coordx=0;
int coordy=0;
void Knight(int x,int y)
{
	x=x+1;
	y=y+2;
	printf("(%d,%d) ",coordx,coordy);
	coordx=coordx+2;
	coordy=coordy+1;
	printf("(%d,%d) ",coordx,coordy);
}
int main(int argc,char **argv)
{
	coordy++;
	Knight(coordx,coordy);
	printf("(%d,%d) ",coordx,coordy);
}
```

Output:

With Call-by-Value:

(0,1) (2,2) (2,2)

With Call-by-Reference:

(1,3) (3,4) (3,4)

With Call-by-Value-Result:

(0,1) (2,2) (1,3)

Problem 8 (15 points) Activation Records

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 throughout the method calls:

```
#include <stdio.h>
int x=0,y=0;
int square_sum(int x,int y)
{
  int sum= x*x+y*y;
  return sum;
}
int main(int argc,char **argv)
{
      int x=1, y=2;
      x=square_sum(x,y); //first call of square_sum
      y=square_sum(y,x);//second call of square_sum
}
```

Draw activation records at the following points in time:

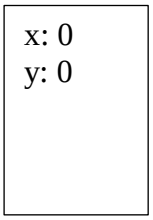
(i) right before calling main;

(ii) right before exiting the first call of square_sum;

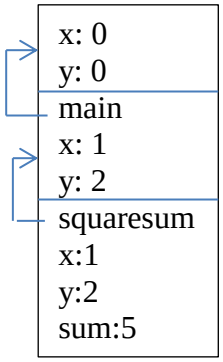
(iii) right before exiting the second call of square_sum;

(iv) right after exiting main.

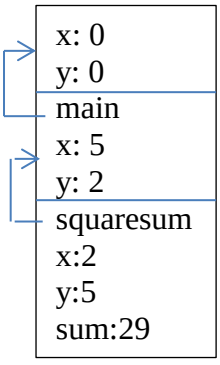
(i)



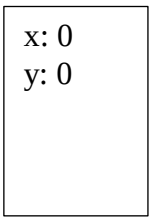
(ii)



(iii)



(iv)



(Additional blank paper for draft work)