ENSF 337 – Fall 2018 Tutorial 11

Problem I - Consider the definition of the following class, and answer the following questions:

```
typedef double Type;
class Matrix{
public:
   Matrix(int r, int c);
   ~Matrix() { destroy(); }
    Matrix(const Matrix& src) {copy (src); }
    Matrix& operator= (const Matrix& rhs);
    // assume more function, as needed
private:
   Type** storageM;
   int rows;
   int columns;
};
```

```
Matrix::Matrix(int r, int c): columns(c),rows(r){
   storageM = new Type* [r];
    assert(storageM != nullptr);
   for(int i=0; i<r; i++){
       storageM[i] = new Type[c];
       for(int j=0; j < columns; j++)
              storageM[i][j] = 0;
   }
   // point one
}
int main(void){
   Matrix a(3, 4);
   ...
   return 0;
}
```

Question a: Draw a memory diagram for point one.
Question b: Write the definition of assignment operator.

Problem II – Consider the following program and draw a memory diagram when the value of R is 3 for the second time;

```
int f (int n){
     int R;
     if (n == 1 || n == 2)
        R = 1;
     else
        R = f(n-1) + f(n-2);
     return R;
}
```

```
int main(){
    int x = 6;
    int R = f(x);
    return 0
}
```

Problem III:

1. Draw the memory diagram for point one in function `func`.
2. What is the program output?

```
void func(double **m){
    cout << "\n  *m[0] =  " <<  *m[0];
    cout <<"\n  *m[1] = " << *m[1];
    cout << "\n  (*m)[1] = " << (*m)[1];
    cout <<"\n  **m = "<< **m;
    cout <<"\n  **(m+1)) =  "<< **(m+1);
    cout <<"\n  m[1][1] = " << m[1][1];
    // POINT ONE
}
```

```
int main() {
  int row = 3;
  int col = 4;
  double* p[4];
  double x[]={100, 340.3, 55.6, 103, 134.5, 155.6, 203,
                      234.5, 255.6, 303, 334.5, 355.6};
  for(int i = 0, j = 0; i<row ; j+= col, i++)
        p[i] = x + j;
  func(p);
  return 0;
}
```

Problem IV:

Consider the definitions of struct type called `Node`, and class `List`. Then write the implementation of a recursive destroy function that delete allocated nodes.

```
struct Node {
 int item;
 Node *next;
};
```

```
class List {
public:
 List(); // PROMISES: Creates empty list.
~List();
 void insert(const int& the_item);
private:
 Node *headM;
 void copy(const List& src);
 void destroy();
};
```