Short courses

FOR EVERY STAGE OF YOUR LEARNING JOURNEY

KIT205 Data Structures and Algorithms
Assignment 2

Due 23rd April, 11:55pm

The University has recently decided to expand its short course offerings. This will result in a large number of courses, with relatively few students per course. Having recently optimised the student database for MOOCs (many students per course), this will require a rewrite of the database code. (Note: this is a fictional and simplified scenario... names have been changed to protect the innocent)

The previous database was designed as a linked list of courses, with a bst tree of students. The new database will be designed as a bst tree of courses, with each course having a linked list of enrolled students.

The student database must support the following operations:

1. Add course
2. Remove course
3. Enrol student
4. Un-enrol student
5. Print an ordered summary of courses and the number of students enrolled in each course
6. Print an ordered list of students enrolled in a course
7. Print an ordered list of courses that a given student is enrolled in

Assignment Specification - Part A (~80% of marks)

For this part of the assignment you will implement a prototype that uses a binary search tree of courses (sorted by course *name*), with each course having a name and a linked list of enrolled students. You must use the BST and linked list code as developed in the tutorials, however the data structures will be modified for the new data (and functions will also require minor modifications to accommodate these changes). The following definitions MUST be used:

```
typedef char* String;

typedef struct listNode{
    long id;
    struct listNode *next;
} *ListNodePtr;

typedef struct list {
    ListNodePtr head;
} StudentList;

typedef struct course {
    String name;
    StudentList students;
} Course;

typedef struct bstNode {
    Course course;
    struct bstNode *left;
```

```
    struct bstNode *right;
} *BSTNodePtr;

typedef struct bst {
    BSTNodePtr root;
} CourseBST;
```

The ListNodePtr and StudentList definitions and (modified) linked list functions must be placed in files list.h and list.c. The Course, BSTNodePtr and CourseBST definitions and modified BST functions must be placed in files bst.h and bst.c.

All remaining code should be placed in a file called main.c that contains the main function and all program logic (**there should not be any program logic or I/0 code in your linked list or bst files**). This file will contain separate functions for each of the seven operations listed above, as well as an eighth function to handle termination. Other functions may be added if required.

Program I/O

All interactions with the program will be via the console. Operations 1-7 will be selected by typing 1-7 at the command prompt. Quitting the application will be selected by typing 0. For example, the following input sequence would create a course called "abc123" and enroll a student with id "123456" in that course then quit the application:

```
1
abc123
3
abc123
123456
0
```

Note that this sequence shows the input only, not the program response (if any). You are free to add prompts to make the application more user friendly, but this will not be assessed (although it may be useful).

Program output in response to operations 5-7, should be as minimal as possible. You may print a header if you wish, but this should be followed by one record per line with spaces separating data. For example, in response to operation 5, the output might be:

```
Current enrolments:
abc123 32
def123 0
def456 10236
```

I/O Restrictions

You may assume that all input will always be in the correct format and contain no logical errors.

- Commands will always be in the range 0-7
- Course names will always be strings less than 100 characters long and may contain any alpha-numeric characters (**no spaces**)
- Student ids will always be positive integers in the range 0-999999
- The user will never attempt to add a student to a non-existent course
- The user will never attempt to print data for a non-existent course
- The user will never attempt to print data for a non-existent student
- The user will never attempt to remove non-existent students or courses

Note: Courses that contain enrolled students may be removed, in which case student data for that course should also be removed. Courses with zero students should not be automatically removed.

Memory Management

Course names should be stored in appropriately size dynamically allocated memory. Names will always be less than 100 characters long. For example, the course name "abc123" would be stored in a char string of length 7.

Removing (un-enrolling) a student or removing a course should free all associated dynamically allocated memory. Removing a course should free all memory for the enrolled students as well as the course. The quit function should also free all dynamically allocated memory.

free all dynamically allocated memory.

Assignment Specification - Part B (~20% of marks)

This part of the assignment should only be attempted once you have fully implemented and thoroughly tested your solution to part A. It would be better to submit a complete part A and no part B than to submit a partially complete part A and part B. Part B is worth only 20% of the marks, but will require more than 20% of the effort.

The requirements for this part of the assignment are exactly the same as for Part A except that an AVL tree must be used to store courses, rather than storing them in a standard BST. To implement the AVL tree, reuse the BST struct definitions, but add a height variable to the node struct. Leave all BST functions in place, but add AVL functions to your bst.h and bst.c files. Add a switch in your main.c file so that it is easy for the marker to switch between BST and AVL functions (e.g. to switch between bst_insert and avl_insert).

Minimal assistance will be provided for this part of the assignment. No assistance at all will be given unless you can demonstrate a fully implemented and thoroughly tested solution to part A.

Testing

It can be very time consuming to thoroughly test a program like this when all input is done manually (imagine testing that your solution can manage 1000's of students and courses). A common method of testing code like this is to use input redirection (and possibly output redirection). When using input redirection your code runs without modification, but all input comes from a file instead of from the keyboard.

This facility is provided in Visual Studio through the project properties dialog. For example, to redirect input from a file called “test.txt”, you would add:

```
<"$(ProjectDir)test.txt"
```

to Configuration Properties|Debugging|Command Arguments. This will be demonstrated in tutorials.

Some small test files have been provided with input files (input1, input2) you can use for redirection and output files (output1, output2) that you can use to check for correct output. **However, it is recommended that you construct your own larger files to fully test your program!** As well as larger test files, it would also be wise to construct files that test edge cases.

Testing is very important! Few marks will be deducted for minor programming errors, but *many* marks may be deducted if these errors could be easily fixed if identified by thorough testing.

Assignment Submission

Assignments will be submitted via MyLO (using the Assignment 2 dropbox). Submissions should consist of a single zipped Visual Studio project. You should use the following procedure to prepare your submission:

- Make sure that your project has been thoroughly tested
- Choose “Clean Solution” from the “Build” menu in Visual Studio. This step is very important as it ensures that the version that the marker runs will be the same as the version that you believe the marker is running.
- Quit Visual Studio and zip your entire project folder
- Upload a copy of the zip file to the MyLO dropbox

History tells us that mistakes frequently happen when following this process, so you should then:

- Unzip the folder to a new location
- Open the project and confirm that it still compiles and runs as expected
    - If not, repeat the process from the start (a common error occurs when copying projects, where the code files you are editing end up existing outside of the project folder structure – and therefore don’t get submitted when you zip the folder)

Learning Outcomes and Assessment

This assignment is worth 10% of your overall mark for KIT205. Your submission will be assessed by examining your code and by running additional tests in Visual Studio. If your submission does not run in Visual Studio 2019 on Windows, it cannot be assessed.

Grades will be assigned in accordance with the Assignment 2 rubric.

The assignment contributes to the assessment of learning outcomes:

The assignment contributes to the assessment of learning outcomes:

LO1 Transform a real-world problem into a simple abstract form that is suitable for efficient computation
LO2 Implement common data structures and algorithms using a common programming language
LO3 Analyse the theoretical and practical run time and space complexity of computer code in order to select algorithms for specific tasks

---