Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Tot

CEng 242 - Programming Language Concepts

Spring 2015-2016, Final, Closed book(10 pages, 9 questions, 102 points, 150 minutes)

Name: ______________________________ **No:** __________

QUESTION 1.(12 points)

Determine the output of the following C++ program. Assume that all necessary headers and namespaces are included and all compiler optimizations are disabled.

```
class A {
    int x;
    public:
        A(int p) { x = 2*p; }
        A(const A& p) { x = 2*p.x; }
        A& operator=(const A& p) { x = 4*p.x; }
        ~A() { x = x/2; }
        int getx() const { return x; }
};

A t(2);

A f() {
    A t(2);
    return t;
}

A& h() {
    return t;
}

void g(const A &p) {
    cout << p.getx() << endl;
}

void q() {
    A a1 = A(2);
    A a2 = a1;
    a1 = a2;
    cout << a1.getx() << endl;
}

int main () {
    cout << "First output: "; g(f());
    cout << "Second output: "; g(h());
    cout << "Third output: "; q();
    return 0;
}
```

First output: 8 Second output: 4 Third output: 64

QUESTION 2. (10 points)

You are asked to implement a stock management program using C++ and object-oriented programming. The requirements are as follows:

- You must have an **abstract base class** that defines the behavior of a stock manager. Give it the name `StockManager`. This abstract base class must contain two **pure virtual** member functions, called `buy` and `sell` both of which take a **constant reference to an object** representing the historical information about the stock trades. Assume this information is of type `StockHistory`. They return an **integer** representing how many stocks to buy or sell. Complete the function prototype for the `buy` function only (assume that you are declaring this function inside the class scope):
  - [int] `buy(`[const StockHistory&]`)` [= 0];

- This base class must also contain a **protected** member variable called `stockCount` that represents how many stocks are currently owned by us (an integer value) as well as **public accessor** and **mutator** member functions to get and set the value of this variable. Add this member variable and the related functions to this class using the correct access rights:

```
class StockManager {

protected:
        int stockCount;
public:
        int accessor() const { return stockCount;}
        void mutator(int v) { stockCount = v;}

};
```

- Assume that two new classes called `AggressiveStockManager` and `ConservativeStockManager` are derived from the `StockManager` class, both of which implement its the pure virtual functions. Answer the following questions as true (T) or false (F):
  - [T] Both classes can access the `stockCount` variable of their base class.
  - [T] We can safely assign an `AggressiveStockManager` object to a `StockManager` reference.
  - [F] We can safely assign an `AggressiveStockManager` object to a `ConservativeStockManager` reference.
  - [F] We can safely assign a `StockManager` object to an `AggressiveStockManager` reference.
  - [T] We cannot create instances of the `StockManager` class.

QUESTION 3.(15 points)

A new PL called METUPL is being designed and you are expected to write a **preprocessor** and **parser** for this language using Haskell. The **preprocessor** takes a SourceCode as input and produces a list of Tokens. The SourceCode and Token are defined for you as:

```
type SourceCode = String
type Token = String
```

a) Declare the type signature and implement a preprocess function which extracts tokens from the given source code and returns them a list. Note that the tokens are separated from each other only by whitespace characters but there could be multiple whitespaces between each token. For example, preprocess "   void main () " should return ["void", "main", "()"].

Make the type declaration in this box:

```
preprocess :: Sourcode -> [Token]
```

Implement the preprocess function in this box. Do not use any built-in functions (of course, you can use operators such as ++, : for list processing). If necessary implement your helper functions here or on the back of this page.

```
preprocess source = preprocess' [] source where
    preprocess' [] [] = []
    preprocess' tok [] =[tok]
    preprocess' [] (a:rest)  | a == ' ' =  preprocess' [] rest
                             | otherwise = preprocess' [a] rest
    preprocess' tok (a:rest) | a == ' ' =
           tok : preprocess' [] rest
                             | otherwise =
           preprocess' (tok ++ [a])  rest
```

b) For the **parser**, you are expected to declare a Parser typeclass. This typeclass will contain a single function called parse. This function will take two parameters with the first parameter being an **instance** of this typeclass and the second one a list of Tokens. It should return a value of ParseTree data type, whose details are given below.

- Show the definition of your type class. It must contain the type signature of the parse function as well:

```
class Parser a where
      parse ::  a -> [Token]
```

- Show the definition of the data type ParseTree. It is a possibly empty N-ary tree with Tokens represented only in the leaf nodes. You are free to choose the names of your tags.

```
data ParseTree = Node [ParseTree] | Leaf Token
```

QUESTION 4.(10 points)

Determine the output of the following C++ program.

```
class A {
public:
virtual void f() {cout<<"A::f\n";}
        void g() {f();}
virtual void h() {cout<<"A::h\n";}
virtual void i() {cout<<"A::i\n";}
};

class B:public A{
public:
void f() {cout<<"B::f\n";}
void k() {cout<<"B::k\n";}
void i() {cout<<"B::i\n";}
void j() {f();}
};

class C: public B{
public:
void f(){cout<<"C::f\n";}
void k(){cout<<"C::k\n";}
void h(){cout<<"C::h\n";}
};

void test1(A *ta) {ta->g();}
void test2(A &pa) {pa.h();}
void test3(A &pa) {pa.i();}

void test4(B *tb) {tb->j();}
void test5(B *tb) {tb->f();}
void test6(B *tb) {tb->k();}

int main(){
A a; B b; C c;

test1(&a);
test1(&b);
test1(&c);
cout<<"*****\n";
test2(c);
test3(c);
cout<<"*****\n";
test4(&c);
test5(&c);
test6(&c);
}
```

OUTPUT:

A::f
B::f
C::f
****
C ::h
B ::i
****
C::f
C::f
B ::k

QUESTION 5.(10 points)

Trace the execution of the following C program and determine:

- garbage variables (GV) and dangling references (DR) (circle the statement and write as GV and DR)
- the output (fill into the table)

```
int a[2]={10,20};
int *p, *q;
int main()
{
     p=(int *)malloc(sizeof(int));
     q=a;
     *p=30;
     printf("%d %d\n",*p,*q);
     q++;
     (*q)++;
     printf("%d %d\n",*p,*q);
     p=q;        GB p
     *(a+2)=*q;  DR
     printf("%d %d\n",*p,*q);
     q=(int *)malloc(sizeof(int));
     *q=*p;
     free (q);
     (*q)++;     DR
     printf("%d %d\n",*p,*q);
}
```

OUTPUT:

30	10
30	21
21	21
21	?DR

QUESTION 6.(10 points)

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

a. lazy evaluation

b. normal order evaluation (call by name)

c. definitional mechanism (call by reference)

```
int a[4]={10,20,30,40};
int i=1;

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

a. OUTPUT - lazy

2	9	31		
3	8	32		
3	8	20	32	40

b. OUTPUT - name

2	9	31		
3	8	41		
3	8	20	31	41

c. OUTPUT - reference

2	9	21		
3	8	22		
3	8	22	30	40

QUESTION 7.(10 points)

Determine the output of the following C++ program (some of the output is given, just determine the missing lines).

```
int i1=1, i2=2, i3=3, i4=4;

class A {
public:
       int i;
       A(int i){cout<<"A::A(int)\n"; this->i=i;}
       A(const A &a){cout<<"A::A(A)\n"; i=a.i;}
       void operator>(int &i) {cout<<"op>#1\n"; i=this->i;}
       friend void operator>(int &i, A &a) {cout<<"op>#2\n"; a.i=i;}
       friend void operator<(int &i, A &a) {cout<<"op<#1\n"; i=a.i;}
       void operator<(int &i) {cout<<"op<#2\n"; this->i=i;}
       void operator=(A &a){cout<<"A::operator=(A)\n"; a.i=i;}
};

class B:public A {
public:
       A a;
       B(int i):A(i),a(i+1){cout<<"B::B(int)\n";};
};

void f(A a1, A &a2, A *a3, A a4) {
      cout<<"f starts\n";
      a1<i1;
      i2>a2;
      (*a3)>i3;
      i4<a4;
      cout<<"f ends\n";
}

int main() {
       A a10(10), a15(15);
       B b5(5), b10=b5;

       cout<<"declarations ends\n",
       f(5, a10, &a15, b5);
       cout<<a10.i<<" "<<a15.i<<" "<<b5.i<<":"<<b5.a.i<<"\n";
       cout<<i1<<" "<<i2<<" "<<i3<<" "<<i4<<"\n";
       cout<<"assignment\n";
       b10=b5;
       cout<<b5.i<<":"<<b5.a.i<<" "<<b10.i<<":"<<b10.a.i<<"\n";
}
```

```
A::A(int)
A::A(int)
```

$\boxed{\text{A::A(int)}}$

$\boxed{\text{A::A(int)}}$

```
B::B(int)
```

$\boxed{\text{A::A(A)}}$

$\boxed{\text{A::A(A)}}$

```
declarations ends
```

$\boxed{\text{A::A(A)}}$

$\boxed{\text{A::A(int)}}$

```
f starts
```

$\boxed{\text{op}<\#2}$

$\boxed{\text{op}>\#2}$

$\boxed{\text{op}>\#1}$

$\boxed{\text{op}<\#1}$

```
f ends
```

$\boxed{\text{2 15 5:6}}$

$\boxed{\text{1 2 15 5}}$

```
assignment
```

$\boxed{\text{A::operator=(A)}}$

$\boxed{\text{A::operator=(A)}}$

$\boxed{\text{5:6 5:6}}$

QUESTION 8. (15 points)

a) Assume `split/3` clause divides a list into two equal size list. Elements are distributed to first and second list on alternating order. For example `split([a,b,c,d,e,f], X, Y).` gives `X=[a,c,e], Y=[b,d,f].` When list has odd number of elements, first list wil get the extra element as `split([a,b,c,d,e], [a,c,e], [b,d]).` Complete the `split/3` as defined above:

```
split([],[],[]).          /* empty list */
split([H],[H],[]).        /* last element */
split([A|[B|R]], [A|RA] , [B|RB] ) :- split( R, RA, RB)
```

b) Assume `merge/3` clause merges two sorted lists in ascending order into a sorted list containing elements from the both. For example `merge([1,2,4,5,8],[3,7,8], R)` gives `R = [1,2,3,4,5,7,8,8].`

```
merge([],A,A).
merge(A,[],A) :- A=[_|_].  /* make A non-empty to eliminate ambiguity */
merge([A|ARest], [B|BRest],  [A|Result]) :- A =< B,
        merge(ARest, [B|BRest], Result), .
merge([A|ARest], [B|BRest],  [B|Result]) :- A > B ,
        merge([A|ARest], BRest, Result), .
```

c) Write all answers of query `traverse(2,1, L).` for the following Prolog program.

```
right(r).
right(e).
down(d).

traverse(3,3,[]).
traverse(X,Y,[OP|L]) :-  NX is X+1, NX =< 3, right(OP), traverse(NX,Y ,L).
traverse(X,Y,[OP|L]) :-  NY is Y+1, NY =< 3, down(OP), traverse(X ,NY,L).
```

```
L = [r,d,d]
L = [e,d,d]
L = [d,r,d]
L = [d,e,d]
L = [d,d,r]
L = [d,d,e]
```

QUESTION 9. (10 points)

Assume you are asked to define the syntax for a hypothetical page typesetting language.
Language contains the following operators:

1. The terminals of the language are capital letters. All letters from `A` to `Z` are literals describing a page id.
2. Expressions can be put in paranthesis `()` for grouping.
3. `>>`, `<<`, and `^` are unary prefix operators and describe page alignment.
4. `!` and `-` are right associative binary operators indicating current page is divided into two columns and rows respectively.
5. `\\` is a left associative binary operator indicating the page skip.
6. The precedence of operators are:
   highest is `()`, then unary alignment operators in same level, then `!` and `-` in same level, then `\\` has the lowest precedence.

a) Write an **unambigous** grammar respecting precende and associativity rules for this language. Use descriptive non-terminal names as `<aligned>` etc. Assume starting non-terminal is `<page>`.

<page> → <page> \\ <sub> | <sub>

<sub > → < align > ! <sub> | <align>

< align > → << <align> | >> <align> | ^ <align> | <simple>

<simple> → (<page>) | A | B | . . . | Z

b) Draw syntax tree of the expression '`>>(A\\B!^C-D\\E)!F`'.