Assignment 3 –Dynamic Arrays and Inheritance

Note: This is a Preview! More Questions are coming in Version 1.0

Required Reading

Problem Solving with C++

Chapter 7 – Arrays
Chapter 9 – Pointers and Dynamic Arrays
Chapter 10 – Classes
Chapter 11 – Classes and Arrays
Chapter 13 – Pointers and Linked Lists
Chapter 15 – Inheritance

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 submission 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

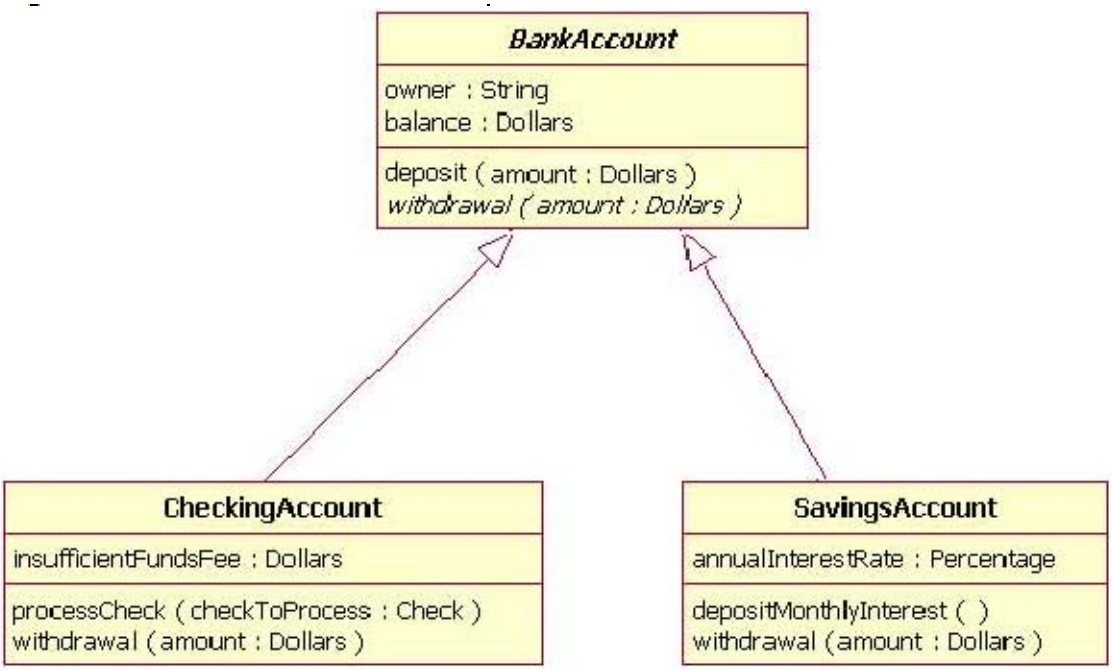
(a) What does the term Stride refer to when discussing arrays?
(b) Given an array of pointers like argv, on a 64-bit machine, how does one compute the address of argv[i] if the argv array starts at address X?
(c) When passing an array as a parameter to a function, can that function make changes to the 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 that the stride for an int is 4 bytes 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)?

C. 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] < a[1] < … < a [ num -1]
```

Your function should be submitted in a file called bubbleSort.cpp and #include the header file bubbleSort.h that will be made available in CourSys.

Test your function using the test program bubbleSortTest.cpp which will also be provided in CourSys.

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

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

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 program should be called 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, 2);
```

Array A will contain:

```
A = {{1,0,2,0,0},
     {0,1,2,0,0},
     {0,0,3,0,0},
     {0,0,2,1,0},
     {0,0,2,0,1}}
```

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 called helloName.cpp.

A single test program called A3C1.cpp will be provided. It will include the header file A3C1.h that specifies which problems you wanted tested. You will need start with the supplied A3C1.h header and must modify the #defines near the top of the header file to specify which files you have implemented and want tested. You will submit your

(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 the string unchanged.

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!"
```

(g) fizzArray

Given a 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 (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 a number n, create and return a new array of C++ strings of length n, containing the C++ strings "0", "1" "2"… n-1. If n is 0, return `nullptr`. Note function 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"}
fizeArray2(0) → nullptr
```

(i) fizzArray3

Given 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 (int n);
```

For example:

```
fizzArray3( 5, 10)  → {5, 6, 7, 8, 9}
fizzArray3(11, 18)  → {11, 12, 13, 14, 15, 16, 17}
fizzArray3( 1,  3)  → {1, 2}
```

(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 `start` and running up to but not including `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 (int start, 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","Fizz"}
```