Assignment 3 –Dynamic Arrays and Classes

Required Reading

Problem Solving with C++

Chapter 7 – Arrays
Chapter 9 – Pointers and Dynamic Arrays
Chapter 10 – Classes
Chapter 11 – Classes and Arrays

Optional Reading

Chapter 13 – Pointers and Linked Lists

Instructions – PLEASE READ *(notice **bold** and underlined phrases)*

This assignment has four parts:

A. Written – Answer the questions, **submit to CourSys**
B. Programming – Code must be well commented, compile/test, then submit
C. Bonus – Optional Programming Assignment for Bonus Marks
D. Submission – Submit specified assignment files by deadline

1. **This is an individual assignment. You may *NOT* work in teams for this assignment. You may discuss ideas with others, but you must answer all questions and write all the code yourself. Plagiarism by students will result penalties that may include receiving zero for the assignment.**

2. **All Files Submitted for Programming Assignments must include block comments for the file, each class, and each function or method. Block comments at the top of each file must include the name of the file plus a short description of what the file does, and the Student's name, Number, and school email address.**

3. **Submission deadline: 11:59pm Tuesday Mar 29th. You should be able to complete the work if you have completed the required reading for the assignment. More material is available in the class Slides. While you have seen most of what is discussed here, some topics discussed in this assignment and needed for your submission may not be seen until another class! You may submit before the deadline if you prefer. You may resubmit again later without penalty provided you resubmit before the deadline.**

4. **Late penalties are based on the CourSys timestamp of the last file submitted by the student. Late penalties apply to the entire assignment, even if only a single file was submitted late.**

A. Written Assignment – Submit in CourSys

Students may discuss the problems with fellow students, but <u>MUST</u> complete the work individually. Submitted answers must be your own work.

This section does not require any programming. Write your answers neatly in a document such as a Word Document that will be submitted to CourSys as part of the assignment.

Your document must be called a3Writeup.doc or a3Writeup.pdf and must include your name, student number, and email address. Use Courier font and double-spacing in your write up. Image files of handwritten work is not acceptable unless otherwise specified.

1. Chapter 7 – Arrays 5 Marks

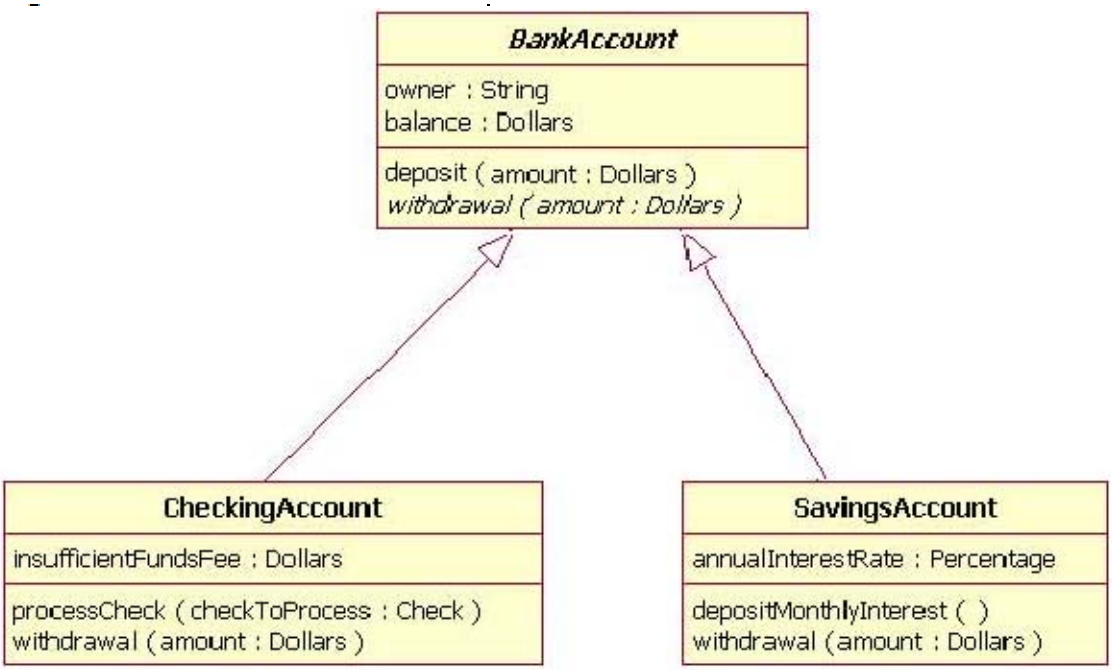
(a) What does the term Stride refer to when discussing arrays?
(b) Given an array of pointers like argv, on a 64-bit machine, how does one compute the address of argv[i] if the argv array starts at address X?
(c) When passing an array as a parameter to a function, can that function make changes to the contents of the array?
(d) When passing an array as a parameter, how do we ensure that the function cannot change the array elements?
(e) Given a two dimensional int Array A[4][6], give the formula to find the address in memory of A[row][col], assume the stride of an int is 4 and the starting address of A is address 1000.

2. Chapter 9 – Pointers and Dynamic Arrays 5 Marks

(a) Use typedef to define a new type called DoubleArrayPtr that points to an array of doubles.
(b) Define a variable of type DoubleArrayPtr called p and initialize p to point to an array of 10 doubles using the new operator.
(c) When allocating memory for array p above, what is the name of the memory that the NEW operator allocates memory from?
(d) When one is finished using a dynamic array, one must return the memory so it can be re-used. Write the code required to free the memory allocated to array p above.
(e) Given the two dimensional dynamic array m presented in class, give the formula to find the address in memory of A[row][col], assume the stride of an int is 4 and the starting address of A is address 1000.

3. Chapter 10.4 - Introduction to Inheritance 5 Marks

Review the UML Diagram below describing the BankAccount Hierarchy. Assume that all methods are public and all member variables are private. Assume that the String class is equivalent to string, and class Dollars is the same as class Money that was presented in class.



(a) Write up the class declaration for the BankAccount class. The class declaration should include the listed public member functions and private member variables. You do not need to implement the constructor or methods, just declare them in the class declaration.

(b) Write up the class declaration for the CheckingAccount class. Ensure that the class declaration captures any parent/child relationship.

(c) Write up the class declaration for the SavingsAccount class. Ensure that the class declaration captures any parent/child relationship.

(d) Which classes are base classes and which are derived classes?

(e) If MySavings is instantiated from the SavingsAccount class, then which class' method gets invoked by MySavings.withdrawal($10) and which class's method gets invoked by MySavings.deposit($10)?

B. Programming – To be completed by Students Individually

You must *not* use the vector class for this assignment.

1. Sorting Arrays 10 Marks

Write a void function called bubbleSort that accepts an array `a` and the number of int elements in that array `num`. This function should sort the elements such that:

$$a[0] \leq a[1] \leq \ldots \leq a[num-1]$$

Your function should be submitted in a file called bubbleSort.cpp. Your file must #include header file bubbleSort.h to declare the function prototype for bubleSort as shown below. This file is available on the Assignment page in CourSys.

Test your function yourself to perform unit testing with your own test program. Then perform integration testing by testing your bubbleSory.cpp file with the test program bubbleSortTest.cpp that will also be provided in CourSys.

Your function must use the following signature defined in bubbleSort.h:

```
void bubbleSort(int a[], int num);
```

Test Example:

Given array A[5] where:

```
A = {5,4,3,2,1}
```

After calling:

```
bubbleSort(A, 5);
```

Array A will contain:

```
A = {1,2,3,4,5}
```

2. Working with Two-Dimensional Arrays 10 Marks

Write a void function called addToColumn that accepts a two-dimensional array arr[ ][5], the number of rows in that array, a column number, and an int value that is to be added to every element in that column.

Your function must be defined in file addToColumn.cpp and #include the header file addToColumn.h which is available in CourSys. Test your function using the test program addToColumnTest.cpp.

Your function must use the following signature defined in addToColumn.h:

```
void addToColumn(int arr[][5], int numRows, int colNum, int numToAdd);
```

Example:

Given array A[5][5] where:

```
A = {{1,0,0,0,0},
     {0,1,0,0,0},
     {0,0,1,0,0},
     {0,0,0,1,0},
     {0,0,0,0,1}}
```

After calling addToColumn:

```
addToColumn(A, 5, 2, 1);
```

Array A will contain:

```
A = {{1,0,1,0,0},
     {0,1,1,0,0},
     {0,0,2,0,0},
     {0,0,1,1,0},
     {0,0,1,0,1}}
```

3. Practice with Dynamic Arrays as Return Values 20 Marks

This section will contain a series of problems involving functions that accept array parameters and must return a new dynamic array (allocated from the heap) as the return value. Remember, you are not to use cin or cout for these problems! The input will involve arrays and the output should be a newly allocated dynamic array.

For each problem below, there will be an associated header file which your C++ file should include. For example, in the first problem, the header file helloName.h will be provided on the Assignment Page, and your solution should be stored in a separate file called helloName.cpp. Note that file helloName.cpp contains no main() function. That will be provided by the assignment test program that you can download from CourSys.

A single test program called A3B3Test.cpp will be provided. It will include header file A3B3.h that specifies which problems you wanted tested. You will need start with the supplied A3B3.h header and must modify the #defines near the top of the header file to specify which files you have implemented and want tested for marks. You will submit your copy of A3B3.h into CourSys as well.

To use the test program, create a project with A3B3Test.cpp and A3B3.h, and add your solutions as separate C++ source files to the project. When you build the project, it will test each function according to the #define statements found in A3B3.h (which defaults to testing nothing).

(a) helloName

Given a C string parameter called `name`, e.g. "Bob", return a greeting of the form "Hello Bob!".

Remember that C strings must be null terminated, so the above string would contain the four chars: name[0]='B', name[1]='o'; name[2]='b'; name[3]=0; Your function will accept a C string that is null terminated, and must return a new dynamic char array that is also null terminated.

Your function must have the following signature:

```
char* helloName(const char name[]);
```

For example:

```
helloName("Bob")     → "Hello Bob!"
helloName("Alice")   → "Hello Alice!"
helloName("X")       → "Hello X!"
helloName(nullptr)   → "Hello !"
```

(b) makeAbba

Given two C strings, a and b, return the result of putting them together in the order abba, e.g. "Hi" and "Bye" returns "HiByeByeHi".

Your function must have the following signature:

```
char* makeAbba (const char a[], const char b[]);
```

For example:

```
makeAbba("Hi", "Bye")    → "HiByeByeHi"
makeAbba("Yo", "Alice") → "YoAliceAliceYo"
makeAbba("What", "Up")   → "WhatUpUpWhat"
```

(c) doubleChar

Given a string, return a C string where for every char in the original, there are two chars.

Your function must have the following signature:

```
char* doubleChar (const char str[]);
```

For example:

```
doubleChar("The")        → "TThhee"
doubleChar("AAbb")       → "AAAAbbbb"
doubleChar("Hi-There")   → "HHii--TThheerree"
```

(d) zipZap

Look for patterns like "zip" and "zap" in the C string, starting with 'z' and ending with 'p'. Return a string where for all such words, the middle letter is gone, so "zipXzap" yields "zpXzp".

Your function must have the following signature:

```
char* zipZap (const char str[]);
```

For example:

```
zipZap("zipXzap")        → "zpXzp"
zipZap("zopzop")         → "zpzp"
zipZap("zzzopzop")       → "zzzpzp"
zipZap("z")              → "z"
```

(e) fizzString

Given a string str, if the string starts with "f" return "Fizz". If the string ends with "b" return "Buzz". If both the "f" and "b" conditions are true, return "FizzBuzz". In all other cases, return a string copy of the original.

Your function must have the following signature:

```
char* fizzString (const char str[]);
```

For example:

```
fizzString("fig") → "Fizz"
fizzString("dib") → "Buzz"
fizzString("fib") → "FizzBuzz"
```

(f) fizzString2

Given an int n, return the string form of the number followed by "!". So the int 6 yields "6!". Except if the number is divisible by 3 use "Fizz" instead of the number, and if the number is divisible by 5 use "Buzz", and if divisible by both 3 and 5, use "FizzBuzz". Note: the % "mod" operator computes the remainder after division, so 23 % 10 yields 3. What will the remainder be when one number divides evenly into another?

Your function must have the following signature:

```
char* fizzString2 (int n);
```

For example:

```
fizzString2(1) → "1!"
fizzString2(2) → "2!"
fizzString2(3) → "Fizz!"
fizzString2(-1)→ "-1!"
```

(g) fizzArray

Given a non-negative number n, create and return a new int array of length n+1, containing the number n as the first element, followed by the numbers 0, 1, 2, ... n-1 for the remaining n elements. The value of n may be 0. You do not need a separate if-statement for the length==0 case; the for-loop should naturally execute 0 times in that case, so it just works.

The syntax to make a new int array is: new int[desired_length].

Your function must have the following signature:

```
int* fizzArray (unsigned int n);
```

For example:

```
fizzArray( 4) → {4, 0, 1, 2, 3}
fizzArray( 1) → {1, 0}
fizzArray(10) → {10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
fizzArray( 0) → {0}
```

(h) fizzArray2

Given non-negative number n, return a new array of strings of length n, containing the C++ strings "0", "1" "2"... n-1. If n is 0 or less, return `nullptr`. Note that to_string() constructs string for most numeric types.

The syntax to make a new string array is: new string[desired_length]

Your function must have the following signature:

```
string* fizzArray2 (int n);
```

For example:

```
fizzArray2(4) → {"0", "1", "2", "3"}
fizzArray2(2) → {"0", "1"}
fizzArray2(0) → nullptr
```

(i) fizzArray3

Given non-negative start and end numbers, return a new array containing the sequence of integers from start up to but not including end, so start=5 and end=10 yields {5, 6, 7, 8, 9}. The end number will be greater or equal to the start number. Note that a zero length array is valid.

Your function must have the following signature:

```
int* fizzArray3(unsigned int start, unsigned int end);
```

For example:

```
fizzArray3( 5, 10)  → {5, 6, 7, 8, 9}
fizzArray3(11, 18)  → {11, 12, 13, 14, 15, 16, 17}
fizzArray3( 1,  1)  → {}
```

(j) fizzBuzz

This is slightly more difficult version of the famous FizzBuzz problem that is sometimes given as a first problem for job interviews. Consider the series of numbers beginning at non-negative number `start` and running up to but not including `end`, where `start` < `end`, so for example `start`=1 and `end`=5 gives the series 1, 2, 3, 4. Return a new string array containing the string form of these numbers, except for multiples of 3, use "Fizz" instead of the number, for multiples of 5 use "Buzz", and for multiples of both 3 and 5 use "FizzBuzz". This version is a little more complicated than the usual version since you have to allocate and index into an array instead of just printing, and this specifies the start/end instead of just always doing 1..100.

Your function must have the following signature:

```
string* fizzBuzz(unsigned int start,unsigned int end);
```

For example:

```
fizzBuzz(1,  6) → {"1","2","Fizz","4","Buzz"}
fizzBuzz(1,  8) → {"1","2","Fizz","4","Buzz","Fizz","7"}
fizzBuzz(12,17) → {"Fizz","13","14","FizzBuzz","16"}
```

4. Dynamic Two-Dimensional Dynamic Arrays 20 Marks

One way to define a two dimensional dynamic array of doubles is to first create a type for each row (`dblRow`) to store the array of doubles for a given row, then define a type for the Matrix (`dblMatrix`) to store pointers to each of the rows.

```
typedef double* dblRow;
typedef dblRow* dblMatrix;
```

With these definitions, one can define a 4x6 dynamic matrix m as follows.

```
dblMatrix m = new dblRow[4];
for(int i=0; i<4; i++)
   m[i] = new double[6];
```

Create a file called dblMatrix.cpp which #includes dblMatrix.h and implements the two functions listed below. Test your program using the dblMatrixTest.cpp program provided in CourSys.

You must create a function that allocates a dynamic matrix of `Rows` rows by `Cols` columns using the new operator as shown above. All elements of the newly allocated array must be initialized to the value specified as initVal. Your function must have the following signature:

```
dblMatrix allocDblMatrix(int Rows, int Cols, int initVal);
```

You must also create a function that deallocates a dblMatrix using the delete operator. Care must be taken to delete the rows before deleting the dblMatrix. The parameter m must be set to nullptr when this routine returns. Your function must have the following signature:

```
void deallocDblMatrix(dblMatrix& m, int numRows);
```

With dblMatrix defined as described above, we can access any element in the array using standard C++ array notation. For example, to set the double in row 2, column 5, to the value 3.14, we can do the following:

```
m[2][5] = 3.14;
```

Create an addDoubleToColumn function that, similarly to the addToColumn function defined in question 2, adds a double to a specific column of a dblMatrix. Your function must use the following signature defined in addDoubleToColumn.h:

```
void addDoubleToColumn (dblMatrix& m,int numRows,int colNum,double numToAdd);
```

Unit test your functions to ensure that they work as expected. The program dblMatrixTest.cpp will be provided prior to the due date so you may perform integration testing before submitting into CourSys.

5. Classes with Dynamic Arrays as Members 25 Marks

The dblMatrix array type defined (using typedef) in question B4 allows a matrix of any size to be dynamically allocated. Access to any element in the array is done using the standard array notation. For example, to set the value in row 2 column 5 to the value 3.14, one does the following:

```
m[2][5] = 3.14;
```

This works well, but has the drawback that array m does not contain any information about the number of rows and columns. While we can access individual elements within a dblMatrix, we have no way to ensure that the indexes are in range, nor create general routines that operate on a dblMatrix without needing to pass in dimensions of the matrix.

The solution is to encapsulate dblMatrix in a class called Matrix. This class contains member variables to store the number of rows and columns associated with the dblMatrix. It also provides overloaded operators for << and for the array indexing operators [ ].

You must implement Matrix.cpp which implements the Constructors for the Matrix class. Use the NEW operator to allocate enough dynamic memory to store the number of rows and columns specified in the constructor.

Because this class contains members that use dynamic memory, you must also implement "*The Big Three*": (see wk10 slides on "Arrays and Classes")

- **The Copy Constructor**
- **The Assignment Operator**
- **The Destructor**

You must also implement the `addToColumn` method that adds a double to all entries in the specified column. Refer to Matrix.h in CourSys for the full declaration of Matrix class needed for this assignment.

```
class Matrix {
public:
   // Constructors - use NEW operator to alloc dynamic memory for m
   Matrix();
   Matrix(int rows, int cols);
   Matrix(int rows, int cols, int dataLen, double *data);
   // Copy Constructor - Allocs new m, and copies data from mat
   Matrix(Matrix& mat)
   // The Assignment Operator
   void operator=(const Matrix& right_side);
   // Destructor - Calls delete[] to return memory to heap
   ~Matrix();
   // Method to add a double to all entries in column colNum
   void addToColumn(int colNum, double numToAdd);
};
```

Header Matrix.h defines class `Matrix` and its helper class `Row`. Helper functions are found in MatrixHelper.cpp. MatrixTest.cpp contains test code. Submit your completed Matrix class to CourSys as file Matrix.cpp.

C. Programming for Bonus Marks – To be completed individually

These programming problems are optional. You may complete them if you choose and will receive bonus marks that may be used to top up your overall Assignment portion of your course grade.

1. Matrix Methods 10 Marks

This question requires that you have successfully completed question B5 since this question depends on using your completed Matrix class.

Create project MatrixMethods that contains the following files: Matrix.h, MatrixHelper.cpp, MatrixMethods.cpp, and MatrixMethodsTest.cpp. Your code will be placed in MatrixMethods.cpp, while the other files will be available for download from CourSys.

Recall from Question B4 that the Matrix class includes getters and setters for determining the number of Rows and Columns in a Matrix, plus methods for getting and setting individual elements (see Matrix.h for details):

```
// getters and setters
int getNumRows() const;
int getNumCols() const;

// Getter returns zero if row or col is out of range
double  getElement(int row, int col) const;

// Setter does nothing if row or col is out of range
void    setElement(int row, int col, double newVal);
```

For example, the following array A[5][5] could also be represented as a object M of our Matrix class by calling the Matrix constructor with a pointer to the first element of the array A[0][0].

```
double A[5][5] = {
  {  0,   2,  0, -2, 2.5},
  {  2,   0,  1,  0, 1.6},
  {  0,   1,  0,  1,   0},
  { -2,   0,  1,  0,   0},
  {2.5, 1.6,  0,  0,   0}
};

Matrix M(5,5,25,&(A[0][0]));
```

In this example, M.getNumRows() returns 5 while M.getElement(4,1) returns the double 1.6. Because we created an overloaded [ ] operator, one can also retrieve the element at row 4 column 1 using the notation M[4][1].

Implement each of the following functions in MatrixMethods.cpp using your Matrix class, and test within your MatrixMethods project:

(a) Write a function that returns true if m has at least one row and one column. It must have this signature:

```
bool is_matrix(const Matrix& m);
```

(b) Write a function that prints a neatly formatted version of m to the specified ostream with one row per line, columns evenly spaced such as the example provided below. Be creative, it need not match this example.

```
void print(ostream& outStream, const Matrix& m);
```

```
Row 0:    0    2    0   -2  2.5
Row 1:    2    0    1    0  1.6
Row 2:    0    1    0    1    0
Row 3:   -2    0    1    0    0
Row 4:  2.5  1.6    0    0    0
```

(c) Write a function that tests if m is a square matrix. A matrix is square if it has the same number of rows as columns.

```
bool is_square(const Matrix& m);
```

(d) Write a function that tests if m has only 0s on its main left to right diagonal. For example, matrix A above has a 0 diagonal.

```
bool has_zero_diag(const Matrix& m);
```

(e) Write a function that tests if m is symmetric. For example, the matrix A above is symmetric.

```
bool is_symmetric(const Matrix& m);
```

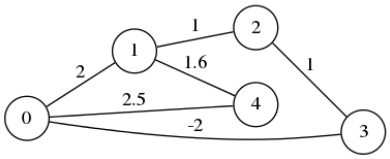
(f) Write a function that returns the sparsity of a matrix m. The sparsity of a matrix is defined to be the number of 0s in it divided by the total number of entries. For instance, the matrix A above has sparsity of 13/25 or 0.52.

```
double sparsity(const Matrix& m);
```

(g) Write a function that tests if m is simple graph. A matrix m is a simple graph just when it is square, symmetric, and has a zero diagonal.

```
bool is_simple_graph(const Matrix& m);
```

For example, the matrix A above is a simple graph that looks like this:



(h) Write a function that returns a vector containing the degrees of all the nodes in a simple graph (i.e. a matrix m for which is_simple_graph(m) returns true). The degree of a node in a simple graph is the number of edges attached to that node. For example, for the matrix A above:

node 0 has degree 3
node 1 has degree 3
node 2 has degree 2
node 3 has degree 2
node 4 has degree 2

Here is the function header:

```
vector<int> degrees(const Matrix& m);
```

2. Linked Lists 20 Marks

Linked lists are a dynamic data structure used to store a collection of data elements. Linked lists are more flexible than arrays. When dynamic arrays must change capacity, an entirely new array must be allocated and the contents of the old array copied into the new one. Link Lists avoid by allowing the linked list to grow or shrink one element at a time.

Elements within a linked list are called Nodes. Each node contains data plus a pointer used to point to the next element in the list. A special node called the Head points to the first node in the list. The last node in the list contains a special pointer value called `nullptr`. If Head's pointer is the `nullptr`, which indicates that the list is empty.

Linked Lists are excellent for storing (key, value) pairs used in databases. One can search a linked list by comparing each node to some key. If the key matches the node's key, return the value associated with that node, otherwise continue searching by following the next pointer until the either the key is found, or a `nullptr` is encountered.

New nodes can be added to the end of the list by creating a new node using the NEW operator, setting its pointer to `nullptr`, then setting the last node's pointer to point to this new node. Insertions and deletions are a little trickier and those concepts will be discussed in class.

A simple struct called Node is defined in header LinkedList.h as follows:

```
struct Node {
   string  key;
   int     value;
   Node   *nextNode;
}
```

Implement the LinkedList methods declared in LinkedList.h to help our Zoo Keeper manage his Zoo Animals with his ZooKeeper.cpp test program:

```
class LinkedList {
public:
    LinkedList(); // Default Constructor

   // Methods for managing a LinkedList of Nodes
   bool findKeyAndGetValue(string key, int& value   );
   void addKeyValuePair   (string key, int  value   );
   void setKeyValue       (string key, int  newValue);
   void delNode           (string key);
   void delAllNodes       (); // Delete all nodes in the list
   int  length            (); // return number of Nodes in the list

   // Method for displaying all (key,value) pairs in a line
   friend ostream &operator<<(ostream &os, const LinkedList &list);
private:
   Node *head = nullptr;
};
```

D. Submission – To be completed by Students Individually

Due date: Tuesday Mar 29th 11:59pm

Students are responsible for submitting the requested work files by the stated deadline for full marks. It is the student's responsibility to submit on time. Submissions via email will *NOT* be accepted.

You must submit your final version of the following files before the deadline. All written answers for Parts A must be included in separate Word or PDF documents which are double spaced and include the Student's Name, Student Number, and email at the top. Students must ensure that all submitted code compiles and is properly commented and formatted for readability.

Submit into CourSys:

Part A	**: a3Writeup.doc, a3Writeup.docx, or a3Writeup.pdf**
Question B1	**: bubbleSort.cpp**
Question B2	**: addToColumn.cpp**
Question B3	**: A3B3.h, helloName.cpp, makeAbba.cpp, …, fizzBuzz.cpp**
Question B4	**: dblMatrix.cpp**
Question B5	**: Matrix.cpp**
Question C1	**: MatrixMethods.cpp**
Question C2	**: LinkedList.cpp**

Submission of solutions for D1 and D2 are optional.

Late Submissions: If you submit any component after the deadline, it affects the timestamp for the entire submission. If any component is submitted late, the late penalty will be applied to all components.

Late Penalty is 10% per day late, or portion of a day late. Late submissions will be accepted up to two days late with the final cut-off Apr 1st at 11:59pm.

Files are to be submitted into CourSys under Assignment 3.