Farmingdale State College
STATE UNIVERSITY OF NEW YORK

BCS 370: Data Structures

Lab 1: unsorted array list

Lab 1 Objectives

Through hands-on coding, we will

- review the core concepts of
  - Arrays
  - Multiple files compilation
  - Pointers and Dynamic Allocated Arrays
  - Heap management
  - Visual Studio Debugging and Profiling Debugging
- Today’s task complete stage 1 of lab
  - implement the core functions of *data_management.cpp*.

My little data management

- *My little data management* is a lab designed to support common operations on various data records, including:

  add/search/delete/display/update/store.

- As an opening project to BCS370, The lab exposes students to a very fundamental data structure: **a list based on an unsorted array**, as well as the operations on it.

Get started

- Download the starting files from the **Labs/Lab 1** directory on Blackboard.
- You will receive 6 files:
  - main.cpp -> the driver file
  - record.cpp -> data record
  - data_management.cpp -> holds an unsorted array and supports the operations on the array.
  - record.h
  - data_management.h
  - small_records.dat
  - big_records.csv
- Please start a new Visual Studio Project and add these files under **source files** and **header files.**

Get started

- Build the project, and run the project.

  You should see a console-based menu with options. At this point of time, most of these operations do not work, because they are waiting to be implemented by you!
- Your instructor will briefly walk you through the starting files. You can request the instructor to demo the completed project as many times as you want.
- You do not need to add or delete any file. All your implementation should be done in **data_management.cpp**. You do not need to change any other file. If you do, you need to document your changes

Data files

- All labs in this course use data files and eventually big data files.
  - Small data does not show the advantage of good data structures.
- We will always start with small datasets, then big datasets.
- We will support various structures of data files, but again we start with simple structures, such as a collection of contact records which only contains user name and user cellphone.
  - The user name represents the key while the cell represents the value.
  - We will formalize complex data format after learning template.

Stage 1

- We will start with fixed-size, dynamically allocated array.
  - The constructor of *data_management* allocates the dynamic array and loads the data records from the file to the array.
  - But the array is fixed-sized.
  - We will implement the operations of list/search/delete/insertion/update

  based on **the array in the memory, NOT directly on the file.**
    - There is a function dedicated to write back the updated array to the disk.
    - We shall focus on how to manipulate the array in order to support these operations.

Stage 2

- We will move on to expandable dynamic array to overcome the drawback of fixed-size array.
  - Data size can vary over time.
  - The array can expand itself when the amount of data records is going beyond the array’s capacity, and shrink itself when the amount of data records is way below the capacity.
  - There MUST be no data loss.
- The resizing function is to be implemented.
  - Make sure there is no memory leak: all the dynamic allocated memory are properly freed after use.
  - Use the big dataset, *big_records.csv* to test your code.
  - Practice debugging skills, such as profiling your own code. In this stage, you are asked to print out how many times the array has been resized as part of profiling.

Stage 3

- We will complete the *data_management* class by implementing its big three: the copy constructor, the destructor and the assignment operator overloading.
- Big three are immensely important in order to support deep copy and thus clone function, which are the bedrocks of the data replication and backup.
- All the data structures implemented in C++ must have big three done right.