KIT205 Data Structures and Algorith
Assignment 2

Due 10th September, 11:55pm

Due to COVID, the University has had recent experience in running courses online. This p
expand online courses to become MOOC (Massive Open Online Course) with many thous
in each unit. This has the potential to create performance issues for the student record sy

Your task is to develop a prototype system to test different data structure options for the

The student database must support the following operations:

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

Assignment Specification - Part A (~80% of mark

For this part of the assignment you will implement a prototype that uses a binary search tr
prototype you will store only student id), with each BST node also storing a linked list of ur
enrolled in (for the prototype, you will store only the unit code, as a string). You must use t
code as developed in the tutorials, however the data structures will be modified for the ne
also require minor modifications to accommodate these changes). The following definitior

```
typedef char* String;

typedef struct listNode{
    String unit_code;
    struct listNode *next;
} *ListNodePtr;

typedef struct list {
    ListNodePtr head;
} UnitList;

typedef struct bstNode {
    long student_id;
    UnitList units;
    struct bstNode *left;
    struct bstNode *right;
} *BSTNodePtr;

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

The ListNodePtr and UnitList definitions and (modified) linked list functions must be place
The BSTNodePtr and StudentBST definitions and modified BST functions must be placed i

All remaining code should be placed in a file called main.c that contains the main function a
logic (**there should not be any program logic or I/0 code in your linked list or bst files**). Th
separate functions for each of the seven operations listed above, as well as an eighth funct
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 b
command prompt. Quitting the application will be selected by typing 0. For example, the fo
would add a student with id “123456”, and then enroll student "123456" in the unit “abc1
application:

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

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

Program output in response to operations 5-7 should be as minimal as possible. You may r

Program output in response to operations 5-7, should be as minimal as possible. You may p
but this should be followed by one record per line with spaces separating data. For examp
operation 5, the output might be:

```
Unit enrollments:
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 er

- Commands will always be in the range 0-7
- Unit names will always be strings less than 100 characters long and may contain any characters (**no spaces**)
- Student ids will always be positive integers in the range 0-999999
- The user will never attempt to enrol a non-existent student in a unit
- The user will never attempt to print data for a non-existent student
- The user will never attempt to remove non-existent students
- The user will never attempt to unenrol a student from a unit that they are not enrol

Memory Management

Unit names should be stored in appropriately size dynamically allocated memory. Names
100 characters long. For example, the course name “abc123” would be stored in a char str

Removing (un-enrolling) a student should free all associated dynamically allocated memor
the units that they are currently enrolled in. The quit function should also free all dynamic

Assignment Specification - Part B (~20% of marks

This part of the assignment should only be attempted once you have fully implemented an
solution to part A. It would be better to submit a complete part A and no part B than to sul
part A and part B. Part B is worth only 20% of the marks, but may require more than 20% (
reading and testing).

The requirements for this part of the assignment are exactly the same as for Part A except
used to store students, rather than storing them in a standard BST. To implement the AVL
struct definitions, but add a height variable to the node struct. Leave all BST functions in p
functions to your bst.h and bst.c files. Add a switch in your main.c file so that it is easy for t
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 wi

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 testing that your solution can manage 1000's of students and courses). A common metho is to use input redirection (and possibly output redirection). When using input redirection 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 exampl file called "test.txt", you would add:

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

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

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

Testing is very important! Few marks will be deducted for minor programming errors, but 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 s zipped Visual Studio project. You should use the following procedure to prepare your subr

- Make sure that your project has been thoroughly tested
- Choose "Clean Solution" from the "Build" menu in Visual Studio. This step is very im the version that the marker runs will be the same as the version that you believe the
- Quit Visual Studio and zip your entire project folder (to save space you may remove before zipping - but this is not required)
- Upload a copy of the zip file to the MyLO dropbox

History tells us that mistakes frequently happen when following this process, so you shoul

- 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 copyi code files you are editing end up existing outside of the project folder structu 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 as
code and by running additional tests in Visual Studio. If your submission does not run in Vi
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:

LO1 Transform a real-world problem into a simple abstract form that is suitable for
LO2 Implement common data structures and algorithms using a common programm
LO3 Analyse the theoretical and practical run time and space complexity of comput
algorithms for specific tasks