Lab 3

Table of Contents

Class Container + Review

What is Class Container in C++

- “A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.” — https://cplusplus.com
  - example: stack, queue, linked lists, trees
- Remember CSEN 12 ADT in C
  - struct
  - generic SET (void *)
- In Our Case
  - we have a Roster, that store a collection of students.

Array as Pointer

- In C
  - assume we have int arr[20];
  - when access arr, we get a pointer that points to the 1st element of arr
- In C++
  - same idea
- Pointer Arthematic
  - arr ++ ➜ point arr to the next element
  - *arr ➜ access value pointed by arr
  - arr + (int)i ➜ point arr to i-th element from the current location of arr

Lab 3 Description

Project Structure

- lab3sampleinput.txt
- Roster
  - roster.h
  - roster.cxx
  - rostermain.cxx
- Makefile

roster.h (Class Containers)

- Two Classes
  - Student (regular class)
  - Roster (class container)

Student (Regular Class)

- Represents a student record object
- Public Variables
  - using ID_t = unsigned int; // define our own type called **ID_t** which is an **integer** type (7-digits)
- Private Variables
  - ? // what is unique about a student?
  - ? // 1st part of student name
  - ? // last part of student name
  - ... // more?

Student (Regular Class)

- Functions to be implemented
  - Student Constructor
    - default? parameterized? copy? multiple?
  - ostream operator<<(ostream& os, const Deck& d)
    - what it means to print a Student Object?
  - ... // more?

Roster (Class Container)

- Represents a collection of Student Record Object
- Public Variables
  - using T = Student; // specify the type **T** to be **Student, WHY?**
  - static const int CAPACITY=30; // capacity of the roster record
  - … // more?
- Private Variables
  - ? // an array to store the Student Objects?
  - ? // current index ?
  - ? // index to insert ?
  - … // more?

Roster (Class Container)

- Functions to be implemented
  - Roster Constructor
    - default? parameterized? copy? multiple?
  - bool insert(T &);
  - void erase(Student::ID_t);
  - T* begin(void); // return a pointer to the first record of roster data
  - T* end(void); // return a pointer to the last record of roster data
  - T* next(void); // return the next record to previous actions, HOW WE KNOW?
  - ... // more?

rostermain.cxx (rostertest)

- Main Function For Test
  - takes user input from input file
    - `rosterTest < input_file > outout_file` takes care of it
    - Or you can use ifstream
  - reads line by line
    - already done in sample main
  - format for line input
    - <command key> <params (depend on key)>
    - <command key> <params> format
      - <A> <ID> <first> <last> – insertion of a student with the input ID, first, last
        - ex: A 1234567 John Doe
      - <X> <ID> – deletion of student with ID
        - ex: X 7654321
      - “L” – list roster of students
        - ex: L

rostermain.cxx testing tips (rostertest)

- Edge Cases
  - insertion
    - under what cases we can not insert a student into the roster?
  - deletion
    - under what cases we can not delete a student into the roster?
  - list
    - under what cases we can not list the roster?

Deliverables

- All .cpp files
- All .h files
- Makefile
- Testing materials (e.g. input, output)
- Test plan

Demo

- Test Plan
- Corresponding test scripts
- Code compilation/run

Other Tips

- Test code frequently
- Test your code comprehensively
  - Think about what needs to be tested
  - Points will be deducted if you missed critical test cases

Don’t Forget

- Submit the code before the deadline
- File with guide to implement and hints are in Camino
  - Make sure your code can run on school Linux server