Name, SURNAME and ID ⇒

Middle East Technical University

Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2014-2015

Final Exam

- **Duration: 120** minutes.
- **Total Points: 100**
- **Exam:**
  - This is a **closed book**, **closed notes** exam. The use of any reference material is strictly forbidden.
  - No attempts of cheating will be tolerated.
- **This exam consists of 10 pages including this page. Check that you have them all!**
- **GOOD LUCK !**

Question 1	
Question 2	
Question 3	
Question 4	
Question 5	
Question 6	
Total ⇒	

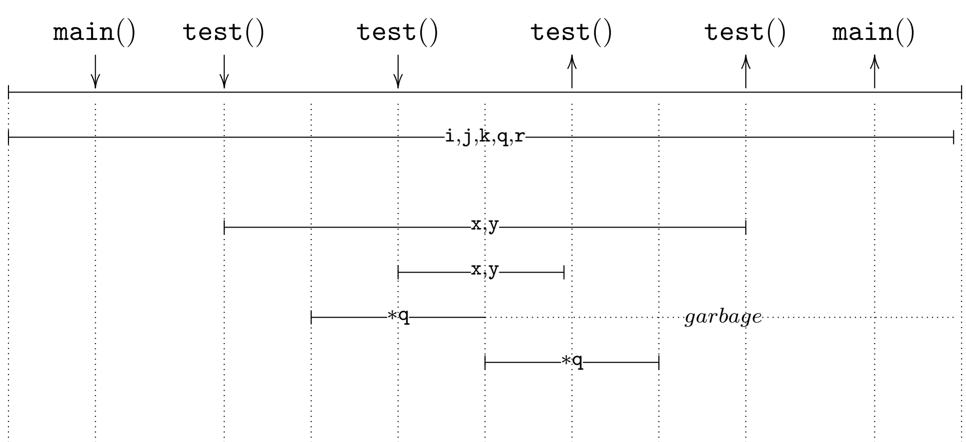
QUESTION 1. (15 points)

In the following C program (assume C features only) determine garbage variable(s) and dangling reference(s). Explain how and where they have occurred. You need to trace the execution of the program and keep track of the lifetimes and the contents of all the variables in order to discover garbage variables and dangling references. So, do the followings:

- Show the lifetimes of all variables on lifetime chart below (add necessary points to the chart for creating/destroying heap variables)
- Show how/if dangling reference and garbage variable occurs on lifetime chart below (such as reference time)
- Determine the output

```
#include "stdio.h"
#include "malloc.h"

int i=0, j=1, k=2;
int *q, *r;
void test(int x, int y)
{
      q=(int *)malloc(sizeof(int));
      *q=y;
      r=&k;
      x++;
      y++;
      (*q)++;
      (*r)++;
      if (x<2) {test(x,y); free (q);}
}
main()
{
      test(i,j);
      printf("%d %d %d %d %d\n",i,j,k,*q,*r);
}
```



0 1 4 *q 4
(*q is dangling reference)

QUESTION 2. (20 points)

a) (10 pts) Determine the output of the following program (written in a C like language) assuming static binding for the following parameter passing mechanisms:

a) normal order evaluation (call by name)
b) definitional mechanism, variable (call by reference)

```
int a[3]={10,20,30};
int i=1;
void test(int x, int y, int z)
{
      x++;  y--;  z++;
      printf("%d\n",z);
      x--;  y++;  z--;
      printf("%d\n",z);
      a[0]++;  a[1]--;  a[2]++;
      printf("%d %d %d \n",a[0],a[1],a[2]);
}
main()
{
      test(i,a[0],a[i]);
}
```

a) OUTPUT - by name

31

19

11 18 32

b) OUTPUT - by reference

21

20

11 19 31

b) (10pts) Determine the **output** of the following program (written in a C like language) assuming dynamic binding and call by value parameter passing technique is used. Determine **the environments** at the start time of each function. For each identifier specify where it is declared (such as `a->global int`, `a->main int`, etc.).

```
int i=5, j=5;
void g(int k)
{                    //E(g) = {i->main, j->f, k->g, f , g, main        }
         k=i+j+k;
         printf(   %d     ,k);
}
void f (int j)
{                    //E(f) = {i->main, j->f, f , g, main       }
         j=i+j;
         g(j);
         printf (   %d     ,j);
}

void main()
{                    // E(main) = {i->main, j->global, f , g, main       }
     int i=10; f(i); printf(   %d   , i);
}
```

OUTPUT

50

20

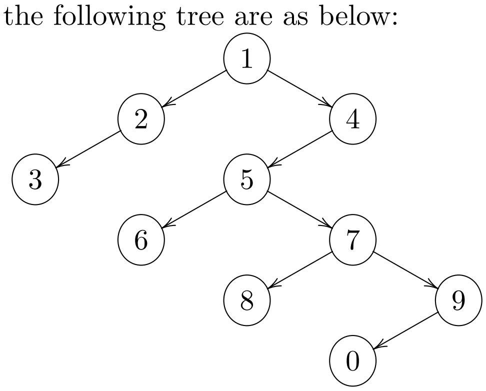
10

QUESTION 3. (20 points)

a) (10 points) Consider the following data type definition used for generating trees:

```
data TREE = EMPTY | NODE (Int, TREE, TREE)
```

Examples corresponding to the following tree are as below:




```
mytree = NODE (1, NODE (2, NODE (3, EMPTY, EMPTY),
                        EMPTY),
                  NODE (4, NODE (5, NODE (6, EMPTY, EMPTY),
                                    NODE (7, NODE (8, EMPTY, EMPTY),
                                             NODE (9, NODE (0, EMPTY, EMPTY),
                                                      EMPTY))),
                        EMPTY))
```

We are given 3 functions to generate the path from the given node to the root node in a tree. Assume that each node value is unique. Only one of them is correct. Determine the outputs for the following calls, and find out which function is correct.

```
path1 EMPTY     x  _   = [ ]
path1 (NODE (a,b,c)) x lst = if ( a==x ) then (a:lst )
      else if (path1 b x (a:lst) == [ ] ) then ( path1 c x (a:lst) )
                                          else  ( path1 b x (a:lst) )

path2 EMPTY     x  _   = [ ]
path2 (NODE (a,b,c)) x lst = if ( a==x ) then (a:lst )
      else if (path2 b x (lst) == [ ] ) then ( path2 c x (lst) )
                                        else  ( path2 b x (lst) )

path3 EMPTY     x  _   = [ ]
path3 (NODE (a,b,c)) x lst = if ( a==x ) then (lst )
      else if (path3 b x (a:lst) == [ ] ) then ( path3 c x (a:lst) )
                                          else  ( path3 b x (a:lst) )
```

```
Main> path1 mytree 9 []
```

[9, 7, 5, 4, 1]

```
Main> path2 mytree 9 []
```

[9]

```
Main> path3 mytree 9 []
```

[7, 5, 4, 1]

b) (10 pts) Consider the following Haskell definitions.

```
data X  = A | B Int Y
data Y  = C | D Int X
data Z  = E X | F Y deriving Show

instance Show X where
  show (A) = "A"
  show (B a1 a2) = "B"++(show a1)++":"++(show a2)++"B"
instance Show Y where
  show (C) = "C"
  show (D a1 a2) = "D"++(show a1)++"+"++(show a2)++"D"

x_gen 0 = (A)
x_gen n = (B n (y_gen (n-1)))

y_gen 0 = (C)
y_gen n = (D n (x_gen (n-1)))

class My_Class a where
  f::a->[Int]
  f x = []

instance My_Class X where
  f (A) = []
  f (B a1 a2) = a1:(f a2)
instance My_Class Y where
  f (C) = []
  f (D a1 a2) = a1:a1:(f a2)
instance My_Class Z
```

Determine the outputs of the following Haskell function calls.

```
Main> x_gen 5
```

B5:D4+B3:D2+B1:CBDBDB

```
Main> y_gen 5
```

D5+B4:D3+B2:D1:ADBDBD

```
Main> f (x_gen 5)
```

[5,4,4,3,2,2,1]

```
Main> f (y_gen 5)
```

[5,5,4,3,3,2,1,1]

```
Main> f (F (y_gen 5))
```

[]

QUESTION 4. (15 points)

Consider the following C++ program.

- Determine its output.
- Circle the expressions in the program corresponding to dynamic binding (late binding), and **show their bindings**.

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

class A{ public: int a;
             A():a(0){}
             A(int p):a(p){}
             virtual void operator+=(int p){a+=p; }
             virtual void incr(int p){a+=p; }
                     void incr2(int p){a+=2*p; }
};

class B: public A { public: int b;
             B():b(0),A(){}
             B(int p):b(p),A(2*p){}
             virtual void operator+=(int p){a+=p; b+=p; }
                     void incr(int p){a+=p; b+=p; }
                     void incr2(int p){a+=2*p; b+=2*p; }
};

class C: public B { public: int c;
             C():c(0),B(){}
             C(int p):c(p),B(2*p){}
             void operator+=(int p){a+=p; b+=p; c+=p; }
             void incr(int p){a+=p; b+=p; c+=p; }
};

void f1(A &a) { a+=10; }
void f2(A  a) { a+=10; }
void f3(B &b) { b+=10; }

main()
{
  A a1(10), *ap; B b1(20), *bp; C c1(30);
  cout<<a1.a<<"\n"<<b1.a<<" "<<b1.b<<"\n"<<c1.a<<" "<<c1.b<<" "<<c1.c<<"\n";

  ap=&b1; ap->incr(10); cout<<b1.a<<" "<<b1.b<<"\n";
  ap->incr2(10); cout<<b1.a<<" "<<b1.b<<"\n";

  f1(b1); cout<<b1.a<<" "<<b1.b<<"\n";
  f2(b1); cout<<b1.a<<" "<<b1.b<<"\n";

  bp=&c1; bp->incr(10); cout<<c1.a<<" "<<c1.b<<" "<<c1.c<<"\n";
  bp->incr2(10); cout<<c1.a<<" "<<c1.b<<" "<<c1.c<<"\n";

  f3(c1); cout<<c1.a<<" "<<c1.b<<" "<<c1.c<<"\n";
}
```

OUTPUT

```
10

40  20

120  60  30
```

50 ______ 30 ______

70   30 ______

80 ______ 40 ______

80 ______ 40

130 ______ 70 ______ 40 ______

150 ______ 90 ______ 40 ______

160 ______ 100 ______ 50 ______

QUESTION 5. (10 points)

Assume the following Prolog program is given:

```
pm([A], [A]).
/* [A,B|C] = [A|[B|C]]  list has at least two elements , A and B*/
pm([A, B | C ], [A|TR]) :-  pm([B|C],TR).
pm([A, B | C ], TRA) :-  pm([B|C],TR), append(TR, [A], TRA).

qA(s(A,B),s(B,A)).

qB(s(A,B),s(B,_)).

qC([X,Y|R], [Y,X|R]).

qD([1,X-1,X], [X+1|_]).

qE([X,X|R], R).
```

Give **all** answers found by Prolog for the following queries. If no solution found, write 'no':

Query	Results
pm([a,b],R)	R = [a,b] R = [b,a]
pm([a,b,c,d],R)	R=[a,b,c,d] R=[b,c,d,a] R=[a,c,d,b] R=[c,d,b,a] R=[a,b,d,c] R=[b,d,c,a] R=[a,d,c,b] R=[d,c,b,a]
qA(s(a,X),s(b,Y))	X=b, Y=a
qA(s(a,X),s(b,X))	no
qB(X,s(c,d))	X=s(_,c)
qC([1,2,3,4],R))	R=[2,1,3,4]
qD([1,2,X],R))	no
qE([Y,2,a],R))	R=[a], Y=2

QUESTION 6. (20 points)

You are asked to design a grammar for expressions of a language called Pi. Pi expressions support the following operators:

- $\parallel$ binary operator (concurrent evaluate)
- $\gg$ binary operator (dependent evaluate)
- $\vee$ binary operator (concurrent disjunction)
- $\triangleright$ postfix unary operator (output)
- $\triangleleft$ postfix unary operator (input)
- `(...)` paranthesis for grouping expressions

Other non-terminals of the language is letters p, t, w, x, y, z which give the basic expression. The precedence of the operators are: `(...)` has highest precedence, then $\triangleright$ and $\triangleleft$ are in the same level, then $\vee$, then $\gg$, and the lowest precedence operator is the $\parallel$. $\parallel$ is right associative, all other binary operators are left associative.
For example the expression $x \triangleright \parallel (y \gg z) \triangleright \vee w \vee t \parallel p \triangleleft$ is equivalent to:
$(x\triangleright) \parallel ((((y \gg z)\triangleright) \vee w) \vee t \parallel (p\triangleleft))$

a) Write and un-ambigous grammar that accept the sentences of this language

b) Draw the parse tree of the expression (not graded if your answer above is completely wrong):
$x \parallel y \gg w \gg (z \parallel p\triangleleft) \parallel p$

a) $\langle$**expr**$\rangle$ ::- $\langle$**depend**$\rangle$ | $\langle$**depend**$\rangle$ $\parallel$ $\langle$**expr**$\rangle$
$\langle$**depend**$\rangle$ ::- $\langle$**disj**$\rangle$ | $\langle$**depend**$\rangle$ $\gg$ $\langle$**disj**$\rangle$
$\langle$**disj**$\rangle$ ::- $\langle$**post**$\rangle$ | $\langle$**disj**$\rangle$ $\vee$ $\langle$**post**$\rangle$
$\langle$**post**$\rangle$ ::- $\langle$**id**$\rangle$ | $\langle$**post**$\rangle$ $\triangleleft$ | $\langle$**post**$\rangle$ $\triangleright$
$\langle$**id**$\rangle$ ::- p | t | w | x | y | z | ($\langle$**expr**$\rangle$)
b)