מתרגל ממונה על התרגיל: סאג'י חשב, sajy@cs.technnion.ac.il

תאריך ושעת הגשה: 2/5/2022 בשעה 23:55

אופן ההגשה: בזוגות. אין להגיש ביחידים. ניתן להיעזר באתר הקורס למציאת שותפים.

הנחיות כלליות:

- שאלות על התרגיל יש לפרסם באתר הפיאצה של הקורס תחת לשונית "wet_1":
  - האתר: https://piazza.com/technion.ac.il/spring2022/234218
  - נא לקרוא את השאלות של סטודנטים אחרים לפני שמפרסמים שאלה חדשה, למקרה שנשאלה כבר.
- נא לקרוא את המסמך "נהלי הקורס" באתר הקורס. בנוסף, נא לקרוא בעיון את כל ההנחיות בסוף מסמך זה.
- התרגיל מורכב משני חלקים: יבש ורטוב.
  - לאחר קריאת כלל הדרישות, יש לתכנן תחילה את מבני הנתונים על נייר. דבר זה יכול לחסוך לכם זמן רב.
  - לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד בכל דרישות הסיבוכיות בתרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול.
  - את הפתרון שלכם מומלץ לחלק למחלקות שונות שאפשר לממש (ולבדוק!) בהדרגתיות.
  - המלצות לפתרון התרגיל נמצאות באתר הקורס תחת: "Programming Tips Session".
    - המלצות לתכנות במסמך זה אינן מחייבות, אך מומלץ להיעזר בהן.
- העתקת תרגילי בית רטובים תיבדק באמצעות תוכנת בדיקות אוטומטית, המזהה דמיון בין כל העבודות הקיימות במערכת, גם כאלו משנים קודמות. לא ניתן לערער על החלטת התוכנה. התוכנה אינה מבדילה בין מקור להעתק! אנא הימנעו מהסתכלות בקוד שאינו שלכם.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת: barakgahtan@cs.technion.ac.il.

הקדמה:

קבוצת סטודנטים בוגרי הקורס מבנה נתונים החליטו לחפש משרות סטודנט בחברות ההייטק שונות. על מנת לוודא שהם מתקבלים לעבודות עם השכר השווה ביותר, הם החליטו לבנות מערכת שתאפשר להם לאסוף סטטיסטיקות על השכר של עובדים קיימים אחרים בחברות אלו. לכל עובד יש שכר ודרגה, והוא מיוצג ע"י מזהה מספרי ייחודי (לשמירה על הפרטיות שלו). עבור כל חברה, המערכת תתחזק את שווי המניות שלה ואת קבוצת העובדים שמועסקים בה.

חזון הסטודנטים הוא לבנות את המערכת בעזרת חוזים חכמים על רשת הבלוקצ'יין של איתריום, אך לפני שהם עושים זאת, הם החליטו לבנות אותה באמצעות פתרון פשוט באמצעות הכלים העומדים לרשותם לאחר מהקורס.

דרוש מבנה נתונים למימוש הפעולות הבאות:

void* Init()

מאתחלת מבנה נתונים ריק. תחילה אין במערכת חברות הייטק או עובדים.

פרמטרים: אין

ערך החזרה:

מצביע למבנה נתונים ריק או $NULL$ במקרה של כישלון בהקצאת זיכרון.

סיבוכיות זמן: $O(1)$ במקרה הגרוע.

StatusType AddCompany(void *DS, int CompanyID, int Value)

הוספת חברה חדשה עם המזהה CompanyID ושווי התחלתי Value. תחילה החברה אינה מכילה עובדים.

פרמטרים:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה החדשה.
Value	השווי ההתחלתי של החברה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $CompanyID \leq 0$ או $Value \leq 0$.
FAILURE	אם CompanyID הוא מזהה של חברה קיימת במערכת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ k)$ במקרה הגרוע, כאשר k הוא מספר החברות.

StatusType AddEmployee(void *DS, int EmployeeID, int CompanyID, int Salary, int Grade)

הוספת עובד חדש עם מזהה EmployeeID ששייך לחברה CompanyID עם שכר Salary ודרגה Grade.

פרמטרים:

DS	מצביע למבנה הנתונים.
EmployeeID	מזהה העובד שצריך להוסיף.
CompanyID	מזהה החברה של העובד.
Salary	השכר ההתחלתי של העובד.
Grade	הדרגה ההתחלתית של העובד.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $EmployeeID \leq 0$, $CompanyID \leq 0$, $Salary \leq 0$ או $Grade < 0$.
FAILURE	אם קיים כבר עובד עם מזהה EmployeeID או שהחברה עם המזהה CompanyID לא קיימת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ n\ +\ log\ k)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת, ו-k הוא מספר החברות במערכת.

StatusType RemoveEmployee(void *DS, int EmployeeID)

העובד בעל המזהה EmployeeID יוצא לפנסיה, וצריך למחוק אותו מהמערכת.

פרמטרים:

DS	מצביע למבנה הנתונים.
EmployeeID	מזהה העובד שיש למחוק מהמערכת.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$ או $EmployeeID \leq 0$.
FAILURE	אם אין עובד עם מזהה EmployeeID.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ n)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת.

StatusType RemoveCompany(void *DS, int CompanyID)

החברה בעלת המזהה CompanyID פושטת רגל ולכן צריך למחוק אותה מהמערכת.

אם קיימים עובדים בחברה בעת הפשיטה, אזי לא ניתן למחוק את החברה, והיא נשארת במערכת, לכן הפעולה נכשלת.

פרמטרים:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה שיש להסיר מהמערכת.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$ או $CompanyID \leq 0$.
FAILURE	אם אין חברה עם מזהה CompanyID או שהחברה מכילה עובדים.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ k)$ במקרה הגרוע, כאשר k הוא מספר החברות.

StatusType GetCompanyInfo(void *DS, int CompanyID, int *Value, int *NumEmployees)

יש להחזיר את שווי החברה בעלת CompanyID, ומספר העובדים שעובדים בה.

פרמטרים:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה שרוצים את המידע עליה.
Value	מצביע למשתנה שיעודכן לשווי החברה הנוכחי במקרה הצלחה.
NumEmployees	מצביע למשתנה שיעודכן למספר העובדים הנוכחי בחברה במקרה הצלחה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $Value == NULL$, $NumEmployees == NULL$ או $CompanyID \leq 0$.
FAILURE	אם אין חברה עם מזהה CompanyID.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ k)$ במקרה הגרוע, כאשר k הוא מספר החברות.

StatusType GetEmployeeInfo(void *DS, int EmployeeID, int *EmployerID, int *Salary, int *Grade)

יש להחזיר את מזהה המעסיק, השכר והדרגה הנוכחיים לעובד EmployeeID.

פרמטרים:

DS	מצביע למבנה הנתונים.
EmployeeID	מזהה העובד שרוצים את המידע עליו.
EmployerID	מצביע למשתנה שיעודכן למזהה החברה בה העובד מועסק במקרה הצלחה.
Salary	מצביע למשתנה שיעודכן לשכר הנוכחי לעובד במקרה הצלחה.
Grade	מצביע למשתנה שיעודכן לדרגת העובד הנוכחית במקרה הצלחה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון
INVALID_INPUT	אם $DS == NULL$, $EmployerID == NULL$, $Salary == NULL$, $Grade == NULL$ או $EmployeeID \leq 0$.
FAILURE	אם אין עובד עם מזהה EmployeeID.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ n)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת.

StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease)

שווי המניות של החברה בעלת מזהה CompanyID עולה, והוא כעת הערך הקודם בתוספת ValueIncrease.

פרמטרים:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה שיש להעלות לה את השווי.
ValueIncrease	התוספת לשווי החברה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $CompanyID \leq 0$ או $ValueIncrease \leq 0$.
FAILURE	אם אין חברה עם מזהה CompanyID.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ k)$ במקרה הגרוע, כאשר k הוא מספר החברות.

StatusType PromoteEmployee(void *DS, int EmployeeID, int SalaryIncrease, int BumpGrade)

העובד עם מזהה EmployeeID מקבל העלאת שכר בגובה SalaryIncrease.

בנוסף, אם BumpGrade>0, אזי דרגת העובד עולה בדרגה אחת, אחרת נשארת כפי שהיא.

פרמטרים:

- DS מצביע למבנה הנתונים.
- EmployeeID מזהה העובד שקיבל העלאה.
- SalaryIncrease התוספת לשכר העובד.
- BumpGrade העובד עולה דרגה אם פרמטר זה גדול מ-0.

ערך החזרה:

- ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.
- INVALID_INPUT אם $DS == NULL$, $EmployeeID \leq 0$ או $SalaryIncrease \leq 0$.
- FAILURE אם אין עובד עם מזהה EmployeeID.
- SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(log\ n)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת.

StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID)

העובד עם מזהה EmployeeID עובר לחברה אחרת עם מזהה NewCompanyID.

פרמטרים:

- DS מצביע למבנה הנתונים.
- EmployeeID מזהה העובד שעובר חברה.
- NewCompanyID מזהה החברה אליה העובד עובר.

ערך החזרה:

- ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.
- INVALID_INPUT אם $DS == NULL$, $EmployeeID \leq 0$ או $NewCompanyID \leq 0$.
- FAILURE אם אין עובד עם מזהה EmployeeID, אין חברה קיימת עם מזהה NewCompanyID או שהעובד נמצא כבר בחברה זו.
- SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(log\ n\ +\ log\ k)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת ו-k הוא מספר החברות.

StatusType AcquireCompany(void *DS, int AcquirerID, int TargetID, double Factor)

החברה בעלת המזהה AcquirerID רוכשת את החברה TargetID.
כדי שתוכל לבצע את הרכישה, שווי המניות של החברה הרוכשת צריך להיות לפחות פי-10 משווי החברה הנרכשת.
החברה TargetID מפסיקה להתקיים והעובדים שלה כעת עובדים של AcquirerID.
השווי של AcquirerID נהיה סכום שווי שתי החברות לפני הרכישה כפול פקטור Factor, מעוגל לשלם בערך תחתון.
למשל: אם לפני האיחוד היה שווי החברות 10 ו-1007 והיה איחוד עם פקטור 1.66 אז השווי החדש של החברה הוא: 1688 כי: $(10 + 1007) * 1.66 = 1688.22$.

<u>פרמטרים</u>:

DS	מצביע למבנה הנתונים.
AcquirerID	מזהה החברה הרוכשת.
TargetID	מזהה החברה הנרכשת.
Factor	הפקטור שכופלים בו את שווה החברה החדשה.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $AcquirerID \leq 0$, $TargetID \leq 0$ $TargetID == AcquirerID$ או $Factor < 1.00$
FAILURE	אם אין חברה עם מזהה AcquirerID, אין חברה עם מזהה TargetID או שהרכישה אינה יכולה להתבצע.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(log\ k\ +\ n_{AcquirerID}\ +\ n_{TargetID})$ במקרה הגרוע, כאשר k הוא מספר החברות. $n_{AcquirerID}$ ו- $n_{TargetID}$ הם כמות העובדים בחברה הרוכשת והנרכשת, בהתאמה.

StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID)

אם CompanyID>0, הפעולה מחזירה את העובד עם השכר הכי <u>גבוה</u> מבין העובדים בחברה.
אם קיים יותר מעובד אחד עם אותו שכר, מחזירים את זה עם המזהה <u>הנמוך</u> ביותר מבניהם.
אם CompanyID<0, הפעולה תחזיר את העובד עם השכר הכי גבוה (ומזהה קטן ביותר במקרה של שוויון), מבין כל העובדים הקיימים במערכת.

<u>פרמטרים</u>:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה הרצויה.
EmployeeID	מצביע למשתנה שיעודכן למזהה העובד המרוויח הכי הרבה במקרה הצלחה.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $EmployeeID == NULL$ או $CompanyID == 0$.
FAILURE	אם $CompanyID > 0$ ואין חברה עם מזהה זה, או אם. $CompanyID > 0$ ואין עובדים בחברה עם מזהה זה או אם $CompanyID < 0$ ואין עובדים מערכת
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: אם $CompanyID < 0$, אז $O(1)$ במקרה הגרוע. אחרת, $O(\log k)$ במקרה הגרוע, כאשר k הוא מספר החברות.

StatusType GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees)

אם CompanyID>0, הפעולה מחזירה את המזהים של העובדים בחברה ממוינים בסדר יורד לפי שכר העובד.
אם קיים יותר מעובד אחד עם אותו שכר, המזהים מוחזרים בסדר עולה לפי המזהה שלהם.
אם CompanyID<0, הפעולה מחזירה את המזהים של העובדים בכל המערכת באותו הסדר כפי שפורט למעלה.

פרמטרים:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה הרצויה.
Employees	מצביע למערך המזהים ממוינים (לפי הסדר שהוגדר) במקרה הצלחה
NumOfEmployees	מצביע למשתנה שיעודכן למספר המזהים במערך במקרה הצלחה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $Employees == NULL$, $NumOfEmployees == NULL$ או $CompanyID == 0$.
FAILURE	אם $CompanyID > 0$ ואין חברה עם מזהה זה, או אם $CompanyID < 0$ ואין עובדים בחברה עם מזהה זה או אם $CompanyID < 0$ ואין עובדים מערכת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: אם $CompanyID < 0$, אז $O(n)$ במקרה הגרוע, כאשר n הוא מספר העובדים במערכת. אחרת, $O(\log k + n_{CompanyID})$ במקרה הגרוע, כאשר k הוא מספר החברות ו- $n_{CompanyID}$ זה מספר העובדים בחברה.
שימו לב: אתם צריכים להקצות את מערך הפלט בגודל המתאים בעצמכם בעזרת malloc, הוא ישוחרר בקוד שניתן לכם עם free.

StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees)

עבור NumOfCompanies החברות בעלות המזהים הנמוכים ביותר מבין החברות המעסיקות עובדים, הפעולה מכניסה לרשימה את מזהה העובד עם השכר הכי גבוה מבין העובדים בחברה, אם קיים יותר מעובד אחד עם אותו שכר, מחזירים את העובד עם המזהה הנמוך ביותר מבינם.
המזהים של העובדים במערך הפלט יהיו ממוינים בסדר עולה לפי מזהה החברה בה כל אחד עובד.
למשל: נניח ובמערכת 5 חברות: 1,2,3,4,5 כאשר חברות 1,3,5 מעסיקות עובד אחד לפחות. וביקשנו את המרוויחים ביותר עבור 2 החברות הראשונות, אז יוחזר מערך באורך 2 שמכיל את מזהי העובדים המרוויחים ביותר של חברות 1 ו-3 (בסדר הזה).

פרמטרים:

DS	מצביע למבנה הנתונים.
NumOfCompanies	מספר החברות עם מזהים נמוכים ביותר הדרוש.
Employees	מצביע למערך המזהים ממוינים (לפי הסדר שהוגדר) במקרה הצלחה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $DS == NULL$, $Employees == NULL$ או $NumOfEmployees < 1$.
FAILURE	אם מספר החברות עם לפחות עובד אחד קטן מ-NumOfCompanie
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(log\ k + NumOfCompanies)$ במקרה הגרוע, כאשר k הוא מספר החברות ו- NumOfCompanies זה הפרמטר לפונקציה (מספר המזהים שיש להחזיר).
שימו לב: אתם צריכים להקצות את מערך הפלט בגודל המתאים בעצמכם בעזרת malloc, הוא ישוחרר בקוד שניתן לכם עם free.

StatusType GetNumEmployeesMatching(void *DS, int CompanyID, int MinEmployeeID, int MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees)

אם CompanyID>0, הפעולה מחזירה שני מספרים:

1. TotalNumOfEmployees יכיל את מספר העובדים בחברה CompanyID שמזהה העובד שלהם הוא לפחות MinEmployeeID ולכל היותר MaxEmployeeId.
2. מבין עובדים אלו, NumOfEmployees יכיל את מספר העובדים ששכרם לפחות MinSalary ודרגתם לפחות MinGrade.

אם CompanyID<0, הפעולה תחזיר את אותן הדרישות אך מבין כל העובדים במערכת.

ייתכן שמבין העובדים הקיימים (בחברה או בכלל) אין עובדים עם מזהה מתאים, במצב זה יוחזר 0 בשני הפלטים, והוא ייחשב כהצלחה.

למשל: נניח שחברה מעסיקה 3 עובדים עם השלשות: [(1, 10, 3), (2, 5, 7), (3, 5, 3)] של (ID, Salary, Grade).
אז קריאה לפונקציה עם MinEmployeeID=2, MaxEmployeeId=4, MinSalary=3 ו-MinGrade=5 תחזיר ב-TotalNumOfEmployees את 2 וב-NumOfEmployees את 1.

<u>פרמטרים</u>:

DS	מצביע למבנה הנתונים.
CompanyID	מזהה החברה הרצויה.
MinEmployeeID	מזהה העובד המתאים המינימלי.
MaxEmployeeID	מזהה העובד המתאים המקסימלי.
MinSalary	השכר המינימלי של עובד מתאים.
MinGrade	הדרגה המינימלית של עובד מתאים.
TotalNumOfEmployees	מצביע למשתנה שיעודכן למספר העובדים עם מזהה מתאים במקרה הצלחה.
NumOfEmployees	מצביע למשתנה שיעודכן למספר העובדים שבנוסף שכר ודרגתם מתאימה במקרה הצלחה.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון
INVALID_INPUT	אם $DS == NULL$, $TotalNumOfEmployees == NULL$, $NumOfEmployees == NULL$, $CompanyID == 0$, $MinEmployeeID < 0$, $MaxEmployeeID < 0$, $MinSalary < 0$, $MinGrade < 0$ או $MinEmployeeID > MaxEmployeeID$.
FAILURE	אם $CompanyID > 0$ ואין חברה עם מזהה זה, או אם $CompanyID > 0$ ואין עובדים בחברה עם מזהה זה או אם CompanyID<0 ואין עובדים מערכת.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: אם CompanyID<0, אז $O(log\ n\ +\ TotalNumOfEmployees)$ במקרה הגרוע, כאשר n זה מספר העובדים במערכת ו- TotalNumOfEmployees זה מספר העובדים (מכלל העובדים) שהמזהה שלהם מתאים, המוחזר מהפונקציה. אחרת, $O(log\ k\ +\ log\ n_{CompanyID}\ +\ TotalNumOfEmployees)$ במקרה הגרוע, כאשר k הוא מספר החברות, $n_{CompanyID}$ זה מספר העובדים בחברה ו- TotalNumOfEmployees זה מספר העובדים (מהחברה) שהמזהה שלהם מתאים, המוחזר מהפונקציה.

---

void Quit(void **DS)

הפעולה משחררת את המבנה (את כל הזיכרון אותו הקצאתם). בסוף השחרור יש להציב ערך NULL ב-DS.

<u>פרמטרים</u>:

DS	מצביע למבנה הנתונים.

<u>ערך החזרה</u>:

אין

<u>סיבוכיות זמן</u>: $O(n + k)$ במקרה הגרוע, כאשר n הוא מספר העובדים הכולל ו-k הוא מספר החברות.

הערה לגבי Init/Quit:

הקלט של התוכנית עשוי להכיל פקודות לפני Init ו/או אחרי Quit. בשני המקרים, המצביע DS שהפונקציה תקבל יהיה NULL ולכן מחזירים את הערך הדרוש כפי שמצוין בכל פונקציה. ניתן להניח כי קובץ הקלט יכיל את הפעולה Init ואח"כ את הפעולה Quit, עם אפס או יותר פעולות אחרות (שאינן מאלו) לפני האתחול, אחרי השחרור וביניהם. למרות שזה לא ייבדק, תשימו לב שתכנון תקין של מבנה הנתונים אמור לאפשר יצירה "במקביל" של שני מופעים בלתי תלויים (או יותר) של מבנה הנתונים שלכם ועבודה בהם בלי שיפריעו אחד לשני בכלל.

סיבוכיות מקום:

סיבוכיות המקום הדרושה עבור מבנה הנתונים היא $O(n+k)$ במקרה הגרוע, כאשר n הוא מספר העובדים ו-k הוא מספר החברות. כלומר בכל רגע בזמן הריצה, צריכת המקום של מבנה הנתונים תהיה לינארית בסכום מספרי העובדים והחברות במערכת.

סיבוכיות המקום הנדרשת עבור כל פעולה (כלומר, זיכרון "העזר" שכל פעולה משתמשת בו) אינה מצוינת לכל פעולה לחוד, אך אסור לעבור את סיבוכיות המקום הדרושה שמוגדרת לכל המבנה.

ערכי החזרה של הפונקציות:

בכל אחת מהפונקציות (עם ערך חזרה), ערך ההחזרה שיוחזר ייקבע לפי הכלל הבא:

- תחילה, יוחזר INVALID_INPUT אם הקלט אינו תקין.
- אם לא הוחזר INVALID_INPUT:
  - בכל שלב בפונקציה, אם קרתה שגיאת הקצאה/שחרור יש להחזיר ALLOCATION_ERROR. מצב זה אינו צפוי אלא באחד משני מקרים (לרוב): באמת השתמשתם בקלט גדול מאוד ולכן המבנה ניצל את כל הזיכרון במערכת, או שיש זליגת זיכרון בקוד.
  - אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד FAILURE מבלי לשנות את מבנה הנתונים.
- אחרת, יוחזר SUCCESS.

הנחיות:

חלק יבש:

- החלק היבש הווה חלק מהציון על התרגיל כפי שמצוין בנהלי הקורס.
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- הגשת החלק הרטוב מהווה תנאי הכרחי לקבלת ציון על החלק היבש, כלומר, הגשה בה יתקבל אך ורק חלק יבש תגרור ציון 0 על התרגיל כולו.
- יש להכין מסמך הכולל תיאור של מבני הנתונים והאלגוריתמים בהם השתמשתם בצירוף הוכחת סיבוכיות הזמן והמקום שלהם. מבנה מסמך (מומלץ):
  1. הציגו את מבנה הנתונים בצורה כללית:
     - מה הם מבני הנתונים הבסיסיים שמימשתם. (למשל: עץ AVL שתומך בפעולות: x,y,z)
     - ממה מבנה הנתונים הכולל שלכם מורכב ואיך החלקים השונים קשורים אחד לשני. (למשל: עץ בשם m מסוג A עם שדות a,b,c ששדה b בצמתים שלו מצביע על הצומת המתאים בעץ n)
     - רצוי ומומלץ להיעזר בציור בחלק זה.
     - אם חלקים מסוימים משמשים לפעולה ספציפית, להזכיר זאת.
     - חלק זה עומד בפני עצמו וצריך להיות מובן לקורא גם לפני העיון בקוד. אין צורך לתאר את כל הקוד ברמת המשתנים, הפונקציות והמחלקות, אלא ברמה העקרונית.
  2. עבור כל אחת מהפעולות הנדרשות:
     - הסבירו כיצד מימשתם אותה בעזרת מבני הנתונים שתיארתם. במקרה שהפעולה משנה את מבנה הנתונים, להסביר את השינויים שמבצעים על מבני הנתונים שתיארתם. (למשל: מחפשים בעץ m את מפתח k ועושים z, מכניסים צומת לעץ n עם q) (כמה משפטים, כתלות בפעולה)
     - ניתן להתמקד במקרי ההצלחה וכישלון. אין צורך לפרט על קלט לא חוקי, זה ייבדק בחלק הרטוב.
     - הוכיחו את סיבוכיות הזמן של הפעולה.
  3. הוכיחו שמבנה עומד בדרישת סיבוכיות המקום ושכל הפעולות הנדרשות עומדות בסיבוכיות מקום זו.
- החסמים הנתונים בתרגיל הם לא בהכרח הדוקים ולכן יכול להיות שקיים פתרון בסיבוכיות טובה יותר. מספיק להוכיח את החסמים הדרושים בתרגיל.
- רמת פירוט: יש להסביר את כל הפרטים שאינם טריוויאליים ושחשובים לצורך מימוש הפעולות ועמידה בדרישות הסיבוכיות. אין לדון בפרטים טריוויאליים (הפעילו את שיקול דעתכם בקשר לזה, ושאלו את האחראי על התרגיל אם אינכם בטוחים). אין לצטט קטעים מהקוד כתחליף להסבר. אין צורך לפרט אלגוריתמים שנלמדו בכתה. כמו כן, אין צורך להוכיח תוצאות ידועות שנלמדו בכתה, אלא מספיק לציין בבירור לאיזו תוצאה אתם מתכוונים.
- **על חלק זה לא לחרוג מ-8 עמודים.** והכי חשוב **keep it simple!**

חלק רטוב:

- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- אנו ממליצים בחום על מימוש **Object Oriented**, ב**++C**, מימוש כזה יאפשר לכם להגיע לפתרון פשוט וקצר יותר לפונקציות אותן עליכם לממש ויאפשר לכם להכליל בקלות את מבני הנתונים שלכם (זכרו שיש תרגיל רטוב נוסף בהמשך הסמסטר). על מנת לעשות זאת הגדירו מחלקה, נאמר EmployeeManager, וממשו בה את דרישות התרגיל. אח"כ, על מנת לייצר התאמה לממשק ה C ב library1.h, ממשו את library1.cpp באופן הבא:

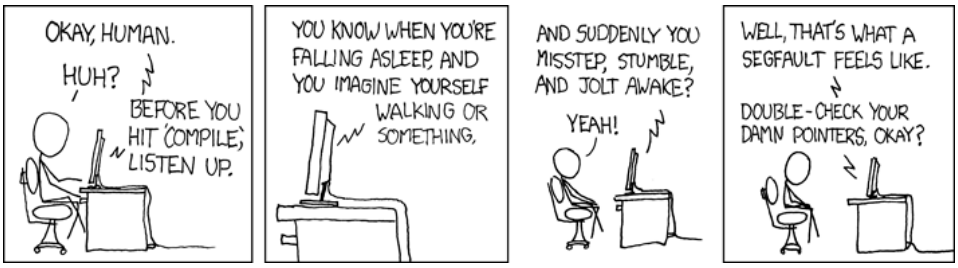
```
#include"library1.h"
#include"EmployeeManager.h"

void* Init() {
        EmployeeManager *DS = new EmployeeManager();
        return (void*)DS;
}
StatusType AddCompany(void *DS, int CompanyID, int Value){
        if (DS == NULL) return INVALID_INPUT;
        return ((EmployeeManager *) DS)-> AddCompany(CompanyID, Value);
}
```

- על הקוד להתקמפל על csl3 באופן הבא:

g++ -std=c++11 -DNDEBUG -Wall *.cpp

עליכם מוטלת האחריות לוודא קומפילציה של התכנית ב+ + g. אם בחרתם לעבוד בקומפיילר אחר, מומלץ לקמפל ב+ + g מידי פעם במהלך העבודה.



הערות נוספות:

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ library1.h
- קראו היטב את הקובץ הנ"ל, לפני תחילת העבודה.
- אין לשנות את הקבצים אשר סופקו כחלק מהתרגיל, **ואין להגיש אותם**.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל **אין להשתמש במבנים של STL** ואין להוריד מבני נתונים מהאינטרנט). **כחלק מתהליך הבדיקה אנו נבצע בדיקה ידנית של הקוד ונוודא שאכן מימשתם את מבני הנתונים שבהם השתמשתם.**
- בפרט, אסור להשתמש ב-std::iterator, std::vector, std::pair, או כל אלגוריתם של STL.
- **ניתן** להשתמש במצביעים חכמים (Smart pointers כמו shared_ptr), בספריית math או בספריית exception.
- חשוב לוודא שאתם מקצים/משחררים זיכרון בצורה נכונה (מומלץ לוודא עם valgrind). לא חייבים לעבוד עם מצביעים חכמים, אך אם אתם מחליטים כן לעשות זאת, לוודא שאתם משתמשים בהם נכון. (תזכרו שהם לא פתרון קסם, למשל, כאשר יוצרים מעגל בהצבעות)
- שגיאות של ALLOCATION_ERROR בד"כ מעידות על זליגה בזיכרון.
- מצורפים לתרגיל קבצי קלט ופלט לדוגמא, ניתן להריץ את התוכנה על הקלט ולהשוות עם הפלט המצורף.
- **שימו לב:** התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.

הגשה:

- **חלק יבש+ חלק רטוב**:
  הגשת התרגיל הנה **אך ורק** אלקטרונית דרך אתר הקורס.
  יש להגיש קובץ **ZIP** שמכיל את הדברים הבאים:
  - בתיקייה הראשית:
    - קבצי ה-Source Files שלכם **(ללא הקבצים שפורסמו).**
    - קובץ PDF אשר מכיל את הפתרון היבש. מומלץ להקליד את החלק הזה אך ניתן להגיש קובץ PDF מבוסס על סריקה של פתרון כתוב בכתב יד. שימו לב כי במקרה של כתב לא קריא, כל החלק השני לא תיבדק.
    - קובץ submissions.txt, המכיל בשורה הראשונה את שם, תעודת הזהות וכתובת הדוא"ל של השותף הראשון ובשורה השנייה את שם, תעודת הזהות וכתובת הדוא"ל של השותף השני. לדוגמה:

John Doe 012345678 doe@cs.technion.ac.il
Henry Taub 123456789 taub@cs.technion.ac.il

- **שימו לב כי אתם מגישים את כל שלושת החלקים הנ"ל.**
- אין להשתמש בפורמט כיווץ אחר (לדוגמה RAR), מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים.
- **יש לוודא שכאשר נכנסים לקובץ הזיפ הקבצים מופיעים מיד בתוכו ולא בתוך תיקיה שבתוך קובץ הזיפ.** עבור הגשה שבה הקבצים יהיו בתוך תיקייה, הבדיקה האוטומטית לא תמצא את הקבצים ולא תוכל לקמפל ולהריץ את הקוד שלכם ולכן תיתן אוטומטית 0.
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב.
- ההגשה האחרונה היא הנחשבת.
- הגשה שלא תעמוד בקריטריונים הנ"ל תפסל ותקנס בנקודות!
  - אחרי שאתם מכינים את ההגשה בקובץ zip מומלץ מאוד לקחת אותה לשרת ולהריץ את הבדיקות שלכם עליה כדי לוודא שאתם מגישים את הקוד שהתכוונתם להגיש בדיוק (ושהוא מתקמפל).

דחיות ואיחורים בהגשה:

- דחיות בתרגיל הבית תינתנה אך ורק לפי תקנון הקורס.
- 5 נקודות יורדו על כל יום איחור בהגשה ללא אישור מראש. באפשרותכם להגיש תרגיל באיחור של עד 5 ימים ללא אישור. תרגיל שיוגש באיחור של יותר מ-5 ימים ללא אישור מראש יקבל 0.
- במקרה של איחור בהגשת התרגיל יש עדיין להגיש את התרגיל אלקטרונית דרך אתר הקורס.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת barakgahtan@cs.technion.ac.il.
  לאחר קבלת אישור במייל על הבקשה, מספר הימים שאושרו לכם נשמר אצלנו. לכן, אין צורך לצרף להגשת התרגיל אישורים נוספים או את שער ההגשה באיחור.

בהצלחה!