Assignment 3 –Dynamic Arrays and Classes Solutions

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 MUST 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?

Stride refers to the number of bytes for the base element of an array. An array element of ints has a stride of 4 bytes (the size of an int on a 64-Bit Ubuntu machine) while an array of pointers has a string of 8 bytes (the size of a pointer on a 64-bit machine).

(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?

Since argv is an array of pointers, its stride is 64-bits, and the address of argv[i] is calculated as: X + 8*i

(c) When passing an array as a parameter to a function, can that function make changes to the contents of the array?

Yes, the array is not passed by copying the array, but instead a pointer to the array is passed to the function. Since the array's address is passed to the function, the function can modify the array's contents, provided the `const` modifier is not specified in the parameter list.

(d) When passing an array as a parameter, how do we ensure that the function cannot change the array elements?

Add the `const` keyword to the array formal parameter in the function declaration. For example: `void foo(const int Arr[10]);`

(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.

Address of A[row][col]

= Start Address of row + col*stride of an int
= Start Address of Array A + row*stride of a row + col*stride of an int
= 1000 + row * stride of a row + col*stride of an int
= 1000 + row *(stride of an int * number of ints per row)+ col*4
= 1000 + row *(4 * 6)+ col*4
= 1000 + 24*row + 4*col

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.

```
typedef double* DoubleArrayPtr;
```

(b) Define a variable of type DoubleArrayPtr called p and initialize p to point to an array of 10 doubles using the new operator.

```
DoubleArrayPtr p = new double[10];
```

(c) When allocating memory for array p above, what is the name of the memory that the NEW operator allocates memory from?

The Heap

(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.

```
delete [] p;
```

(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.

From slide 32 of slides wk10.1 - Dynamic Arrays:

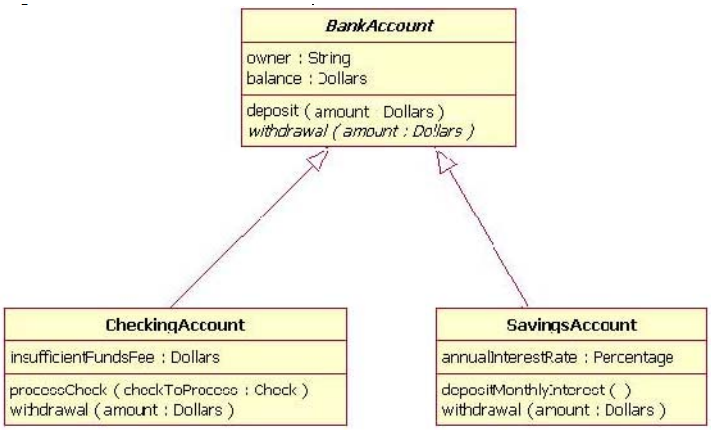
```
typedef int* IntArrayPtr;
IntArrayPtr *m = new IntArrayPtr[4];
for(int i = 0; i<4; i++)
    m[i] = new int[4];
```

Since array A is defined using the method used by m above,
Address of A[row][col]
= Start Address of row + col*stride of an int
To find the start address of row, need to locate the row pointer within A
Row Pointer address:
= Start Address of Array A + row * stride of a pointer
= 1000 + 8 * row
Dereference Row Pointer Address to get start address of the row:
= *(1000 + 8*row)
Address of A[row][col]=
= Start Address of row + col*stride of an int
= *(1000 + 8*row) + col*4
= A[row]+4*col

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.

Do



(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.

```
// Anything close is ok.
class BankAccount {
public:
    void deposit(Dollars amount);
    void withdrawal(Dollars amt); // ok to add virtual keyword
private:
    string  owner;
    Dollars balance;
};
```

(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.

```
class CheckingAccount : BankAccount {
public:
    void processCheck(Check checkToProcess);
    void withdrawal(Dollars amt);// Must use override keyword if
                                 // used virtual for BankAccount
private:
    Dollars insufficientFundsFee;
};
```

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

Base class: `BankAccount`

Derived classes: `CheckingAccount and SavingsAccount`

(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)?

MySavings.withdrawal($10) invokes: SavingsAccount:: withdrawal
MySavings.deposit($10) invokes: BankAccount::deposit

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);
```

Solution:

```
void bubbleSort(int d[], int num) {
// Any sort will do, but this is bubbleSort
// ***************************************
// This FOR loop looks to "bubble up" the
// largest double from d[0] to d[i] to d[i].
// i starts with i=num-1, so first pass finds
// the largest number in array and places it
// in d[num-1]. Next loop finds next largest
// and places it in d[num-2], and so on.
for(int i=num-1; i>0; i--)

    // This FOR searches for the largest of
    // those remaining to be bubbled into d[i].
    for(int j=0; j<i; j++)

    // Check if numbers are out of order
    // Swap them if they are.
      if (d[j] > d[j+1]) {
            // d[j] and d[j+1] in wrong order
            // So let's swap them!
            // ******************************
            int tmp = d[j];
            d[j]      = d[j+1];
            d[j+1]    = tmp;
      }
}
```

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}}
```

Solution:

```
#include "addToColumn.h"
void addToColumn(int arr[][NUM_COLS],int numRows,int colNum,int numToAdd) {
// check for bad input before starting
// ***************************************
// For Quiz, no marks lost if did not do check
// ***************************************
if (arr != nullptr)
   if (numRows > 0)
       if ((colNum >= 0) && (colNum < 5))

         // Valid input
         // Loop through each row adding numToAdd
         // ************************************
         for(int row=0; row<numRows; row++)
             arr[row][colNum] += numToAdd;
}
```

3. Practice with Dynamic Arrays as Return Values 20 Marks

(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 !"
```

Solution:

```
#include <cstring>
#include "helloName.h"

char* helloName(const char name[]) {
   // set len=0 in case name equals nullptr
   int len = 0;

   // don't call strlen if name==nullptr
   if (name != nullptr)
        len = strlen(name);

   // Allocate enough chars for retStr
   char *retStr = new char[len+8];

   // Use strcpy because retstr is not yet
   // initialized. strcat may fail if used.
   strcpy(retStr, "Hello ");

   // cat name if it was not nullptr
   if (name != nullptr)
        strcpy(retStr+6, name);

   // Complete answer by adding the "!" string
   strcpy(retStr+len+6, "!");

    // return completed retStr to caller
    return retStr;
}
```

Alternate Solution:

```
char* helloName_v2(const char name[]) {
        // set len=0 in case name equals nullptr
        int len = 0;

        // don't call strlen if name==nullptr
        if (name != nullptr)
             len = strlen(name);

        // Allocate enough chars for retStr
        char *retStr = new char[len+8];

        // Use strcpy for "Hello " because
        // retStr memory is not zeroed
        strcpy(retStr, "Hello ");

        // cat name if it was not nullptr
        if (name != nullptr)
             strcat(retStr, name);

        // Complete answer by adding the "!" string
        strcat(retStr, "!");

        // return completed retStr to caller
        return retStr;
}
```

Remaining Solutions can be found in CourSys