No: _____



Name: ___

}

First output: 8

Q1	$\mathbf{Q2}$	$\mathbf{Q3}$	$\mathbf{Q4}$	$\mathbf{Q5}$	$\mathbf{Q6}$	Q7	$\mathbf{Q8}$	$\mathbf{Q}9$	Tot

CEng 242 - Programming Language Concepts Spring 2015-2016, Final, Closed book(10 pages, 9 questions, 102 points, 150 minutes)

QUESTION 1.(12 points)
Determine the output of the following C++ program. Assume that all necessary headers and namespace are included and all compiler optimizations are disabled.
<pre>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; } };</pre>
J ,
A t(2);
<pre>A f() { A t(2); return t; }</pre>
A& h() { return t; }
<pre>void g(const A &p) { cout << p.getx() << endl; }</pre>
<pre>void q() { A a1 = A(2); A a2 = a1; a1 = a2; cout << a1.getx() << endl; }</pre>
<pre>int main () { cout << "First output: "; g(f()); cout << "Second output: "; g(h()); cout << "Third output: "; q(); return 0;</pre>

Third output: 64

Second output: 4



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 <u>list</u> 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.

- 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 <u>list</u> 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
```

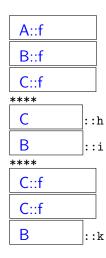


QUESTION 4.(10 points)

Determine the output of the following C++ program.

```
class A {
public:
virtual void f() {cout << "A::f\n";}</pre>
         void g() {f();}
virtual void h() {cout << "A::h\n";}</pre>
virtual void i() {cout << "A::i\n";}</pre>
class B:public A{
public:
void f() {cout << "B::f\n";}</pre>
void k() \{cout << "B::k\n";\}
void i() {cout << "B::i\n";}</pre>
void j() {f();}
};
class C: public B{
public:
void f(){cout << "C::f\n";}</pre>
void k(){cout << "C::k\n";}</pre>
void h(){cout << "C::h\n";}</pre>
void test1(A *ta) {ta->g();}
void test2(A &pa) {pa.h();}
void test3(A &pa) {pa.i();}
void test4(B *tb) \{tb \rightarrow j();\}
void test5(B *tb) \{tb \rightarrow f();\}
void test6(B *tb) \{tb->k();\}
int main(){
Aa; Bb; Cc;
test1(&a);
test1(&b);
test1(&c);
cout <<"****\n";
test2(c);
test3(c);
cout <<"****\n";
test4(&c);
test5(&c);
test6(&c);
}
```

OUTPUT:





QUESTION 5.(10 points)

Trace the execution of the following C program and determine:

- ullet 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);
                DR
    (*q)++;
    printf("%d %d\n",*p,*q);
}
```

OUTPUT:

30	10
30	21
21	21
21	?DR



QUESTION 6.(10 points)

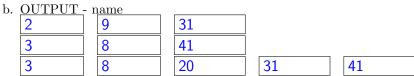
Determine the output of the following program (written in a C like language) assuming static binding for the following parameter passing mechanisms:

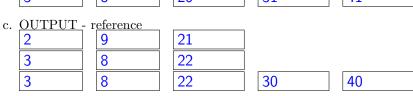
- a. lazy evaluation
- b. normal order evaluation (call by name)
- c. definitional mechanism (call by reference)

```
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







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) \setminus n"; this -> i = i; \}
      A(const A \&a)\{cout << "A:: A(A) \setminus 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;}</pre>
};
class B:public A {
public:
       B(int i):A(i),a(i+1){cout << "B::B(int) \setminus n";};
};
void f(A a1, A &a2, A *a3, A a4) {
     cout << "f starts \n";</pre>
     a1<i1;
     i2 > a2;
      (*a3)>i3;
     i4 < a4;
     cout << "f ends \n";</pre>
}
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";</pre>
       b10 = b5;
       cout << b5.i <<":"<< b5.a.i <<" "<< b10.i <<":"<< b10.a.i << "\n";
}
```





A::A(int)
A::A(int)
A::A(int)
A::A(int)
B::B(int)
A::A(A)
A::A(A)
declarations ends
A::A(A)
A::A(int)
f starts
op<#2
op>#2
op>#1
op<#1
f ends
2 15 5:6
1 2 15 5
assignment
A::operator=(A)
A::operator=(A)

5:6 5:6



QUESTION 8.(15 points)

right(r).

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:

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].

```
\label{eq:merge} \begin{split} &\text{merge}([],A,A)\,.\\ &\text{merge}(A,[],A)\,:-\,A=[\_|\_]\,.\quad /*\ \textit{make}\ \textit{A}\ \textit{non-empty}\ \textit{to}\ \textit{eliminate}\ \textit{ambiguity}\ */\\ &\text{merge}([A|ARest],\ [B|BRest],\ [A|Result])\,:-\,\boxed{A=<B},\\ &\boxed{\text{merge}(ARest,\ [B|BRest],\ Result)},\\ &\text{merge}([A|ARest],\ [B|BRest],\ [B|Result])\,:-\,\boxed{A>B}\,,\\ &\boxed{\text{merge}([A|ARest],\ BRest,\ Result)},\\ &\boxed{\text{merge}([A|ARes
```

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

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

traverse(3,3,[]).
traverse(X,Y,[0P|L]) :- NX is X+1, NX =< 3, right(0P), traverse(NX,Y ,L).
traverse(X,Y,[0P|L]) :- NY is Y+1, NY =< 3, down(0P), traverse(X ,NY,L).</pre>
```

```
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 <u>right associative</u> 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>.

$$\rightarrow$$
 \\ <sub>| <sub><sub>\rightarrow ! <sub>|
 \rightarrow << | >> | ^ |
 \rightarrow () | A | B | ... | Z</sub></sub></sub></sub>

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

