

Question 1: (35 points)

```
class Customer {  
    public:  
    ...  
    void remove();  
    ...  
    private:  
        Account* accounts;  
        int numAccounts;  
};
```

```
class Account {  
    public:  
        int accountNum;  
        double balance;  
};
```

Consider the following `Customer` and `Account` classes. Each customer object stores a dynamically allocated array of accounts (pointed by `accounts`) and the number of accounts in this array (`numAccounts`). Each account is associated with a unique identifier (`accountNum`) and a `balance`.

Write down the implementation of the `remove` member function of the `Customer` class. The function removes the accounts with a zero balance.

Important Notes:

- This implementation must use only the necessary amount of memory to store the accounts of the customer where the number of accounts is given in the variable `numAccounts`. For example, if the customer has 5 accounts, the memory dynamically used by the customer corresponds to only these 5 accounts.
- You should avoid unnecessary variable declarations and copies.
- There should be no memory leaks in your solution.
- You cannot call member functions in your implementation. You cannot alter the class definitions either.

```
void Customer::remove() {  
  
    int newSize = 0;  
  
    for(int i = 0; i < numAccounts; ++i )  
        if ( accounts[i].balance != 0 ) newSize++;  
  
    if (newSize == 0){  
        numAccounts=0;  
        if(accounts) delete [] accounts;  
        accounts = NULL;  
        return;  
    }  
  
    else{  
  
        Account* newArray = new Account[newSize];  
        int j = 0;  
        for ( i = 0; i < numAccounts; i++ )  
            if ( accounts[i].balance != 0 )  
                newArray[j++] = accounts[i];  
  
        delete[] accounts;  
        accounts = newArray;  
        numAccounts = newSize;  
    }  
  
}
```

Question 2: (30 points) Consider the following `Foo` class implementation. What is the output of the program given below?

```
class Foo {
public:
    Foo( int no = 0 ) {
        id = no;
        cout << "Constructor " << id << endl;
    }
    Foo( const Foo& source ) {
        id = source.id;
        cout << "Copy constructor " << id <<
endl;
    }
    ~Foo() {
        cout << "Destructor " << id << endl;
    }
    void operator=( const Foo& right ) {
        id = right.id;
        cout << "Assignment operator " << id
<< endl;
    }
    void setId( const int no ) {
        id = no;
    }
private:
    int id;
};
```

```
void bar( Foo w[], Foo& y, Foo* z, Foo x ) {
    Foo* p, o( 7 );
    cout << "Inside bar..." << endl;
    y = *z;
    z->setId( 2 );
    w[0].setId( 4 );
    p = w;
    x = *p;
    delete [] p;
    cout << "End of bar..." << endl;
}


int main() {
    Foo* p = new Foo[2], *q = p; Foo o( 5 );
    q[1].setId( 1 );
    cout << "Before bar..." << endl;
    bar( p, o, q + 1, *p );
    cout << "End of main..." << endl;
    return 0;
}
```

```
Constructor 0
Constructor 0
Constructor 5
Before bar...
Copy constructor 0
Constructor 7
Inside bar...
Assignment operator 1
Assignment operator 4
Destructor 2
Destructor 4
End of bar...
Destructor 7
Destructor 4
End of main...
Destructor 1
```

Question 3: (35 points) Consider the global `deleteFirstColumn` function whose prototype is given below. This function takes a 2D array (`arr`), its dimensions (where `M` is the first dimension and `N` is the second dimension). It deletes the first column from this 2D array such that it becomes an `M x (N-1)` array (with `M` being the first dimension and `(N-1)` being the second dimension). This function also modifies the value of the second dimension. You may assume that `M > 0` and `N > 1`.

For example, consider the following 2D array where the `M = 3` and `N = 4`, respectively. After calling the function with 2D array as argument, the array should become as shown on the right below and the second dimension should become 3.

5	6	7	8
9	10	11	12
13	14	15	16



6	7	8
10	11	12
14	15	16

Note that your function should not have any memory leaks. You will lose points if you make more dynamic memory allocations than necessary. You are not allowed to change the prototype below.

```
void deleteFirstColumn( int** arr, int M, int& N ) {  
  
void deleteFirstColumn( int** arr, const int M, int& N ) {  
  
    for ( int i = 0; i < M; i++ ) {  
  
        int* oldRow = arr[i];  
  
        arr[i] = new int[N-1];  
  
        for ( int j = 1; j < N; j++ )  
  
            arr[i][j-1] = oldRow[j];  
  
        delete [] oldRow;  
  
    }  
  
    N--;  
  
}
```