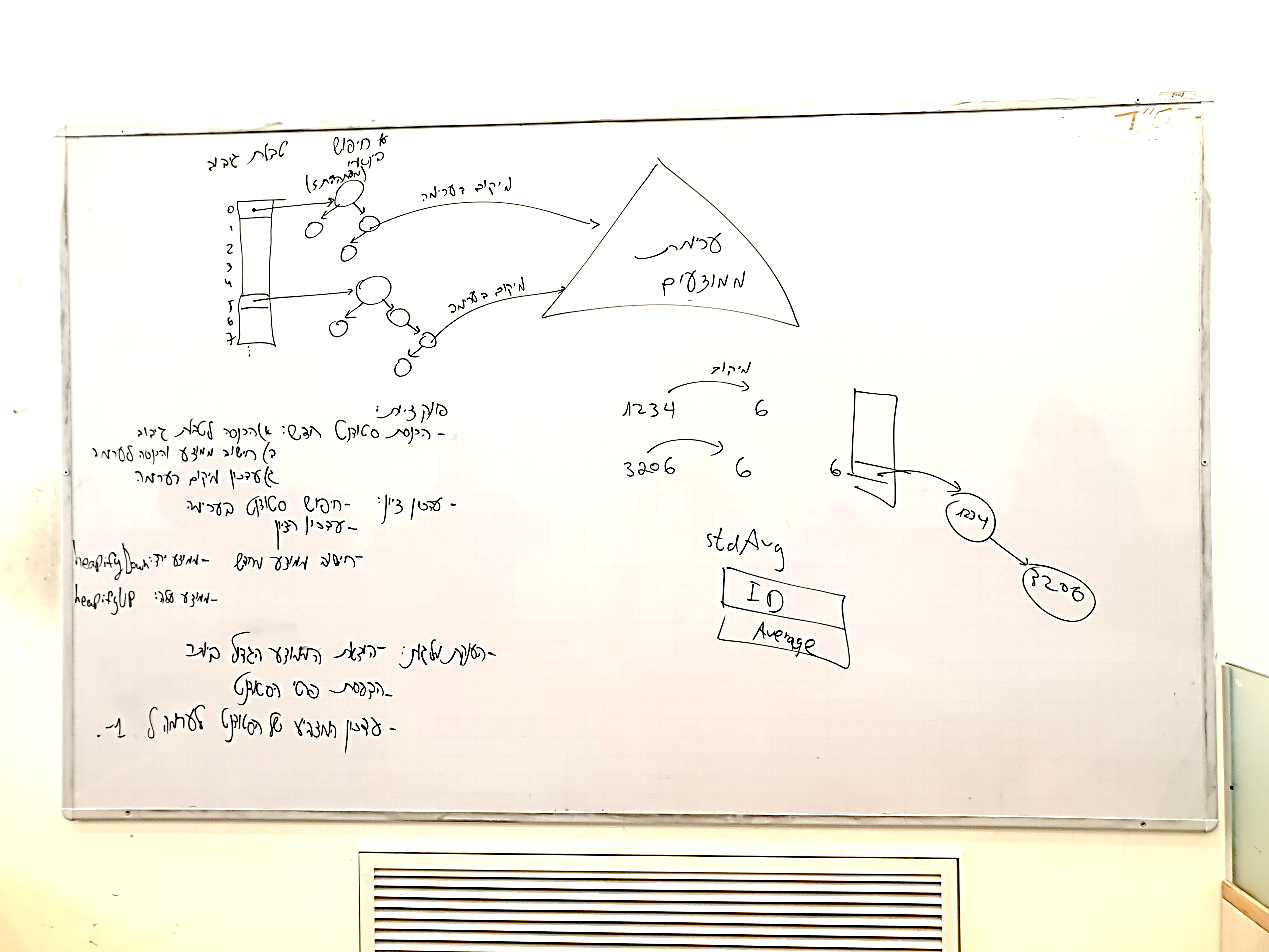
**תרגיל סיכום – ערימה, גיבוב עצים**

**להגשה בתאריך 17/2/2019 עד 23:55**



הערה: מבנה הקבצים צריך להראות כך:

**main.cpp**

#include "binTree.h"

#include "heap.h"

**binTree.h**

#include "student.h"

struct binNode

**heap.h**

struct stdAvg

struct heap

**student.h**

struct course

struct student

1. **בסעיף זה אפשר לבחור כל חישוב של גיבוב שתרצו, החישוב פה הוא הצעה בלבד.**

כתבו פונקציית גיבוב המקבלת מספר id המייצג תעודת זהות (מספר unsigned long בן 9 ספרות), ומספר נוסף M (המייצג גודל של מערך). הפונקציה צריכה להחזיר מספר שלם כפונקציה של מספרי תעודת הזהות לפי החישוב הבא: S % M

כאשר S הוא מכפלת מספרי תעודת הזהות **שהם לא אפס** *ללא ספרת הביקורת (ללא* הספרה האחרונה).

לדוגמא: אם תעודת הזהות היא 142042879 ו – M הוא 27

אז S=1\*4\*2\*4\*2\*8\*7 = 3584 ולכן הפונקציה תחזיר 20.

[כדי להדפיס משתנה מסוג long יש להשתמש בסימן "%ld"]

1. הגדירו מבנה בשם course המייצג קורס המכיל: מספר קורס, ציון (מספר שלם)
2. הגדירו מבנה בשם student המייצג סטודנט המכיל : ת.ז, מערך דינאמי של קורסים, מספר שלם המייצג את כמות הקורסים, ומספר שלם המייצג מיקומו בערימת הציונים (שתוגדר בסעיף 12)
3. כתבו פונקציה student \* newStudent(long id) המקבלת ת.ז ומחזירה מצביע לסטודנט הלומד 0 קורסים. המספר בערימה צריך להיות -1.
4. הגדירו פונקציה בשם void addCourse(student \*s, course c) המקבלת מצביע לסטודנט וקורס, הפונקציה מוסיפה את הקורס לסטודנט המסויים (תוך הגדלת המערך הדינאמי במידה וצריך)
5. הגדירו מבנה node המייצג צומת בעץ חיפוש בינארי, כאשר המידע בצומת הוא מצביע למבנה מסוג סטודנט, והצומת מכיל מצביע לבן שמאלי ומצביע לבן ימני

* **לצורך השימוש בעץ חיפוש בינארי יש להביא את כל הקוד של טיפול בעץ חיפוש בינארי לקובץ נוסף binTree.h ושם לשנות את הקוד שיוכל לטפל בנתונים מסוג student**

1. הגדירו מערך **גלובאלי** בגודל 17 כאשר כל תא במערך הוא מצביע לצומת בעץ חיפוש בינארי שהגדרתם בסעיף 6

#define M 17

BinNode \* students[M];

1. כתבו פונקציה student \* getStudent(long id) המקבלת ת.ז של סטודנט ומחזירה מצביע לסטודנט המבוקש (יש לחפש אותו בטבלת הגיבוב בעץ הבינארי). אם הסטודנט לא קיים, הפונקציה מחזירה NULL
2. כתבו פונקציה double getAverage(stundet \*s) המקבלת מצביע לסטודנט ומחזירה את הממוצע שלו בעזרת מעבר על כל ציוני הקורסים
3. הגדירו מבנה בשם stdAvg המכיל משתנה עשרוני לייצג ממוצע ומשתנה long עבור תעודת זהות
4. הגדירו מבנה המייצג ערימה בשם heapAvg המכילה מבנים מסוג stdAvg. הערימה תהיה ערימת **מקסימום** לפי ערך הממוצע.
5. כתבו פונקציה int heapifyUp(heapAvg \*h, int i) המקבלת את הערימה, ושדה המציין אינדקס במערך של הערימה שאותו יש להעלות כלפי מעלה. הפונקציה מחזירה את המיקום בערימה הסופי אליו האינדקס הגיע. לדוגמא: אם הפונקציה קיבלה כאינדקס את המיקום 9 והעלתה את הממוצע במעלה הערימה עד מיקום 1, הפונקציה תחזיר 1.
   * **הערה חשובה:** לא לשכוח, שהפונקציה heapifyUp וכן הפונקציה heapifyDown משנות את מיקום הממוצעים של סטודנטים בערימה. לכן, לכל שינוי יש לעדכן בחזרה אצל הסטודנט את המיקום החדש
   * **כתבו פונקציה** void updateAvgLocation(int id, int i)

הפונקציה מקבלת תעודת זהות של סטודנט ומיקום חדש של הממוצע שלו בערימה, ומעדכנת אצל מבנה הסטודנט את המיקום החדש. כדי לכתוב פונקציה זו יש להעביר את הפונקציות heapifyUp ו – heapifyDown [בסעיף 13] לקובץ main.cpp

* + אפשרות נוספת (האפשרות המועדפת) היא לכתוב פונקציה void updateAll() בקובץ הראשי שעובר על כל הממוצעים בערימה ומעדכן לכל הסטודנטים את המיקומים (גם כאלה שלא בוצע בהם שינוי)

1. כתבו פונקציה בשם int heapifyDown(heapAvg \*h, int i) המקבלת את הערימה, ושדה המציין אינקדס במערך של הערימה שאותו יש להוריד כלפי מטה. הפונקציה מחזירה את המיקום בערימה הסופי אליו האינדקס הגיע.
2. הגדירו בקובץ main.cpp מצביע לערימה גלובאלית

heapAvg \*h;

ובתוכנית הראשית הקצו לה זיכרון דינאמי בעזרת malloc.

h = (heapAvg \*) malloc(sizeof(heapAvg));

1. כתבו פונקציה בשם int heapInsert(heapAvg \*h, stdAvg a) המקבלת מצביע למבנה המכיל ממוצע ומוסיפה אותו לערימה. הפונקציה מחזירה את המיקום בערימה שאליו נכנס המבנה (לצורך עדכון בפרטי הסטודנט בצומת בעץ הבינארי)
2. כתבו פונקציה void InsertStudent(student \*s) המקבלת מצביע לסטודנט. הפונקציה תבצע את הפעולות הבאות:
   * חישוב הממוצע של הסטודנט (בעזרת הפונקציה בסעיף 9)
   * יצירת מבנה חדש מסוג stdAvg עם פרטי הסטודנט והוספה לערימת הממוצעים (בעזרת הפונקציה מסעיף 15)
   * עדכון בפרטי הסטודנט את מיקום השדה של הממוצע שלו בערימת הממוצעים
   * הכנסה של הסטודנט לטבלת הגיבוב בעזרת פונקציית הגיבוב בסעיף 1. (אחרי שמצאנו את המיקום במערך של הסטודנט, יש לבצע הכנסה שלו כצומת בעץ חיפוש בינארי למניעת התנגשויות)
3. כתבו פונקציה בשם changeGrade המקבלת ת.ז של סטודנט, ומצביע לקורס(בסעיף 2). הפונקציה משנה את הציון של הקורס הזה של הסטודנט (כמובן שהממוצע משתנה, והמיקום בערימה משתנה וכו')
4. כתבו פונקציה בשם void printStudent(student \*s) הפונקציה מקבלת מצביע לסטודנט ומדפיסה את הפרטים שלו, הקורסים שלו, וכן את ממוצע הקורסים שלו בפורמט הבא:

(כדי להדפיס מספר long יש להשתמש בסימון %ld)

----------

Student ID: 123456

Student name: Moshe Cohen

Courses:

Course number: 12111

Course grade: 95

---

Course number: 15225

Course grade: 80

Average: 86.5

----------

1. כתבו פונקציה בשם stdAvg ExtractMax(heapAvg \*h) המוציאה מהערימה את הממוצע הגבוה ביותר (הפונקציה גם מסדרת את שאר הערימה) לא לשכוח לעדכן מיקומים חדשים
2. כתבו פונקציה בשם void milga(). הפונקציה מדפיסה **ומוציאה** את פרטי חמשת הסטודנטים שלהם הממוצעים הגבוהים ביותר. לאחר הוצאת ממוצע מהערימה יש לעדכן אצל הסטודנט את המיקום בערימה ל -1
3. **בתוכנית הראשית:**

* צרו 15 סטודנטים כאשר השתמשו בפונקציה רנדומאלית כדי לייצר ת.ז

100000000 + 2\*3\*4\*5\*6\*7\*rand()%1000000000

**לכל סטודנט** צרו מספר רנדומאלי של קורסים בטווח של 1 עד 8 ולכל קורס מספר קורס רנדומאלי וציון בטווח של 0 עד 100.

**הכניסו** כל סטודנט לטבלת הגיבוב בעזרת הפונקציה בסעיף 15

* **הדפיסו** הסטודנטים ולכל סטודנט את הממוצע שלו
* **הדפיסו** את ערימת הממוצעים
* **בחרו סטודנט באופן אקראי ושנו את אחד מציוני הקורסים שלו ל 0.**
* **הדפיסו** **שוב** את הסטודנטים ולכל סטודנט את הממוצע שלו
* **הדפיסו שוב** את ערימת הממוצעים
* **הדפיסו** את חמשת הסטודנטים הזכאים למילגה
* **הדפיסו** עוד חמישה סטודנטים הזכאים למילגה

1. כתבו פונקציה בשם double courseAvg(int courseID, long stds[], int size) המקבלת מספר קורס, מערך של ת.ז של סטודנטים ואת גודלו ומדפיסה את ממוצע הקורס