בס"ד

בית הספר הגבוה לטכנולוגיה בירושלים

מיני-פרויקט בארגון וניהול קבצים – התשע"א

מורים:
ד"ר חיים דיין
ד"ר משה גולדשטיין, מר עזרא דשט, מר אליעזר גינסבוגר, מר אושרי כהן ומר יואב זילברברג – מכון לב, מכון נווה
ד"ר דן בוכניק וגב' אורית רוזנבליט – מכון לוסטיג
גב' חני נדלר וגב' עדינה מילסטון – מכון טל

הקדמה

ככל מיני-פרוייקט, עיקר הקורס הזה יהיה מעשי-תכנותי. משימתך במשך הסמסטר תהיה בניית מערכת תכנה שמיישמת את ארגון הקבצים הישיר (hash). העבודה העיקרית תתבצע בשפת C++. לא תתקבלנה עבודות כתובות בשפה כלשהי אחרת. סביבת הפיתוח היחידה שבה כל העבודה תתבצע חייבת להיות Visual Studio (יש גרסת חינם שניתן להוריד מהאתרים של חברת Microsoft).

המטרה העיקרית של העבודה במיני-פרויקט תהיה לכתוב חבילה של מחלקות (classes) שמיישמות את ארגון הקבצים הישיר. כלומר, באמצעות חבילה זו יהיה אפשר לבצע פעולות על קבצים בעלי ארגון כנ"ל, כגון יצירה, ביטול, פתיחה, סגירה, כתיבה, קריאה/חיפוש לפי ערך של מפתח וכו'. חבילה זו תשמש ממשק תכנה (API) שבאמצעותו תכניתן (אתה עצמך או מישהו אחר) יוכל להשתמש בכל מה שחבילה זו תעמיד לרשותו. על מנת שתהיה אפשרות להשתמש ב-API כזה, יהיה צורך במימוש חבילת המחלקות הנ"ל כספריה דינאמית (DLL) מלווה בקבצי *.h מתאימים. למימוש כל הנ"ל בשפת C++ תצטרך לדעת: (א) איך להשתמש בפקודות הקלט/פלט של שפת C++, (ב) איך להשתמש בקבצי *.h, (ג) איך יוצרים ספרייה במסגרת מערכת ההפעלה MS-Windows.
מטרת משנה, לא פחות חשובה, תהיה להפגיש אתכם בפעם הראשונה עם תכנות שדורשת חשיבה מערכתית, שדורשת התמודדות עם מורכבות של בעיה.

המיני-פרויקט ימומש בשלבים. שלבים אלו יהיו תלויים זה בזה; כלומר, לא יהיה אפשר לבצע שלב מסוים עד שהשלב הקודם לא הושלם.
(א) לפני ביצוע השלב הראשון של המיני-פרויקט, יהיה שלב מיוחד, שנקרא לו שלב 0 לצורך תרגול הנושאים הבאים, המהווים תנאי ליתר השלבים:

- קלט/פלט בשפת C++
- איך משתמשים בצורה נכונה בקבצי *.h
- איך יוצרים ספרייה סטאטית (LIB).
- איך משתמשים בספריה סטאטית (LIB).
- איך יוצרים ספרייה דינאמית (DLL).
- איך משתמשים בספריה דינאמית (DLL).

(ב) בכל שלבי המיני-פרויקט, פרט לשלב האחרון, עבודתכם תתמקד במימוש חבילת המחלקות הנ"ל.

- כל שלב יממש שכבת תכנה שתשמש בסיס לשלב הבא. בגלל זה, כל שלב יהיה תלוי בזה שקדם לו; כלומר, על מנת לבצע שלב מסוים יהיה צורך בהשלמת השלב הקודם לו.
- על מנת להדגים את כל מה שימומש בכל שלב, תכתבו יישום מסוג console שישמש כ-main. ה-main הזה יהיה תפריט טקסטואלי שבאמצעותו יהיה אפשר להפעיל (ולבדוק) את הפונקציונאליות של המחלקות הנ"ל. ברור שבכל שלב נוסיף אופציות לתפריט, בהתאם לפונקציונאליות שאותו שלב מוסיף לחבילת המחלקות. כל אופציה בתפריט תפעיל פונקציה מתאימה בחבילת המחלקות. פונקציה זו תבצע את משימתה ובסיומה השליטה תחזור ל-main שיציג פלט טקסטואלי מתאים על המסך.
אם התפריט לא יכסה את כל מה שנידרש באותו שלב, תהינה הורדות ציון בהתאם.

למימוש כל השלבים האלה, עליכם להכיר את המחלקות המאפשרות טיפול בקבצים (קלט/פלט) בשפת C++. כחומר התחלתי בנושא, ניתן להסתכל כאן, כאן, כאן וכאן. חומר נוסף משלים אפשר למצוא כאן וכאן.
לקראת הרצת כל אחד מהשלבים האלה,

- תיצור project של Visual Studio מהסוג שיוצר ספריה סטאטית (LIB) בה תהיינה כל המחלקות שכתבת לאותו שלב. באמצעות project מהסוג הזה, תיצור את הספרייה הסטאטית שלך.
- תיצור project נוסף של Visual Studio שבאמצעותו תיווצר תכנית הרצה (exe) שתריץ בזמן ההדגמה במעבדה. project מסוג זה יכלול את ה-main של התכנית, את כל קבצי *.h המתאימים, ואת הספרייה הסטאטית שבנית עם ה-project הראשון.

- בהדגמת כל שלב של המיני-פרויקט, הרצת התכנית חייבת להתבצע *מחוץ לסביבת הפיתוח* (מחוץ ל-Visual Studio); כלומר, תכנית ההרצה (exe) שתיווצר על ידי סביבת הפיתוח תצטרך לרוץ באופן עצמאי לחלוטין, בלחיצת double click או מה-command line של חלון ה-console.

(ג) בשלב האחרון של המיני-פרויקט, התפריט הטקסטואלי יוחלף בממשק גראפי (GUI) שייכתב בשפת C# (או C++). למימוש השלב הזה, עליך להכיר תכנות ממשקים גראפיים (GUI) בשפת C# (אפשר להסתכל באתר עם דוגמאות תכנותיות מאוד שימושיות, בשפת C#: www.java2s.com/tutorial/csharp/catalogcsharp.htm).

הערות חשובות

(א) חובה להסתכל מדי פעם באתר הקורס (האתר הזה) כי ייתכנו שינויים במשך הסמסטר. אם בפועל יהיו שינויים, הם יפורסמו כאן בנוסף להודעה שתפורסם בזמן השיעור ובאמצעות ה-MOODLE.

(ב) יהיה חשבון e-mail ב-gmail שישמש כמעין לוח מודעות של הקורס בו כל תלמיד/תלמידה יוכל/תוכל לשלוח שאלות, הערות והארות; מורי הקורס ישלחו את תשובותיהם פעם בשבוע לכל המשתתפים בקורס (לפי הרשימה המופיעה ב-MOODLE). שם החשבון יפורסם בדף זה וגם ישלח לכל המשתתפים באמצעות ה-MOODLE.

(ג) בקשר להצטרפות לקורס: *לא יהיה ניתן להצטרף* לקורס הזה אחרי מועד הגשת שלב מס' 0, שהוא הבסיס לכל המיני-פרוייקט. הסיבה לכך היא אופיו של הקורס, שכל שלב נבנה על השלבים הקודמים.

(ד) חשוב לציין שזה קורס שמתפתח במשך הסמסטר, ובכוונה לא כל דבר יהיה מוגדר באופן חד-משמעי וסגור לגמרי בכל פרט ופרט. משמעות הדבר, שכמעט בכל שלב של המיני-פרויקט ייתכנו פתרונות שונים, שבאותה מידה יכולים להיות נכונים בהתאם לפירוש שכל אחד (ואחת) יעניק לעניין הנידון. כלומר, כל אחד (וכל אחת) יצטרך להפעיל חשיבה עצמאית ויצירתית על מנת להתמודד עם מצבים שניתן להסתכל עליהם מנקודות מבט שונות ולהגיע בהתאם לפתרונות שמשיגים את אותה המטרה למרות שהם שונים מבחינת מבני הנתונים ומבחינת האלגוריתמים שכל אחד יחליט ליישם.

(ה) התפישה בקורס הזה היא שהשיעור במעבדה מיועד, בין היתר, לדיון ולניתוח של המשימות שעל הפרק. המורה יסביר מה משמעותן של המשימות השונות של אותו שלב, וינחה את הדיון בכיתה תוך עידוד הבאת אלטרנטיבות אפשריות לפתרון תכנותי של אותן משימות. במקרים יוצאי דופן יהיה הסבר על דברים קשורים לשפת התכנות עצמה; בדרך כלל אתם בעצמכם תצטרכו לחפש תשובות לשאלות על פרטים טכניים של שפת התכנות ו/או סביבת הפיתוח – יש אין ספור אתרים באינטרנט, שניתן להיעזר בהם בקשר לעניינים אלה.

נוהלי עבודה

העבודה *חייבת* להיות בזוגות; נוכחותם של שני בני הזוג בזמן השיעור במעבדה ***חובה***.
אם תהיינה בעיות כלשהן במשך הסמסטר, ניתן יהיה לפנות למורי הקורס באמצעות כתובת הדואר האלקטרוני של הקורס *בלבד*.

נוהלי הגשה ובדיקה

בהתאם ללוח הזמנים המפורט כאן, בתום ביצוע כל אחד מהשלבים במיני-פרוייקט,

(א) שמות, תעודות זהות וכתובות דוא"ל של שני בני הזוג חייבים להופיע כהערות בהתחלת כל קובץ הקובץ של התכנית, במיוחד ב-main.

(ב) כל זוג יכין קובץ zip שיכיל את כל המיני-פרויקט במצבו הנוכחי, כולל כל ה-source files בשפת C++ (ובשפת C#, במקרה של השלב האחרון, אם לא תעשה אותו גם ב-C++), כל ה-object files, קובץ הספרייה , קובץ ההרצה (exe) והדו"ח החלקי (ראה בהמשך) המתייחס לאותו שלב/שלבים שמגישים באותו רגע.

(ג) שם הקבוץ zip הנ"ל חייב להיות לפי הפורמאט הבא: FMS5771_nnn_ID1_ID2.zip (nnn – מס' השלב של המיני-פרויקט, IDi – מס' תעודת זהות של בן הזוג i).

(ד) הקובץ zip הנ"ל יוגש לתא ההגשה המתאים שיהיה ב-MOODLE.

שימו לב שעל כל איחור בהגשה יירשם ובסוף הסמסטר זה יכול להשפיע לרעה בציון הסופי. כלומר, כדאי מאוד לא לאחר בהגשה.

בתום השלב מס' 1, בתום אחד מהשלבים האמצעים ובסוף המיני-פרוייקט,

(א) תתבצע הדגמה של המיני-פרויקט במצבו באותו רגע: בזמן שיעור מעבדה שיתקיים מייד אחרי תאריך ההגשה ב-MOODLE, כל זוג ידגים למורה שלהם את הרצת המיני-פרוייקט במצבו הנוכחי באותו שלב.

(ב) ציון ההרצה של אותו שלב יינתן על המקום ע"י המורה, והוא יהיה בין 0 ל-100.

(ג) תתבצע בדיקת התיעוד ואיכות התכנות של אותו שלב באופן לא יסודי במיוחד; כל מטרתה להעיר עליהם על מנת לאפשר תיקון הליקויים כבר בשלבים המוקדמים של העבודה, וכך למנוע מצב שבסוף הסמסטר יתברר שהעבודה לא נעשתה לפי הדרישות והציפיות, מה התבטא בהורדת נקודות בציון הסופי.

(ד) *דו"ח חלקי*: ביחד עם כל מה שתואר בסעיפים (א) עד (ג), מומלץ ביותר לכתוב *דו"ח חלקי* שמתייחס לשלב/שלבים הספציפיים שההרצה שלהם נבדקת באותו רגע במעבדה (להנחיות על כתיבת הדו"ח, ראה כאן). כך המורה יוכל להסתכל ולהעיר הערות שיגרמו לשיפור הדו"ח שבסופו של דבר יוגש בסוף הסמסטר. אם חשבתם על מבני נתונים אלטרנטיביים לפתרון הבעיה הנידונה באותו שלב שאליו הדו"ח החלקי מתייחס, זה המקום להביא אותם ולנתח את היתרונות והחסרונות של כל אחד מהם, ולהסביר למה נבחר מבנה הנתונים שמומש בפועל בתכנית.

הציון של כל שלב (ראה סעיף "ציון" בהמשך) יינתן בסוף הסמסטר בזמן הגשת, הדגמת והגנת המיני-פרויקט כולו. בכל זאת, המורה יוודא שהשלב הוגש ל-MOODLE בזמן ולפי ההנחיות דלעיל.

הגשת השלב האחרון: עם הגשת השלב האחרון ל-MOODLE (כולל הדו"ח הסופי של המיני-פרויקט), תודגם המערכת כולה באופן הבא:

מועד הדגמת/הגנת המיני-פרויקט כולו ייקבע על ידי מדור בחינות כחלק ממועדי א' של הבחינות של סוף הסמסטר. מובן מאליו שלהדגמת/הגנת המיני-פרויקט אין מועד ב'. הרצת המערכת תיבדק באמצעות הממשק הגראפי, תוך הענקת ציון לכל שלב ולמיני-פרויקט כולו לפי מה שכתוב בסעיף "ציון" בהמשך.

כל ההיבטים האחרים של המיני-פרויקט (איכות התכנות, תיעוד ודו"ח סופי) ייבדקו ויינתן בהתאם ציון למיני-פרויקט כולו.

ציון

הציון הסופי ייקבע לפי הטבלה הבאה:

קריטריונים לקביעת הציון

משימה	משקל	הערות
ה-main כיישום Win32 Console (טקסטואלי)	**10%**	
ה-main כיישום חלונאי בעל GUI, כתוב בשפת C# (או ב-C++).	**10% בונוס**	הציון ייקבע לפי איכות תכנון הממשק, מבחינת השימושיות (usability) שהממשק מעניק למערכת, וכו'.
הרצת שלבי המיני-פרויקט באמצעות ה-main הטקסטואלי.	**32%**	התכנית חייבת לבצע את משימותיה בצורה נכונה, בהתאם לדרישות ולמפרטים.
איכות התכנות של המיני-פרויקט כולו	**32%**	איכות התכנות שלך יוערך לפי הקריטריונים הבאים: ▪ מומלץ שהמחלקות תהיינה בנויות תוך ניצול מרבי של היכולות object-oriented של שפת התכנות. ▪ מומלץ שהפונקציות במחלקות, ובתכנית בכלל, תהיינה כתובות לפי סגנון כתיבה "נכון"; כלומר, לפי הקווים המנחים הבאים : ➢ כל פונקציה חייבת להיות באורך סביר; כלומר, לא יותר מעשרות בודדות של שורות לכל היותר. זה יאפשר לקרוא ולהבין אותה בקלות, עד כדי כך שלא יהיה צורך בתיעוד בתוך גוף הפונקציה עצמו. ➢ זה לא תקין מבחינה תכנותית שתהיינה העתקות של קטעי קוד; מן הראוי להגדיר פונקציה ולהשתמש בה (לקרוא לה) כל הפעמים שיהיה צורך. ➢ למשתנים, לפונקציות ולקבועים, חייבים להיות שמות משמעותיים, כך שמשמעותם ותפקידם יהיו מובנים מאליהם, מה שיתרום לקריאות התכנית. ➢ שמות הקבועים ייכתבו באותיות גדולות, לעומת שמות המשתנים והפונקציות שייכתבו באותיות קטנות. ➢ להגדרת קבועים, אל תשתמש ב-#define אלא במשתנים מסוג const. ➢ גוף של פונקציה, ובתוכה גוף של לולאה (for, while או do-while), או גוף של הסתעפות (if-else או switch-case) *חייבים* להיות ב*אינדנטציה* (indentation) יחסית לפתיח של המבנה התחבירי המתאים. ➢ לא מומלץ להשתמש בכל מיני "טריקים" של השפה על מנת לחסוך בכתיבת שורת קוד או במשתנה אחד; ברוב המקרים זה פוגע בקריאות התכנית.

תיעוד של המיני-פרויקט כולו	**16%**	תעד באופן ברור כל פרט חשוב בתכנית. למשל: ▪ בקובץ h מתאים, לכל מחלקה (class), תאר את מטרתה, ואיפה ניתן להשתמש בה. ▪ בקובץ h מתאים, כתוב מפרט (ספסיפיקציה) לכל פונקציה, מייד לפניה; כלומר, כתוב תיאור כללי של הפונקציה, מהם הפרמטרים (הקלטים) ומשמעותם, סוג הערך המוחזר (הפלט) ומשמעותו. למשל: /********** * FUNCTION * *genprime* * PARAMETERS * int – highest possible prime value * RETURN VALUE * A (positive) integer: the highest prime number, * smaller than the integer received as parameter. * MEANING * This functions computes a prime number p, such that * 0 <= p <= PARAMETER * SEE ALSO * *list of names of other functions in your system,* * *related to this function.* **********/ int genprime(int); ▪ לכל משתנה שמשמעותו (או תפקידו) אינה טריוויאלית, כתוב תיעוד קצר ליד הגדרתו. אם ידוע לך על כלי לתיעוד תכנה, תשתמש בו.
דו"ח מסכם של המיני- פרוייקט כולו	**10%**	הדו"ח המסכם יכלול: (1) צרוף הדו"חות החלקיים שנכתבו על כל שלב, במשך הסמסטר. (2) הערות ביקורתיות בקשר לקורס בכללותו (3) הערות ביקורתיות בקשר לסגל ההוראה של הקורס (4) הצעות לשיפור הקורס (5) הצעות להוספת שלב/שלבים למיני-פרויקט. להנחיות מפורטות בקשר לדו"ח, תסתכל כאן. מועד הגשת הדו"ח המסכם יהיה מועד הדגמת המיני-פרויקט כולו.
הציון הכולל	**100%**	הציון הכולל של המיני-פרויקט יינתן אחרי שהמורה יקרא את כל הדו"חות – בבקשה, תנסו להיות מאוד תמציתיים (לא יותר מ-20 עמודים, ב-Arial 12 ורווח כפול).

סיכום הציון הכולל עם פירוט משקלם של כל השלבים	- ה-main כיישום Win32 Console (טקסטואלי) 10 - הרצה, איכות התכנות ותיעוד של כל השלבים 80 - שלב מס' 0 15 - שלב מס' 1 20 - שלב מס' 2 30 - שלב מס' 3 15 - דו"ח מסכם 10 סה"כ ---------- 100 ה-main כממשק גראפי *בונוס* 10% <u>הסבר על הבונוס</u>: נניח שקיבלת 85 מתוך ה-100 של המיני-פרויקט (ללא הממשק הגראפי). אם לא עשית את הממשק הגראפי, הציון הכולל שלך יהיה 85. אם עשית את הממשק הגראפי וקיבלת עבורו 100, תקבל בונוס של 10%; כלומר, זה יוסיף 85*0.1 (= 8.5) לציון הכולל שלך שיעלה ל- 93.5 (כלומר, 93). אם קיבלת 90 עבור הממשק הגראפי, זה יוסיף 85*0.09 (= 7.65) לציון הכולל שלך שיעלה ל-92.65 (כלומר, 92).
הגנה על המיני-פרויקט כולו.	ההגשה הסופית וההגנה על המיני-פרויקט תתקיימנה בתאריך שייקבע על ידי מדור בחינות כחלק ממועדי א' של מבחני סוף הסמסטר. ההגנה תשמש לקביעת ה***<u>ציון הסופי</u>*** של המיני-פרויקט כולו לפי הנוסחה הבאה: (ציון ההגנה) * (הציון הכולל), כאשר *ציון ההגנה* יהיה ערך בין 0.0 ל-1.0 *הציון הכולל* יהיה ערך בין 0 ל-100. לדוגמה: אם הציון הכולל של המיני-פרויקט שלך הוא 90, ובהגנה נותנים לך 1.0, הציון הסופי שלך יהיה גם 90; לעומת זאת, אם בהגנה נותנים לך 0.95, הציון הסופי שלך יהיה 85.5 במקום 90. בקשר לסוג השאלות שייתכן שתישאלנה, תסתכל <u>כאן.</u>

תכנית הסמסטר

בקשר למטלות הקורס לאורך הסמסטר

שבוע	פעילות
1,2,3	הרצאה: התמונה הכוללת של המיני-פרויקט *משימתך* במיני-פרויקט הזה, לבנות מערכת תכנה שמיישמת את ארגון הקבצים הישיר (hash file). על מנת ליישם את המערכת הזאת, נזהה הפשטות מתאימות לארגון קבצים זה. תיאור של עקרונות ארגון הקבצים hash אפשר למצוא כאן מצגת שמבוסס על פרק 12 של המהדורה הרביעית של הספר "Database System Concepts" מאת A. Silberschatz et al.. אנחנו נממש את ארגון הקבצים הישיר לפי המבנה הכללי שניתן למצוא כאן. המרכיבים המבניים וההתנהגותיים שלפיהם אנחנו מציעים לממש את ארגון הקבצים הישיר, יבואו לידי ביטוי כמחלקות בשפת C++, והקשרים המוגדרים ביניהן, שמוצגות כאן וכאן. בכל אופן, לך יהיה חופש להוסיף, לשנות וכו', אפילו להציע מבנה משלך (הסברים עליו ביחד עם הנימוקים המתאימים, יצטרכו להופיע בדו"ח המסכם של המיני-פרויקט). כלומר, אם תוך כדי פיתוח המיני-פרויקט יש לך צורך להגדיר מבני נתונים ומחלקות כלשהם נוספים, או לממש את הדברים בצורה אחרת ממה שמוצע בחוברת הזאת, זה יהיה עניין של החלטות תכנוניות/תכנותיות שלך. המגבלה היחידה לחופש והגמישות האלה היא שכל מה שנידרש בשלבים השונים של המיני-פרויקט יעבוד בדיוק לפי הדרישות, בלי שכל תוספת שלך תפריע להפעלת המיני-פרויקט בצורה הצפויה. כלומר, שם המחלקות ושמות הפונקציות שיופיעו במטלות של שלבי המיני-פרויקט חייבות להישמר אפילו אם אופן המימוש שבחרת שונה ממה שתוכנן על ידי צוות המורים. בשלב הסופי של המיני-פרויקט מתוכנן ליישם את ה-main של התכנית כממשק גרפי. הצעה לתפריט ראשי של ממשק זה ניתן למצוא כאן. במיוחד בשלב זה של המיני-פרויקט, אתם חופשיים לבטא את היצירתיות שלכם בכל מה שנוגע לתכנון ומימוש של הממשק הגראפי, כשההיבט שחייב לעמוד במרכז התכנון והמימוש שלכם היא השימושיות (usability) של המערכת שלכם מבחינת המשתמש. **שלב מס' 0: תשתית הקובץ ברמה הפיזית שלו – הגשה: 07/03/11 – א' אדר א'** הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא כאן
4,5	**שלב מס' 1: תשתית הקובץ ברמה הלוגית שלו – הגשה: 22/03/11 – ט"ז אדר ב'** הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא כאן.
6,7,8	**שלב מס' 2 : מימוש מנגנון ה-hash: חיפוש, קריאה וכתיבה של רשומות לפי ערך המפתח – הגשה: 11/05/11 – ז' אייר** הגדרה מפורטת של משימתך לשלב זה של המיני-פרוייקט אפשר למצוא כאן
9,10	**שלב מס' 3: מחיקה ועדכון של הרשומה הנוכחית – הגשה: 23/05/11 – י"ט אייר** הגדרה מפורטת של משימתך לשלב זה של המיני-פרוייקט אפשר למצוא כאן
11,12,13	**שלב מס' 4: מימוש הממשק הגראפי – הגשה: 13/06/11 – י"א סיון** הגדרה של משימתך לשלב זה של המיני-פרוייקט אפשר למצוא כאן .

בס"ד

תאור מפורט של המשימות לביצוע בשלב מס' 0 של המיני-פרויקט

משימתך בשלב זה של המיני-פרויקט להגדיר וליישם את התשתית (מבנה + פעולות) של קובץ hash ברמה הפיזית שלו כרצף של סקטורים. בשלב הבא נשלים את התשתית בהגדרת מבנה ופעולות לטיפול בקובץ hash מבחינה לוגית.

הערה חשובה:
שים לב שבמיני-פרויקט הזה משתמשים במחרוזות בשני מובנים (וייצוגים) שונים. מחרוזות מאוחסנות ברשומות של קבצי ה-hash כמחרוזות בסגנון שפת C (מערך של תווים שמסתיים ב-'\0'). לעומת זאת, רוב המחרוזות שמתקבלות כפרמטרים בפונקציות השונות של המחלקות שנממש במיני-פרויקט, הינן אובייקטים מסוג המחלקה string של C++. כלומר, ברוב המקרים, ברמת האפליקציה המחרוזות אמורות להיות מהסוג של המחלקה string של C++, וברמת מערכת ניהול קבצי hash שאנו מיישמים, מדובר במחרוזות בסגנון שפת C. לכן, *חייבים* לדאוג להמיר מחרוזות מייצוג אחד לייצוג השני, בשני הכיוונים. ניתן ללמוד על המרות מסוג זה בתכנית שנמצא בקובץ strconvert.zip.

מחלקת PhysicalFile

אובייקט מסוג זה מייצג קובץ ברמה הפיזית שלו. ברמה זו אנו נסתכל על קובץ כרצף של בלוקים ללא ציון מבנה כלשהו אחר. לכן, כל אובייקט מסוג זה יספק את הפונקציונאליות הפיזית הבסיסית של קובץ כרצף של בלוקים. בשלב הבא של המיני-פרויקט נתייחס לקובץ ברמה הלוגית שלו כקובץ hash. רמה זו תיבנה מעל הרמה הפיזית שבונים בשלב הזה.

- **החלק המבני של המחלקה (בהתאם למבנה המפורט שאפשר למצוא כאן וכאן)**

לכל אובייקט מסוג זה חייבים להיות לפחות השדות הבאים (בשלב הבא תהיינה השלמות למחלקה הזאת בכל מה שקשור לרמה הלוגית של הקובץ):

- **opened** : שדה בולאני שישמש כדגל: אם ערכו false, משמעותו שהקובץ עדיין סגור מבחינה פיזית (כלומר, שעדיין לא בוצע popen; אם ערכו true, משמעותו שהוא פתוח).
- **openmode** : שדה מספרי שיעיד על אופן הפתיחה של הקובץ (ראה הסבר בפונקציה popen).
- **Filefl** : אובייקט מסוג fstream המייצג את הקובץ.
- **WorkingDir** : מסלול מלא של התיקיה בה נמצא הקובץ.
- **FileName** : שם הקובץ.
- **FileSize** : סה"כ הבלוקים שיש לקובץ; כלומר, זה מספר שמציין את גודל הקובץ במספר בלוקים .
- **CurrBlock** : אובייקט שמייצג את הבלוק הנוכחי; כלומר, הבלוק בו אנו ממוקמים כרגע. לאובייקט זה יש שני שדות:
  - **Nr** : מספר סידורי של הבלוק, מהתחלת הקובץ.
  - **Buffer** : אובייקט מסוג PhysicalBlock שמשמש כחוצץ שאליו ייקרא וממנו ייכתב פיזית כל בלוק שבקובץ. PhysicalBlock (ראה כאן וכאן) אמור להיות מבנה או מחלקה שמיישמת בלוק ברמה הפיזית הבסיסית.
- **FHBuffer** : אובייקט מסוג PhysicalBlock שמשמש כחוצץ שאליו ייקרא וממנו ייכתב פיזית הבלוק של ה-File Header (ראה כאן וכאן).

- **<u>החלק ההתנהגותי (פונקציונאלי) של המחלקה</u> (ראה <u>כאן</u>):**

<u>הערה כללית חשובה</u>: *כל* תקלה, תוצאה מפעולה כלשהי, תיצור exception מתאים שיטופל על ידי *מנגנון ה-exception handling* של שפת C++ (מנגנון ה-try-throw-catch).

כוונתנו בשלב זה של המיני-פרויקט, להגדיר ולממש את הפונקציות הבאות:

- **PhysicalFile(void) - default constructor**

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מהסוג הזה בלי לקשר לו אף שם של קובץ (כלומר, ערכו של השדה FileName יהיה מחרוזת ריקה), ובלי לקשר לו אף מסלול מלא של תיקיה בה הקובץ אמור להימצא (כלומר, ערכו של השדה WorkingDir יהיה מחרוזת ריקה). השדה opened יאותחל בערך false, ולשדות Filefl,openmode ו-FileSize אין אף ערך משמעותי.

- **PhysicalFile(string & [, string & [, int [, unsigned int]]])**

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מהסוג הזה.

<u>הפרמטר הראשון</u> הוא מחרוזת שהיא <u>שם הקובץ</u> ללא סיומת (הסיומת הסטנדרטית היחידה של שם של קובץ מסוג זה, חייבת להיות המילה hash).

<u>הפרמטר השני</u> הוא מחרוזת שהיא <u>המסלול המלא של התיקייה בה הקובץ אמור להיות מאוחסן</u>. מחרוזת ריקה תציין שהכוונה לתיקייה הנוכחית. אם פרמטר זה לא מופיע, ברירת המחדל תהיה התיקייה הנוכחית.

<u>הפרמטר השלישי</u> הוא <u>קוד</u> שיכול להיות אחד משני ערכים: 1 או 2. הפרמטר הזה קובע אחד משני דברים:

(1) <u>אם ערכו 1</u>, הפונקציה תפעיל את pcreate על מנת ליצור את הקובץ ברמה הפיזית. ברור שבמקרה הזה הקובץ חייב להיות *<u>לא קיים</u>*; אם הוא קיים, ייווצר תקלה (exception) מתאימה.

(2) <u>אם ערכו 2</u>, הפונקציה תפעיל את popen על מנת להפוך את הקובץ לזמין ברמה הפיזית. במקרה זה, הקובץ חייב להיות *<u>קיים</u>*; אם לא, תיווצר תקלה מתאימה.

אם הפרמטר הזה אינו מופיע, ערך ברירת המחדל שלו יהיה 2.

<u>הפרמטר הרביעי</u> הוא מספר שלם וחיובי.

(1) אם הערך של הפרמטר השלישי הוא 1, הפרמטר הזה מציין את <u>גודל הקובץ</u> בבלוקים (לתיאור מפורט של משמעות הפרמטר הזה, ראה את הסעיף המתאים, בפונקציה pcreate, בהמשך).

(2) אם הערך של הפרמטר השלישי הוא 2, הפרמטר הזה מציין את <u>אופן הפתיחה של הקובץ</u> (לתיאור מפורט של משמעות הפרמטר הזה, ראה את הסעיף המתאים, בפונקציה popen, בהמשך).

אם הפרמטר הזה אינו מופיע, וערך הפרמטר השלישי הוא 2, ערך ברירת המחדל שלו יהיה 0 (מבקשים לפתוח את הקובץ לקריאה בלבד).

אם הפרמטר הזה אינו מופיע, וערך הפרמטר השלישי הוא 1, ערך ברירת המחדל שלו יהיה 1000.

- **~PhysicalFile(void)**

תפקידה של הפונקציה הזאת להשמיד אובייקט מהסוג הזה באופן אוטומטי ברגע המתאים, <u>בלי למחוק פיזית את הקובץ</u>. הפונקציה *חייבת* לבצע pclose אם היא מגלה שעוד לא בוצעה. רק אם היא מגלה ש-pdelete בוצעה (ובפועל הקובץ כבר לא קיים), היא לא תבצע pclose.

- **void pcreate(string & [, unsigned int [, string &]])**

תפקידה של פונקציה זו ליצור בפועל את הקובץ.

<u>הפרמטר הראשון</u> הוא מחרוזת שהיא <u>שם הקובץ</u> ללא סיומת (הסיומת הסטנדרטית היחידה של שם של קובץ מסוג זה, חייבת להיות המילה hash).

הפרמטר השני הוא מספר שלם וחיובי שמציין את גודל הקובץ בבלוקים (על מנת ליצור קובץ חייבים לדעת מראש את גודלו!). אם הוא לא מופיע, ערך ברירת המחדל שלו היא 1000 (מבקשים ליצור קובץ בעל 1000 בלוקים).
הפרמטר השלישי הוא מחרוזת שהיא המסלול המלא של התיקייה בה הקובץ יאוחסן. מחרוזת ריקה תציין שהכוונה לתיקייה הנוכחית. אם הפרמטר הזה לא מופיע, ברירת המחדל שלו היא התיקייה הנוכחית.
הפונקציה הזאת תיצור קובץ בתיקיה המוגדרת על ידי הפרמטר השלישי:

- קיומו של קובץ ה-hash ייבדק. לצורך זה, הקובץ ייפתח לקריאה בלבד תוך שימוש בשדה Filefl שהוא מסוג fstream (ראה למעלה).
  - אם הפתיחה ברמת ה-fstream הצליחה, משמעות הדבר שהקובץ קיים; הקובץ ייסגר וייגרם exception מתאים .
  - אם הפתיחה ברמת ה-fstream לא הצליחה, משמעות הדבר שהקובץ לא קיים, ואפשר להמשיך הלאה עם יצירת הקובץ. לצורך זה, הקובץ ייסגר וייפתח שוב (ברמת ה-fstream) לכתיבה בלבד.
- מייד אחרי הפתיחה מחדש, הקובץ ייווצר כרצף של בלוקים שהמבנה שלהם כמבנה הבסיסי ברמה הפיזית, כפי שמתואר כאן, כאן ו-כאן. הבלוקים יהיו ממוספרים בסדר עולה, כאשר המספר הסידורי של הבלוק הראשון הוא 0 והמספר הסידורי של הבלוק האחרון הוא ערכו של הפרמטר השלישי. הפונקציה הזאת תאתחל באפסים בינאריים את אזור הנתונים של כל אחד מהבלוקים בקובץ. הכתיבה של כל אחד מהבלוקים, אחד אחרי השני, יתבצע באמצעות הפונקציה WriteBlock (ראה בהמשך).
- לאחר יצירה ואתחול של כל הבלוקים של הקובץ, הפונקציה תסגור אותו.

שים לב שהפונקציה הזאת חייבת לאתחל גם את כל השדות של האובייקט (ראה למעלה): השדה opened יקבל false, השדה FileName יקבל את הערך של הפרמטר הראשון, השדה FileSize יקבל את הערך של הפרמטר השני, והשדה WorkingDir יקבל את הערך של הפרמטר השלישי.
רמז: בדוגמה fig14_12.cpp שבמצגת של פרק 14 של הספר של Deitel&Deitel, אפשר למצוא דוגמה של יצירת קובץ מאוד דומה למה שהפונקציה pcreate חייבת לבצע.

- **void pdelete(void)**

תפקידה של פונקציה זו למחוק פיזית את הקובץ. מאותו רגע, לאובייקט הזה אין ממש משמעות כי הוא מתייחס לקובץ שכבר לא קיים. בגלל עובדה זו הפונקציה הזאת חייבת "לאפס" את FileName של האובייקט מסוג PhysicalFile; כלומר, שדה זה חייב להפוך למחרוזת ריקה (מומלץ "לאפס" בערכים מתאימים גם את יתר השדות של אותו אובייקט). פונקציה זו תוכל להתבצע רק אם הקובץ סגור (מומלץ לאתחל את השדה opened ב-false).

- **void popen(string & [, int [, string &]])**

הפרמטר הראשון הוא מחרוזת שהיא שם הקובץ hash ללא סיומת.
הפרמטר השני הוא מספר שמשמעותו אופן הפתיחה של הקובץ (0 – קלט; 1 – פלט; 2 – קלט/פלט). שים לב שמשמעותו של "קלט" היא "פתיחה לקריאה בלבד", משמעותו של "פלט" היא "פתיחה לכתיבה בלבד", ומשמעותו של "קלט/פלט" היא "פתיחה לקריאה, כתיבה ועידכון"._הפרמטר הזה אופציונאלי: אם הוא אינו מופיע, ערכו יהיה 0; כלומר, הפתיחה תהיה לקריאה בלבד.
הפרמטר השלישי הוא מחרוזת שהיא המסלול המלא של התיקייה בה הקובץ hash מאוחסן. הפרמטר הזה אופציונאלי: אם הוא אינו מופיע, ערכו יהיה המסלול המלא של התיקייה הנוכחית.
פונקציה זו תבצע את הדברים הבאים:
(א) השדה FileName יקבל את הערך של הפרמטר הראשון.
(ב) השדה openmode יקבל את הערך של הפרמטר השני.
(ד) השדה WorkingDir יקבל את הערך של הפרמטר השלישי.
(ה) קיומו של קובץ ה-hash ייבדק. לצורך זה, הקובץ ייפתח לקריאה בלבד תוך שימוש בשדה Filefl שהוא מסוג fstream (ראה למעלה).
- אם הפתיחה ברמת ה-fstream לא הצליחה, משמעות הדבר שהקובץ לא קיים, מה שיגרום ל-exception מתאים.
- אם הפתיחה ברמת ה-fstream הצליחה, משמעות הדבר שהקובץ קיים ואפשר להמשיך הלאה.

(ו) בלוק ה-File Header ייקרא לתוך FHBufferבאמצעות הפונקציה readFH (ראה בהמשך)
(ו) אם ערך ה-openmode שווה 0, הקובץ יישאר פתוח ומוכן לשימוש. השדה opened יאותחל עם הערך true ו השדה CurrBlock.Nr יקבל את הערך השלילי -1, מה שיעיד שברגע פתיחת הקובץ אין עדיין בלוק נוכחי שנקרא או נכתב, וגם שחוצץ הקלט/פלט (השדה CurrBlock.Buffer) ריק עדיין.
(ז) אם הערך של השדה openmode שונה מ-0, הקובץ ייסגר ברמת ה-fstream על מנת לפתוח אותו מחדש לקריאה ולכתיבה (שים לב שאפילו במקרה שה-openmode יהיה שווה 1, יהיה צורך לקרוא בלוקים לחוצץ על מנת לבצע חיפוש וכתיבה לוגית של רשומה כלשהי ועל מנת לעדכן את ה-File Header ואת הבלוק הנוכחי, בהתאם לפעולות המתבצעות על הקובץ).
(ח) אם הפתיחה מחדש הצליחה, השדה opened יאותחל עם הערך true והשדה CurrBlock.Nr יקבל את הערך השלילי -1, מה שיעיד שברגע פתיחת הקובץ אין עדיין בלוק נוכחי שנקרא או נכתב, וגם שחוצץ הקלט/פלט (השדה CuurBlock.Buffer) ריק עדיין.

- **void pclose(void)**
הפונקציה הזאת (א) תסגור את הקובץ hash; (ב) תיתן לשדה opened את הערך false.
מהרגע שלשדה opened יש הערך false הקובץ לא זמין פיזית לשימוש.

- **void SeekToBlock(unsigned int)**
<u>הפרמטר</u> הוא מספר שלם וחיובי; מספר זה אמור להיות המספר הסידורי של הבלוק המבוקש בקובץ.
תפקידה של פונקציה זו להתמקם בבלוק המבוקש (בלי עדיין לקרוא אותו). כלומר, השדה CurrBlock.Nr יקבל את ערך הפרמטר. אם ערך הפרמטר גדול מ-FileSize, זה יגרום ל-exception מתאים.
<u>הערה 1</u>: כתוצאה מביצוע הפונקציה הזאת, הבלוק המבוקש הופך להיות הבלוק הנוכחי; כלומר, מייד אחרי ביצוע הפונקציה הזאת, יהיה אפשר לקרוא את הבלוק המבוקש או לכתוב לתוך הבלוק המבוקש.
<u>הערה 2</u>: הפונקציה הזאת תוגדר כ-<u>*private*</u>; כלומר, היא תתבצע כפונקצית שרות לפונקציות readBlock ו-writeBlock.

- **void writeBlock(unsigned int)**
תפקידה של פונקציה זו לכתוב מהחוצץ CurrBlock.Buffer לתוך הבלוק שמספרו הסידורי מתקבל כפרמטר.
<u>הערה</u>: מייד אחרי הכתיבה של הבלוק שבחוצץ לתוך הבלוק שמספרו הסידורי מתקבל כפרמטר, השדה CurrBlock.Nr יקודם באחד.

- **void writeBlock(void)**
תפקידה של פונקציה זו לכתוב מהחוצץ CurrBlock.Buffer לתוך הבלוק הנוכחי בקובץ.
<u>הערה</u>: מייד אחרי הכתיבה של הבלוק הנוכחי מהחוצץ, השדה CurrBlock.Nr יקודם באחד.

- **void readBlock(void)**
תפקידה של פונקציה זו לקרוא את הבלוק הנוכחי מהקובץ לתוך החוצץ CurrBlock.Buffer.
<u>הערה</u>: מייד אחרי הקריאה של הבלוק הנוכחי לתוך החוצץ, השדה CurrBlock.Nr יקודם באחד.

- **void readBlock(unsigned int)**
תפקידה של פונקציה זו לקרוא בלוק מהקובץ, שמספרו הסידורי מתקבל כפרמטר, לתוך החוצץ CurrBlock.Buffer.
<u>הערה</u>: מייד אחרי הקריאה של הבלוק לתוך החוצץ, השדה CurrBlock.Nr יקודם באחד.

- **void writeFH(void)**
תפקידה של פונקציה זו לכתוב מהחוצץ FHBuffer לתוך הבלוק מס' 0 בקובץ.

הערה: מייד אחרי הכתיבה של הבלוק מס' 0, ערך השדה CurrBlock.Nr יהיה 1.

- **void readFH(void)**
תפקידה של פונקציה זו לקרוא את הבלוק מס' 0 של הקובץ לתוך החוצץ FHBuffer.
הערה: מייד אחרי הקריאה של הבלוק מס' 0, ערך השדה CurrBlock.Nr יהיה 1.

- **פונקציות למיניהן, לצורך הדגמת המערכת שנבנתה עד השלב הזה**
לצורך הדגמת המערכת למורה האחראי לקבוצת המעבדה שלך, חשוב ביותר להוסיף פונקציות שבאמצעותן יהיה אפשר לבדוק יותר בקלות את תפקוד המערכת על כל היבטיה. אנו משאירים ליצירתיות שלכם להחליט איזה פונקציות צן הראוי שתהיינה בקטגוריה הזאת.

הערה: בשלבים הבאים של המיני-פרוייקט נוסיף מחלקות וגם פונקציות למחלקות קיימות; הכל לפי צורכי הפרויקט. כמובן שגם אתם תוכלו להוסיף פונקציות משלכם בהתאם להחלטותיכם התכנוניות.

בס"ד

תאור מפורט של המשימות לביצוע בשלב מס' 1 של המיני-פרוייקט

משימתכם בשלב זה של המיני-פרוייקט להגדיר וליישם את התשתית (מבנה + פעולות) של קובץ hash ברמה הלוגית על סמך התשתית ברמה הפיזית שממשנו בשלב הקודם של המיני-פרוייקט. בשלב הבא נממש פעולות כתיבה, קריאה וחיפוש של רשומות בקובץ ה-hash על סמך התשתית שאנו בונים בשני השבלים הראשונים האלה.

מחלקת hashfile שמאפיינת קובץ hash ברמה הלוגית; hashfile תהיה תת-מחלקה (sub-class) של PhysicalFile.

- **החלק המבני של המחלקה (בהתאם למבנה המפורט שאפשר למצוא כאן וכאן)**

בשלב הקודם בנינו את המחלקה PhysicalFile שמספקת את הפונקציונאליות הפיזית הבסיסית של הקובץ, כרצף של בלוקים. בשלב הזה של המיני-פרויקט, מעל הרמה הפיזית הבסיסית, אנו נבנה את המחלקה hashfile שתספק את הפונקציונאליות של הרמה הלוגית של קבוץ hash.

מחלקת hashfile יכולה להיות מיושמת בשתי דרכים אפשריות: (א) כתת-מחלקה של PhysicalFile או (ב) כך שהרמה הפיזית של קובץ ה-hash תהיה מיוצגת כשדה מסוג PhysicalFile בתוך המחלקה hashfile. באחריותכם להחליט איזה משתי האפשרויות האלה לממש.
בדו"ח של השלב הזה אתם חייבים לנתח את היתרונות והחסרונות של כל אחת מהן ולנמק את החלטתכם.

לכן, לכל אובייקט מסוג זה חייבים להיות לפחות השדות הבאים (ראה כאן, וכאן):
UserName: שם המשתמש שמבקש לבצע פעולת hcreate או פעולת hopen (ראה בהמשך).
LogicalBuffer – מצביע מסוג LogicalBlock. שדה זה יצביע לחוצץ הפיזי, CurrBlock.Buffer, שהוגדר במחלקה Physical File וישמש כחוצץ בפועל, לכל בלוק בקובץ (פרט ל-File Header שיש לו חוצץ משלו). בזמן ביצוע בנאי המחלקה, שדה זה יאותחל עם הכתובת של השדה CurrBlock.Buffer שבמחלקה PhysicalFile.
updateflag – שדה בוליאני. אם ערכו true, זה יעיד שהקריאה האחרונה שהתבצעה הייתה לצורך עדכון. אם ערכו false, יעיד שהקריאה האחרונה שהתבצע הייתה קריאה רגילה שאינה לצורך עידכון. תפקידו של שדה זה יבוא לידי ביטוי מעשי בשלב מס' 3.
LBuffChanged – שדה בוליאני שיעיד שהבלוק הנוכחי, שבחוצץ שלו כרגע, השתנה מאז ביצוע ה-read האחרון (read היא פונקציה שתמומש בשלב הבא).
LogicalFHBuffer – מצביע מסוג LogicalFileHeader. שדה זה יצביע לחוצץ הפיזי של ה-File Header של הקובץ, FHBuffer, שהוגדר במחלקה Physical File וישמש כחוצץ בפועל, לבלוק מס' 0 של הקובץ. בזמן ביצוע בנאי המחלקה, שדה זה יאותחל עם הכתובת של השדה FHBuffer שבמחלקה PhysicalFile.
CurrRecNrInBuffer – שדה מספרי שלם וחיובי, שערכו הוא המספר הסידורי של הרשומה הנוכחית, מהתחלת החוצץ. תפקידו של שדה זה יבוא לידי ביטוי מעשי בשלבים מס' 2 ו-3.
LHBuffChanged – שדה בוליאני שיעיד שהבלוק של ה-File Header של הקובץ, שבחוצץ שלו כרגע, השתנה מאז ביצוע ה-open של הקובץ.

הערה 1: זה ברור שעל מנת לטפל באופן מסודר בשני השדות הנ"ל, יהיה צורך להגדיר שני struct-ים (או מחלקות) מתאימים, בהתאם למה שתואר כאן וכאן. כדאי שתסתכל בקובץ buffreusexample.zip בו יש דוגמה תכנותית מאוד פשוטה שמדגימה איך לעשות מה שמוצע כאן על מנת לקשר בין החוצץ ברמה פיזית עם החוצץ ברמה הלוגית.
הערה 2: ייתכן שיהיה צורך להגדיר שדות נוספים, בהתאם להחלטות יישום שלכם.

בזמן פתיחה ברמה הלוגית של קובץ קיים (באמצעות hopen) , או בזמן יצירת קובץ חדש ברמה הלוגית (באמצעות hcreate), הבלוק מס' 0 של הקובץ ייקרא/ייכתב באמצעות המימוש של הרמה הפיזית שכבר נעשה בשלב 0 של המיני-פרויקט.

- **<u>החלק ההתנהגותי (פונקציונאלי) של המחלקה (ראה כאן):</u>**

<u>הערה כללית חשובה</u>: *כל* תקלה, תוצאה מפעולה כלשהי, תיצור exception מתאים שיטופל על ידי *מנגנון ה-exception handling* של שפת C++ (מנגנון ה-try-throw-catch).

כוונתנו בשלב זה של המיני-פרוייקט, להגדיר ולממש את הפונקציות הבאות:

- **hashfile(void) - default constructor**

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג hashfile בלי לקשר לו אף שם של קובץ.
כידוע לך, באופן סטנדרטי, הבנאי הזה יקרא ל-default constructor של המחלקה PhysicalFile ומבצע את כל האיתחולים הדרושים ברמה הפיזית.
ברמה הלוגית, הפונקציה הבונה הזאת חייבת לבצע את הפעולות הבאות: (א) איתחול המצביע LogicalFHBuffer עם הכתובת של השדה FHBuffer של PhysicalFile; (ב) איתחול המצביע LogicalBuffer עם הכתובת של השדה CurrBlock.Buffer של PhysicalFile; (ג) איתחול של השדות הבוליאניים LHBUffChanged ו-LBuffChanged עם הערך false; (ד) איתחול השדה UserName במחרוזת ריקה.

- **hashfile(string &, string &, string & [, int [, unsigned int [, unsigned int [, int [, int [, int]]]]]])**

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג hashfile.
- <u>הפרמטר הראשון</u> הוא מחרוזת שתהיה <u>שם הקובץ</u> ללא סיומת.
- <u>הפרמטר השני</u> הוא מחרוזת שהיא <u>שם המשתמש</u> שמבקש לפתוח את הקובץ, או שמבקש ליצור את הקובץ (ואז הופך <u>לבעל הקובץ</u>).
- <u>הפרמטר השלישי</u> הוא מחרוזת שתהיה <u>המסלול המלא של התיקייה בה הקובץ hash יאוחסן</u>. ראה תיאור של הפרמטר המתאים בפונקציה hcreate.
- <u>הפרמטר הרביעי</u> הוא <u>קוד</u> שקובע אחד משני דברים:

(1) אם הקוד הוא 1 הפונקציה תפעיל את hcreate על מנת ליצור ממש את קובץ ה-hash על כל חלקיו והאיתחולים המתאימים (בהתאם למה שתואר כאן, כאן וכאן).
(2) אם הקוד הוא 2 הפונקציה תפעיל את hopen על מנת לפתוח את קובץ ה-hash.
אם הפרמטר אינו מופיע, ברירת המחדל שלו הוא הערך 2.

- <u>הפרמטר החמישי</u> הוא מספר שלם וחיובי שמשמעותו תלויה בערך של הפרמטר הרביעי:

- אם הערך של הפרמטר הרביעי הוא 1, הפרמטר הזה מציין את <u>גודל הקובץ</u> בבלוקים (על מנת ליצור קובץ חייבים לדעת מראש את גודלו!).
במקרה זה, הפונקציה הבונה הזאת חייבת להפעיל את הפונקציה hcreate על מנת לבצע בפועל את יצירת קובץ ה-hash ברמה הפיזית וברמה הלוגית גם יחד.
- אם הערך של הפרמטר הרביעי הוא 2, הפרמטר הזה מציין את <u>אופן הפתיחה של הקובץ</u> (לתיאור מפורט של משמעות הפרמטר הזה, ראה את הסעיף המתאים, בפונקציה popen). במקרה זה, הפונקציה הבונה הזאת חייבת להפעיל את הפונקציה hopen על מנת לבצע בפועל את פתיחת קובץ ה-hash ברמה הפיזית וברמה הלוגית גם יחד.
- אם הפרמטר הזה אינו מופיע, וערך הפרמטר הרביעי הוא 2, ערך ברירת המחדל שלו יהיה 0 (מבקשים לפתוח את הקובץ לקריאה בלבד).
- אם הפרמטר הזה אינו מופיע, וערך הפרמטר הרביעי הוא 1, ערך ברירת המחדל שלו יהיה 1000 (מבקשים ליצור קובץ בעל 1000 בלוקים).

- שים לב שהפרמטרים שישי עד תשיעי רלוונטיים רק לערך 1 של הפרמטר הרביעי:
  - הפרמטר השישי הוא מספר שלם וחיובי שקובע את אורך הרשומה, במספר בתים.
  - יתר הפרמטרים האלה הם בדיוק כפרמטרים השישי עד שמיני של הפונקציה hcreate (ראה בהמשך את תיאור הפרמטרים האלה של הפונקציה hcreate).

מובן מאליו שבשתי הגרסאות של הפונקציה הבונה של המחלקה hashfile, ה-default constructor של מחלקת האב, מחלקת PhysicalFile, יופעל באופן אוטומטי בזמן יצירת אובייקט מסוג hashfile. בדומה למה שתואר ב-default constructor של hashfile דלעיל, גם בפונקציה הבונה הזאת חייבים לאתחל באותה צורה את המצביע LogicalFHBuffer, את המצביע LogicalBuffer, ואת השדות הבוליאניים LHBuffChanged ו-LBuffChanged.

- **~hashfile(void)**

תפקידה של הפונקציה הזאת להשמיד אובייקט מהסוג הזה באופן אוטומטי ברגע המתאים. הפונקציה *חייבת* לבצע hclose אם היא מגלה שעוד לא בוצע.

- **void hcreate(string &, string &, unsigned int, [, string & [, unsigned int [, unsigned int [, string & [, unsigned int]]]]])**

תפקידה של פונקציה זו ליצור קובץ hash, קודם מבחינה פיזית ומייד לאחר מכן תשלים את יצירתו מבחינה לוגית.
הפרמטר הראשון הוא מחרוזת שהיא שם הקובץ, ללא סיומת.
הפרמטר השני הוא מחרוזת שהיא שם של משתמש שיהיה שם בעל הקובץ.
הפרמטר השלישי הוא מספר שלם וחיובי שקובע את אורך הרשומה, במספר בתים.
הפרמטר הרביעי הוא מחרוזת שהיא המסלול המלא של התיקייה בה הקובץ יאוחסן. מחרוזת ריקה תציין שהכוונה לתיקייה הנוכחית. אם הפרמטר הזה אינו מופיע, ברירת המחדל שלו היא מחרוזת ריקה.
הפרמטר החמישי הוא מספר שלם וחיובי שמציין את גודל הקובץ במספר בלוקים; ברירת המחדל הוא 1000.
הפרמטר השישי הוא מספר לא שלילי שקובע את מיקום התחלת המפתח ברשומה; ברירת המחדל היא 0 (הבית הראשון ברשומה).
הפרמטר השביעי הוא מחרוזת שתקבע את טיפוס הנתונים של המפתח בהתאם למה שתואר כאן; ברירת המחדל היא "I".
הפרמטר השמיני הוא מספר שלם וחיובי שקובע את אורך המפתח, במספר בתים. הפרמטר הזה אופציונאלי אם סוג המפתח הוא "I" כי במקרה זה אורך המפתח ידוע מראש והוא 4.
יצירת קובץ ה-hash מתבצעת בשלושה צעדים:

1. יצירת הקובץ מבחינה פיזית באמצעות הפונקציה pcreate.
2. יצירת הקובץ מבחינה לוגית – מייד לאחר יצירת הקובץ מבחינה פיזית, יאותחל בלוק ה-File Header, שהוא הבלוק מס' 0, לפי המבנה המיוחד לו (ראה כאן, כאן וכאן). על מנת לבצע את זה, חייבים לפתוח את הקובץ לקלט/פלט ברמה הפיזית (כלומר, קודם כל הפונקציה popen מופעלת). מייד לאחר מכן, הבלוק מס' 0 ייקרא ל-FHBuffer באמצעות הפונקציה ReadFH, כל השדות של ה-File Header יאותחלו עם ערכיהם של הפרמטרים המתאימים, ולאחר מכן, הבלוק מס' 0 ייכתב בחזרה לקובץ באמצעות הפונקציה WriteFH. שים לב שהשדה NrOfRecsInFile יאותחל ב-0 והשדה CreationDate יקבל את התאריך של היום, שניתן למשוך אותו ממערכת ההפעלה. לאחר איתחול ה-File Header, חייבים לאתחל לוגית את כל הבלוקים של הקבוץ; כלומר, השדות NrOfRecsInBlock ו-NrOfOverflowedRecs של כל אחד מהבלוקים בקובץ חייבים להיות מאותחלים לערך 0.
3. בסופו של התהליך, הקובץ ייסגר מבחינה פיזית (כלומר, תופעל הפונקציה pclose, שבין היתר נותנת לשדה opened את הערך false).

הערה: בסופו של דבר, יהיה קיים קובץ hash מאותחל ברמה הפיזית וגם ברמה הלוגית שלו, אבל במצב לא זמין מבחינה לוגית וגם לא מבחינה פיזית. כלומר, אם מישהו רוצה להשתמש בקובץ, יהיה חייב לבצע hopen על מנת להפוך אותו לזמין מבחינה לוגית וגם מבחינה פיזית.

- **void hdelete(void)**

תפקידה של פונקציה זו למחוק את הקובץ. פונקציה זו תוכל להתבצע רק אם הקובץ סגור (כלומר, אם ערך השדה opened הוא false). המחיקה בפועל של הקובץ תתבצע על ידי הפונקציה pdelete. כדי שלאובייקט הזה יהיה משמעות שוב, חייבים להפעיל את הפונקציה hcreate כדי ליצור ולאתחל קובץ hash חדש.

- **void hopen(string &, string & [, string & [, int]])**

תפקידה של פונקציה זו לפתוח קובץ מבחינה לוגית ומבחינה פיזית גם יחד.

- הפרמטר הראשון הוא מחרוזת שתהיה שם הקובץ (ללא סיומת).
- הפרמטר השני הוא מחרוזת שהיא שם המשתמש שמבקש לפתוח את הקובץ.
- הפרמטר השלישי הוא מחרוזת שתהיה המסלול המלא של התיקייה בה קובץ ה-hash מאוחסן (ראה תיאור הפרמטר השלישי של הפונקציה hcreate).
- הפרמטר הרביעי הוא מספר שמציין את אופן הפתיחה של הקובץ (ראה את הסעיף המתאים, בפונקציה popen).

ובכן, הפתיחה הלוגית של הקובץ מתבצעת כדלהלן:

- הפונקציה popen תופעל על מנת לטפל בהיבט הפיזי של פתיחת הקובץ. כתוצאה מהפעלת popen ובגלל העובדה שהפונקציות הבונות של hashfile מאתחלות את המצביעים LogicalFHBuffer ו-LogicalBuffer, הדבר היחיד שנישאר לפונקציה זו לבצע על מנת להשלים את הפתיחה הלוגית, הוא לאתחל את שני השדות הבולאיניים LHBUffChanged ו- LBuffChanged ב-false והשדה UserName עם הערך של הפרמטר השני.
- לכל משתמש יש הרשאת קריאה. לכן, אם ערכו של השדה openmode שווה ל-0, הקובץ יישאר פתוח ומוכן לשימוש, לקריאה בלבד.
- אם ערכו של השדה openmode שונה מ-0, רק לבעל הקובץ יש הרשאה לפתוח אותו; כלומר,
- אם UserName שונה מ-OwnerName, הקובץ ייסגר ותיווצרexception שיעיד שאין למשתמש מספיק הרשאות לפתיחת הקובץ.
- אם UserName שווה ל-OwnerName, הקובץ יישאר פתוח ומוכן לשימוש בהתאם לערך השדה openmode.

- **void hclose(void)**

תפקידה של פונקציה זו לסגור את הקובץ. על מנת לבצע את סגירת הקובץ, (א) תופעל הפונקציה flush עם פרמטר 2, ו-(ב) תופעל הפונקציה pclose שמבצעת את הסגירה ברמה הפיזית.

- **void flush([int])**

תפקידה של פונקציה זו לגרום לכתיבה פיזית של תוכן החוצץ של הבלוק הנוכחי, של תוכן החוצץ של ה-File Header, או שניהם, בהתאם לערך הפרמטר:

אם ערך הפרמטר 0, הכוונה לחוצץ של ה-File Header בלבד. משמעות הדבר שתופעל הפונקציה writeFH.

אם ערך הפרמטר 1, הכוונה לחוצץ של הבלוק הנוכחי בלבד. משמעות הדבר שתופעל הפונקציה writeBlock.

אם ערך הפרמטר 2, הכוונה לחוצץ של ה-File Header וגם לחוצץ של הבלוק הנוכחי. משמעות הדבר שתופעלנה הפונקציות writeFH ו- writeBlock.

אם הפרמטר לא מופיע, ברירת המחדל של ערכו תהיה 1.

תפקידו של השדה הבוליאני LHBuffChanged הוא להעיד על שינוי בחוצץ של ה-File Header מאז פתיחתו של הקובץ, ותפקידו של השדה הבוליאני LBuffChanged הוא להעיד על שינוי בחוצץ של הבלוק הנוכחי מאז קריאתו של אותו בלוק לתוך החוצץ. אם חוצץ לא השתנה מאז קריאתו של הבלוק, אין צורך לכתוב אותו בחזרה לקובץ, מה שחוסך זמן קלט/פלט. לכן, לפני הפעלת הפונקציה writeFH, ייבדק השדה הבוליאני LHBuffChanged; רק אם הוא true, הפונקציה תופעל. גם לפני הפעלת הפונקציה writeBlock, ייבדק השדה הבוליאני LBuffChanged ורק אם הוא true, הפונקציה תופעל.

הערה: לפונקציה זו יש משמעות רק על קובץ שנפתח לפלט או לקלט/פלט. במקרה של קובץ שנפתח לקלט בלבד, הפונקציה זו תיצור exception מתאים.

- **פונקציות למיניהן, לצורך הדגמת המערכת שנבנתה עד השלב הזה**

לצורך הדגמת המערכת למורה האחראי לקבוצת המעבדה שלך, חשוב ביותר להוסיף פונקציות שבאמצעותן יהיה אפשר לבדוק יותר בקלות את תפקוד המערכת על כל היבטיה. אנו משאירים ליצירתיות שלכם להחליט איזה פונקציות צן הראוי שתהיינה בקטגוריה הזאת.

הערה: מובן מאליו שאתם חופשיים להוסיף מחלקות, שדות ו/או פונקציות משלכם בהתאם להחלטותיכם התכנוניות.

בס"ד

תאור מפורט של המשימות שעליך לבצע בשלב מס' 2 של המיני-פרוייקט

בשלב זה של המיני-פרוייקט תיישם את המנגנון העיקרי של ארגון הקבצים שלנו: חיפוש מיקומן, קריאתן וכתיבתן של רשומות בקובץ מסוג hashfile. לפונקצית ה-hash תפקיד מרכזי במנגנון הזה. לכן, לפני שנדון בשלוש הפונקציות (seek, read ו-write) שעליך לכתוב, נתייחס לנושא ה-hash ואיך ניישם אותו.
אנחנו מספקים לך מחלקה מוכנה מראש, בשם HashValue, שתשמש ליצירה וחישוב של ערכי hash על סמך מספר ראשוני (prime number) מסוים. לפרטים על המחלקה הזאת ראה את HashValue.h ואת HashValue.cpp.
על מנת להשתמש באובייקטים מסוג HashValue:
א. אתה חייב להוסיף את הגדרת המחלקה הזאת לתכנית באמצעות #include:

```
#include “HashValue.h”
```

ב. בתוך התכנית שלך, במקום שתרצה (בתוך הגדרת המחלקה hashfile, למשל), אתה חייב להגדיר משתנה מסוג HashValue באמצעות הפונקציה הבונה
HashValue(highest_hash_val);

חשוב מאוד לשים לב שהערך שהפונקציה הבונה מעניקה לשדה Hvalmax, באובייקט שהיא יוצרת, הוא המספר הראשוני הקרוב ביותר (מלמעלה) לערך שהתקבל בפרמטר highest_hash_val.
אם ברצונך לראות איך משתמשים במחלקה הזאת, ראה את התכנית HashTest.cpp שבתוך הקובץ HashTest.zip. ניתן להוריד אותו, לקמפל ב- Visual Studio ולהריץ את התכנית HashTest.exe בחלון של Command Prompt.
במחלקה HashValue מוגדרות עשר פונקציות hash; לכל אחת מהן הענקנו קוד (HashFuncID) המייצג אותה:

```
// Hash Function ID codes
enum HashFunclDcode {
	DUMMYfunc = -1,
	MODHfunc = 0,
	MULTHfunc,
	RSHfunc,
	JSHfunc,
	PJWHfunc,
	ELFHfunc,
	BKDRHfunc,
	SDBMHfunc,
	DJBHfunc,
	APHfunc
};
```

על מנת להפעיל אותן, הוגדרו הפונקציות הבאות:

```
int HashFunction(int HashFuncID, const char* key);
int HashFunction(int HashFuncID, int key);
```

בפונקציות האלו, פונקצית ה-hash המיוצגת על ידי הערך של HashFuncID, מופעלת על ערך המפתח (הערך של key). הפונקציה מחשבת ומחזירה את ערך ה-hash המתאים למפתח, על סמך המספר הראשוני שהוא ערך השדה Hvalmax של האובייקט.

```
int HashFunction(int HashFuncID, const char* key, int highest_hash_val);
int HashFunction(int HashFuncID, int key,        int highest_hash_val);
```

ההבדל בין הפונקציות האלו לבין הקודמות, הוא שניתן לאתחל מחדש את השדה Hvalmax עם מספר ראשוני חדש על סמך הערך של הפרמטר highest_hash_val. לאחר מכן, יהיה ניתן לחשב את ערך ה-hash על סמך המספר הראשוני החדש.

ידוע לך שבעת יצירת קובץ hashfile (באמצעות הפונקציה hcreate) נקבע מספר הבלוקים שתכנית היישום מבקשת להקצות לשטח הנתונים הראשי של הקובץ. בשלב מס' 1 של המיני-פרויקט לא ביקשנו שמספר הבלוקים יהיה מספר ראשוני, למרות שידוע לנו שפונקציות hash פועלות בצורה מיטבית כאשר חישוב ערכי הפונקציה מבוסס על מספר ראשוני.
זה המקום להכניס שני תיקונים הכרחיים לפונקציה hcreate:

- חייבים להקצות מספר ראשוני של בלוקים לשטח הנתונים של הקובץ, על סמך ערך הפרמטר שמקבלים מתכנית היישום:
  1. יוצרים אובייקט מסוג HashValue עם הפונקציה הבונה הנ"ל, תוך כדי העברת הערך שקיבלנו מתכנית היישום ל-highest_hash_val.
  2. לאחר יצירת האובייקט מסוג HashValue, ערך השדה Hvalmax הוא מספר הבלוקים בפועל שצריכים להקצות.
  3. באמצעות הפונקציה maxHval() מהמחלקה Hashvalue ניתן לקבל את הערך של השדה Hvalmax; זה מספר הבלוקים שבפועל hcreate תקצה.
- חייבים להוסיף פרמטר חובה נוסף, והוא הקוד המזהה של פונקצית ה-hash (HashFuncID) שלפיה הקובץ נבנה (לפיה רשומות מוכנסות לקובץ, וחיפושי רשומות מתבצעים). זה ברור גם שיהיה צורך בהגדרת שדה נוסף ב-File Header, ברמה הלוגית שלו, שיכיל את הקוד המזהה של פונקצית ה-hash הקשורה לקובץ (אותו פרמטר חדש שערכו יועבר ל-hcreate). ברור שגם לפונקציה הבונה של hashfile חייבים להוסיף פרמטר כזה.

ובכן, <u>*משימתך*</u> בשלב זה של המיני-פרוייקט, לכתוב את הפונקציות הבאות (במחלקת hashfile, כמובן):

- **void seek(string &)**
  void seek(char *)
  void seek(int)
  <u>ערך הפרמטר</u> הוא ערך המפתח שעליו תופעל פונקצית ה-hash.
  <u>תפקידה של פונקציה זו</u> לאתר מקום בקובץ, מתאים לערך המפתח שמתקבל כפרמטר.
  <u>תפקידה של פונקציה זו יתבצע כדלהלן</u>:
  ערך ה-hash המתאים למפתח שמתקבל כפרמטר ולמקם את ה-"מצביע" לבלוק הנוכחי (ה-CurrBlock.Nr) למספר הסידורי של הבלוק המתאים לערך המפתח שמתקבל כפרמטר, ולקרוא פיזית את הבלוק הזה לחוצץ.
  א. פונקצית הhash- (שהקוד המזהה שלה מאוחסן ברמה הלוגית של ה-File Header) תופעל על ערך המפתח המתקבל כפרמטר. ערך תוצאת החישוב של פונקצית ה-hash יהיה מספר סידורי של הבלוק המתאים לאותו ערך של מפתח.
  שים לב שחלק מפונקציות ה-hash המוגדרות במחלקה HashValue, מקבלות ערך מספרי כמפתח ואחרות מקבלות מחרוזת כמפתח. במקרים שמדובר במחרוזת, פונקציות אלו מצפות לקבל מחרוזת בסגנון שפת C (מערך של תווים שמסתיים ב-'\0'). לכן, במקרה שהפרמטר שמועבר ל-seek הוא מסוג string &, *חייבים* להמיר אותו למחרוזת בסגנון שפת C לפני שמעבירים אותו כפרמטר לפונקצית ה-hash; שים לב שמפתח בתוך רשומה, אם הוא מחרוזת, גם יהיה מחרוזת בסגנון שפת C.
  ב. <u>אם ערך תוצאת החישוב של פונקצית הhash- שווה למספר הסידורי של הבלוק הנוכחי</u>, נשארים באותו בלוק.

ג. אם ערך תוצאת החישוב של פונקצית הhash- שונה מהמספר הסידורי של הבלוק הנוכחי, מבצעים את שני הדברים הבאים:
- אם הבלוק שנמצא כרגע בחוצץ עבר שינוי מאז קריאתו, יוחזר למקומו בקובץ באמצעות הפונקציה flush.
- הבלוק שמספרו הסידורי חושב על ידי פונקצית ה-hash, נקרא לתוך החוצץ והופך להיות הבלוק הנוכחי.

ד סורקים את הבלוק שבחוצץ על מנת לחפש רשומה שערך המפתח שלה שווה לערך הפרמטר הראשון.

ה אם מתברר שהרשומה לא נמצאת בבלוק הזה,
1. אם הערך של NrOfOverflowedRecs שווה ל-0, משמעות הדבר שהחיפוש לא היה מוצלח; אי מציאת הרשומה המבוקשת יגרום לאי-הצלחתה של הפעלת הפונקציה הזאת, מה שיצור exception.
2. אם הערך של NrOfOverflowedRecs שונה מ-0, משמעות הדבר שחייבים להמשיך לחפש בבלוקים הבאים:
   - עוברים לבלוק הבא וקוראים את הבלוק לתוך החוצץ, והוא הופך להיות הבלוק הנוכחי.
   - מבצעים סריקה כמו ב-ד', תוך ספירת הרשומות בעלות אותו ערך hash שחושב על ערך המפתח שקיבלנו כפרמטר.
   - אם לא נמצאה רשומה בעלת מפתח שווה לערך הפרמטר, עוברים לבלוק הבא וחוזרים על אותה סריקה וספירה. שים לב שכאשר מגיעים לבלוק האחרון של הקובץ, החיפוש ממשיך בצורה מעגלית; כלומר, הבלוק הבא הוא הבלוק הראשון (הבלוק מס' 1).
3. החיפוש יסתיים ללא הצלחה כאשר מתברר שעברנו על כל הרשומות בעלות אותו ערך hash, בלי למצוא את הרשומה המבוקשת
4. החיפוש יסתיים בהצלחה כאשר נמצאה בבלוק הנוכחי רשומה בעלת מפתח שווה לערך הפרמטר הראשון. במקרה זה מבצעים כל מה שנאמר ב-ו'.

ו אם נמצאה רשומה בעלת מפתח שווה לערך הפרמטר הראשון,
1. הבלוק בו נמצאה הרשומה הופך להיות הבלוק הנוכחי (מה שבא לידי ביטוי בעדכון השדה CurrBlock.Nr)
2. אותה רשומה הופכת להיות הרשומה הנוכחית (מה שבא לידי ביטוי בעדכון השדה CurrRecNrInBuffer)

ז אי-הצלחתה של הפונקציה הזאת תיצור exception מתאים.

בשני המקרים, מקרה של מציאת או אי-מציאת הרשומה המבוקשת, הבלוק הנוכחי יהיה האחרון שנקרא לתוך החוצץ.

הערה 1: זה ברור, מהתיאור דלעיל, שתפקידה של פונקציה זו להכין את התשתית לביצוע קריאה (כתיבה) לוגית של רשומה על ידי פעולת read (write).

הערה 2: הפונקציה הזאת תוגדר כ-*private*; כלומר, היא תתבצע כפונקצית שרות לפונקציות read ו-write.

הערה 3: בדוגמה התכנותית strconvert ניתן ללמוד איך ממירים ממחרוזות של C++ למחרוזות בסגנון C.

- **void read(string &, char * [, int])**
  void read(char*, char* [, int])
  void read(int, char*, [int])

ערך הפרמטר הראשון הוא ערך המפתח של הרשומה שתכנית היישום מבקשת לקרא מהקובץ.

ערך הפרמטר השני הוא מצביע לרשומה (בזיכרון של תכנית היישום, כמובן) שמתאימה לרשומות שמאוחסנות בקובץ.

ערך הפרמטר השלישי הוא מספר המסמל סוג הקריאה (0 – קריאה רגילה, 1 – קריאה לצורך עידכון); כלומר, אם ערך הפרמטר הזה 0, משמעותו קריאה בלי כוונת עדכון; אם ערך הפרמטר הזה 1, משמעותו קריאה לצורך עדכון. כמובן שבקשה כזאת לקובץ שנפתח לכתיבה, היא שגיאה שתיצור exception.

ברירת המחדל של הפרמטר הזה היא 0.

תפקידה של פונקציה זו לקרא רשומה מקובץ ה-hashfile לתוך הכתובת שהתקבלה כפרמטר שני.

תפקידה של פונקציה זו יתבצע כדלהלן

א. קוראים לפונקצית seek על מנת להתמקם בבלוק בשטח הנתונים בו הרשומה המבוקשת נמצאת. בסוף ביצועה של פונקצית ה-seek, הבלוק שבתוכו נמצאת הרשומה המבוקשת נמצא בחוצץ והשדה CurrRecNrInBuffer "מצביע" לרשומה המבוקשת בחוצץ.
   - אותה רשומה מועתקת (באמצעות הפונקציה memncpy) לכתובת שקיבלנו כפרמטר שני.
   - אם ערך הפרמטר השלישי הוא 1, השדה הבוליאני forupdate יקבל ערך true; אם לא, הוא יקבל ערך false.

במקרה זה ה-read התבצע בצורה מוצלחת.

ב. אי מציאת הרשומה המבוקשת על ידי seek יגרום לאי-הצלחתה של הפעלת הפונקציה הזאת, מה שיצור exception מתאים.

- **void write(string &, char*)**
  void write(char*, char)
  void write(int, char*)

  ערך הפרמטר הראשון הוא ערך המפתח של הרשומה שתכנית היישום מבקשת לכתוב לתוך הקובץ.
  ערך הפרמטר השני הוא מצביע לרשומה שתכנית היישום מבקשת לכתוב לתוך הקובץ.
  תפקידה של פונקציה זו לכתוב רשומה, שכתובתה התקבלה כפרמטר שני, לתוך קובץ ה-hashfile.
  תפקידה של פונקציה זו יתבצע כדלהלן
  קוראים לפונקצית seek תוך העברת ערך הפרמטר הראשון כמפתח חיפוש.
  - אם נמצאה רשומה שערך המפתח שלה שווה לערך הפרמטר הראשון, תיווצר exception מתאים.
  - אם לא נמצאה רשומה כזאת, אנחנו ממוקמים בבלוק שאליו תועתק הרשומה שמצביע אליה התקבל כפרמטר שני.
    - אם הבלוק הזה לא מלא, הרשומה תועתק מייד אחרי הרשומה האחרונה שבאותו בלוק.
    - אם הבלוק הזה מלא, עוברים סדרתית מבלוק לבלוק עד שיימצא בלוק בו יש מקום להכניס רשומה חדשה מייד אחרי הרשומה האחרונה שבאותו בלוק, והרשומה תועתק לשם. החיפוש הסדרתי הזה, בלוק אחרי בלוק, הוא מעגלי; כלומר, אם מגיעים לסוף הקובץ ממשיכים בבלוק הראשון, עד שמגיעים בחזרה לבלוק שבו seek עצרה.
  - אם ה-write הצליח, זה אומר שהוספה בהצלחה רשומה לחוצץ; משמעות הדבר שהחוצץ השתנה והשדות NrOfRecsInBlock ו- NrOfRecsInFile, LBuffChanged ו-LHBuffChanged, חייבים לעיד על כך. בנוסף לזה, אם הרשומה הוספה לבלוק שאינו הבלוק שהמספר הסידורי שלו חושב על ידי פונקצית ה-hash, חייבים לעדכן בהתאם את השדה NrOfOverflowedRecs של הבלוק שהמספר הסידורי שלו הוא זה שחושב על ידי פונקציית ה-hash.

- **השוואה בין שיטות hash שונות**

  אחרי כתיבת שלוש הפונקציות הנ"ל, אתה מתבקש ליצור מספר קבצי hash, כל אחד לכל אחת מפונקציות ה-hash שדלעיל.
  משימתך:
  - להכניס את אותם הנתונים בדיוק לכל הקבצים (למשל, כ-30 רשומות) ולהשוות בין הקבצים:
  - מבחינת פיזור הרשומות בין הבלוקים של כל קובץ.
  - מבחינת יעילות ביצוען של הפעולות השונות (seek, read, write).
  - מה היא פונקצית ה-hash הטובה יותר; כלומר, זאת שגורמת לחיפוש יותר יעיל, יותר קרוב ל-O(1).

- **פונקציות למיניהן, לצורך הדגמת המערכת שנבנתה עד השלב הזה**

  לצורך הדגמת המערכת למורה האחראי לקבוצת המעבדה שלך, חשוב ביותר להוסיף פונקציות שבאמצעותן יהיה אפשר לבדוק יותר בקלות את תפקוד המערכת על כל היבטיה. אנו משאירים ליצירתיות שלכם להחליט איזה פונקציות צן הראוי שתהיינה בקטגוריה הזאת.

בס"ד

תאור מפורט של המשימות שעליך לבצע בשלב מס' 3 של המיני-פרוייקט

משימתך בשלב זה של המיני-פרוייקט, ליישם שלוש פונקציות נוספות של ארגון הקבצים hash (במחלקת hashfile): update, delrec, ו-updateoff.

לשדה הבוליאני updateflag יש תפקיד מכריע בשלושת הפונקציות הנ"ל:

- אם ערכו true, משמעות הדבר
  (1) שהפעולה האחרונה שהתבצעה על הקובץ הייתה קריאה לצורך עדכון, והרשומה שנקראה נעולה ועדיין העדכון שלה לא התבצע.
  (2) שהפעולות היחידות שניתן לבצע במצב זה הן update, delete ו-updateoff, שמשחררות את הרשומה מנעילתה.
- אם ערכו false, משמעות הדבר
  (1) שאף רשומה לא נעולה
  (2) שניתן לבצע כל פעולה שהי על רשומה כלשהי בהתאם לאופן הפתיחה של הקובץ.

(א) ישום הפונקציות delrec, update ו-updateoff במחלקת hashfile

- **void delrec(void)**

תפקידה של פונקציה זו לבטל/למחוק את הרשומה הנוכחית בקובץ. כעקרון, יש שתי אפשרויות לביטול רשומה: (א) ביטול "לוגי", כלומר באמצעות סימון הרשומה כמבוטלת, בלי למחוק אותה ממש; (ב) ביטול "פיזי", כלומר "דחיסת" הבלוק הנוכחי.
פונקציה זו תיישם ביטול "פיזי" של רשומה, כמתואר לעיל. כמובן שלאחר הביטול הפיזי של הרשומה, חייבים גם להחסיר באחד את מונה הרשומות שבבלוק וכו'.

הערות:

- תנאי הכרחי לביצועה של פונקציה זו הוא שמיד לפניה התבצעה read לצורך עדכון שמטרתה העיקרית לנעול את הרשומה עד ביצוע העדכון (או עד ביטול העדכון באמצעות updateoff– ראה בהמשך); אם לא, זאת שגיאה שתתבטא ביצירת exception מתאים.
- אחרי ביצוע הביטול, השדה CurrRecNrInBuffer שומר על ערכו.
- אחרי ביצוע הביטול, גם השדות NrOfRecsInBlock, NrOfRecsInFile, והשדות LBuffChanged ו-LHBuffChanged, חייבים להעיד על השינוי.
- פונקציה זו חייבת גם לעדכן בהתאם את NrOfOverflowedRecs בבלוק המתאים לערך המפתח של הרשומה המבוטלת (בהתאם לפונקציה hash).
- כמובן שבקשה כזאת לקובץ שנפתח לקריאה (או לכתיבה) בלבד היא שגיאה שתתבטא ביצירת exception מתאים.

- **void update(char)**

תפקידה של פונקציה זו לכתוב רשומה (שכתובתה התקבלה כפרמטר) לתוך קובץ ה-hashfile, במקום הרשומה הנוכחית (זאת שקודם לכן נקראה ע"י read לצורך עדכון).
ערך הפרמטר הוא מצביע לרשומה המעודכנת שתכנית היישום מבקשת לכתוב.

הערות:

- תנאי הכרחי לביצועה של פונקציה זו הוא שמיד לפניה התבצעה read לצורך עדכון שמטרתה העיקרית לנעול את הרשומה עד ביצוע העדכון (או עד ביטול העדכון באמצעות updateoff– ראה בהמשך); אם לא, זאת שגיאה שתתבטא ביצירת exception מתאים.
- במקרה שערך המפתח של הרשומה המעודכנת (שמצביע אליה התקבל כפרמטר), שווה לערך המפתח ברשומה הנוכחית, הפונקציה הזאת תעתיק אותה על הרשומה הנוכחית, ותשחרר את נעילתה תוך קריאה לפונקציה updateoff

- במקרה שערך המפתח של הרשומה המעודכנת (שמצביע אליה התקבל כפרמטר), שונה מערך המפתח ברשומה הנוכחית, זה מעיד על ניסיון עדכון הערך של המפתח, מה שנחשב לשגיאה. לכן, במקרה כזה הפונקציה הזאת לא תבצע שום עדכון ותקרא ל-updateoff על מנת לשחרר את נעילת הרשומה הנוכחית ותיצור exception מתאים.
- אחרי ביצועה של פונקציה זו, הרשומה הנוכחית ממשיכה להיות זאת שעודכנה; כלומר, השדה CurrRecNrInBuffer שומר על ערכו.
- אחרי ביצועה של פונקציה זו, גם השדה LBuffChanged יעיד על שהחוצץ השתנה.
- כמובן שבקשה כזאת לקובץ שנפתח לקריאה (או לכתיבה) בלבד היא שגיאה שתתבטא ביצירת exception מתאים.

- **void updateoff(void)**

תפקידה של פונקציה זו לשחרר את הרשומה הנוכחית מנעילתה; כלומר, מטרתה לבטל את המצב שנוצר כתוצאת הביצוע של read "לצורך עדכון".

הערות:

- פונקציה זו תוכל להתבצע רק אם הרשומה הנוכחית נעולה (כלומר, נקראה קודם לכן לעדכון). אם זה לא המצב, תיווצר exception מתאים.
- כמובן שבקשה כזאת לקובץ שנפתח לקריאה (או לכתיבה) בלבד היא שגיאה, מה שיתבטא ביצירת exception מתאים.
- אחרי ביצועה של פונקציה זו, הרשומה הנוכחית ממשיכה להיות זאת שהייתה לפני ביצועה.

- **פונקציות למיניהן, לצורך הדגמת המערכת שנבנתה עד השלב הזה**

לצורך הדגמת המערכת למורה האחראי לקבוצת המעבדה שלך, חשוב ביותר להוסיף פונקציות שבאמצעותן יהיה אפשר לבדוק יותר בקלות את תפקוד המערכת על כל היבטיה. אנו משאירים ליצירתיות שלכם להחליט איזה פונקציות צן הראוי שתהיינה בקטגוריה הזאת.

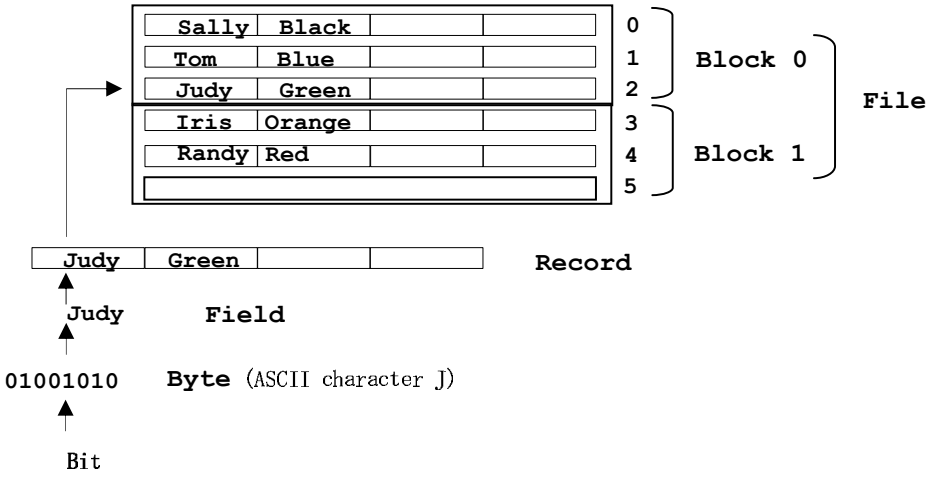
בס"ד

תאור של המשימות שעליך לבצע בשלב מס' 4 של המיני-פרוייקט

משימתך בשלב זה של המיני-פרוייקט, ליישם ממשק גראפי שישמש כ-main של מערכת לניהול קבצי hash שלכם.

במיוחד בשלב זה של המיני-פרויקט, אתם חופשיים לבטא את היצירתיות שלכם בכל מה שנוגע לתכנון ומימוש של הממשק הגראפי, כשההיבט שחייב לעמוד במרכז התכנון והמימוש שלכם היא השימושיות (usability) של המערכת שלכם מבחינת המשתמש.
הצעה לתפריט ראשי של הממשק הגראפי ניתן למצוא כאן.

היררכיה מבנית של קובץ



נקודות מבט שונות בקשר למבנה של קובץ

1. מבחינת מחלקת fstream של C++, קובץ הוא רצף של בתים.
2. כפי שניתן לראות באיור,

- מבחינה *פיזית*, קובץ הוא רצף של בלוקים, שכל אחד הוא רצף של בתים, וכולם בעלי אותו אורך (למשל, 1024 בתים).
- מבחינה *לוגית*, קובץ הוא רצף של רשומות שמאוחסנות פיזית לפי בלוקים. אפשר להסתכל על כל רשומה (א) כרצף של בתים; (ב) כרצף של שדות מסוגים שונים, שכל אחד מהם הוא רצף של בתים.

הערות חשובות:

(1) כל השדות שיוגדרו בהמשך דף זה כמחרוזות, הן מחרוזות בסגנון שפת C; כלומר, מחרוזת היא מערך של תווים שהתוו '\0' הראשון שמופיע בו מציין סוף מחרוזת.

(2) כמו כן, אם שדה מפתח של רשומה בקובץ מוגדר כמחרוזת, הוא גם יפורש כמחרוזות בסגנון שפת C.

תאור מפורט של מבנה קובץ Hash

המבנה הפיזי של קובץ מסוג Hash, כדלהלן:

ברמה הפיסית, נסתכל על קובץ כרצף של בלוקים (הנקראים באנגלית "buckets"):

לכל הבלוקים בקובץ יש, ברמה הפיזית, מבנה בסיסי משותף:

מספר סידורי של הבלוק בקובץ (BlockNr)	unsigned int
נתונים	1020 bytes

למרות זאת, לסוג הקובץ שבו אנו נעסוק כאן יהיו שני סוגים שונים של בלוקים, כל אחד עם התפקיד והמשמעות הייחודית שלו בתוך הקובץ:

א. הבלוק מס' 0 (בלוק ה-File Header), שמיועד לאחסון מידע לניהול הקובץ (ראה בהמשך, בסעיף שדן במבנה הלוגי של הקובץ).

ב. כל יתר הבלוקים, שמיועדים לאחסון מידע של המשתמש.

המבנה הבסיסי של כל בלוק (פרט לבלוק מס' 0), ברמה הפיסית, כדלהלן:

מספר סידורי של הבלוק בקובץ (BlockNr)	unsigned int
שמור לשימוש אחר (Filler)	[char[20
נתונים (Data)	1000 bytes

לכן, באופן בסיסי, לכל בלוק של נתונים יש שני שדות בעלי משמעות מבחינת הקובץ: השדה BlockNr, שהוא מספר שלם וחיובי, שיכיל את המספר הסידורי של הבלוק בקובץ, והשדה Data, שהוא מערך של בתים, יאחסן רשומות של הקובץ.
משמעות הדבר שהאורך של כל בלוק הוא (sizeof(unsigned int) + 20 + 1000) בתים.

המבנה הלוגי של קובץ מסוג Hash, כדלהלן:

באופן עקרוני, בכל אחד מהבלוקים של נתונים בקובץ בעל ארגון hash מאוחסנות רשומות ללא סדר גלוי. מיקומה של רשומה מסוימת מחושב ע"י הפעלת פונקצית hash על ערך של המפתח. פונקציה כזאת תחזיר מספר שלם וחיובי שהוא מיקומה של הרשומה בקובץ. משמעותו של ערך החישוב של פונקצית hash הוא *המספר הסידורי של בלוק* יחסית להתחלת הקובץ.

כל הרשומות שערכי ה-hash שלהם זהים, אמורות להיות מאוחסנות באותו בלוק. אם ערכי ה-hash של הרשומות בקובץ אינם מתחלקים באופן שווה בין הבלוקים השונים, ייתכן שתהיינה רשומות שלא יימצא להן מקום פנוי בבלוק המיועד להן. ידועות מספר שיטות לטיפול בבעיה הזאת (בעיית ה-"התנגשויות"); למשל:

א. שימוש בשטח נתונים משני (הנקרא שטח "גלישה" או overflow data area):

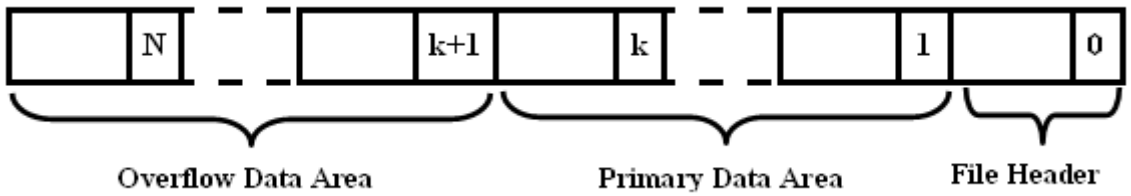
שטח זה מיועד לאחסון הרשומות שלא נמצא להן מקום בבלוק המיועד להן לפי פונקצית ה-hash. לפי שיטה זו הקובץ מורכב משני חלקים:

(1) השטח הראשי של הקובץ (Primary Data Area) מיועד לאחסן את הרשומות לפי ערך ה-hash שלהן.

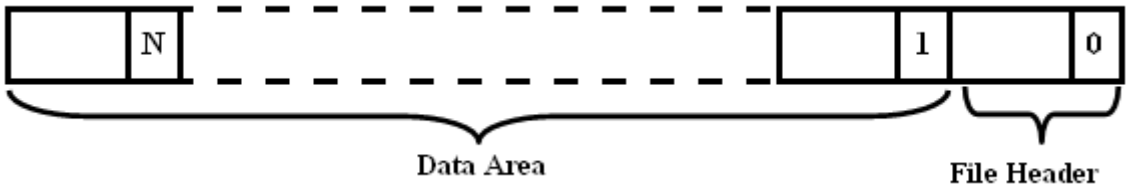
(2) שטח הגלישה (Overflow Data Area) ניתן למימוש בשתי דרכים שונות:

- שטח גלישה משורשר - כלומר, כל בלוק מלא בשטח הראשי, מצביע לשרשרת של בלוקים בשטח הגלישה. כל בלוק שבשרשרת מכיל רשומות שלפי ערך ה-hash שלהן היו מיועדות לאותו בלוק בשטח הראשי.
- שטח גלישה כללי - כל רשומה שלא יימצא לה מקום בבלוק המיועד לה בשטח הראשי, תאוחסן במקום הפנוי הראשון שיימצא לה בשטח הגלישה, ללא שום שרשור עם הבלוק המיועד לה בשטח הראשי.

קובץ hash בעל שטח ראשי ושטח גלישה מומש בדרך הבאה: הקובץ מחולק פנימית לשני אזורים, אחד מיועד לאחסן את שטח הנתונים הראשי, והשני מיועד לאחסן את שטח הגלישה.



ב. שימוש בשיטת ה-linear probing: בשיטה זו יש שטח נתונים (Data Area) אחד *ללא* שטח גלישה.



בשיטה הזאת מחפשים מקום פנוי בבלוק אחר ושמה מכניסים את הרשומה. כלומר, עוברים לבלוק הבא בקובץ ומחפשים את המקום הפנוי הראשון בו; אם נמצא כזה, מכניסים את הרשומה שמה; אם לא, ממשיכים לבלוק הבא אחריו עד שבאופן מעגלי חוזרים לבלוק שממנו יצאנו.

חשוב לציין שבמיני-פרויקט הזה נממש שיטה אחת בלבד, והיא שיטת ה-linear probing.

המבנה הבסיסי של כל בלוק (פרט לבלוק מס' 0), ברמה הלוגית, כדלהלן:

מספר סידורי של הבלוק בקובץ (BlockNr)	unsigned int
מספר רשומות שגלשו מהבלוק שלהן (NrOfOverflowedRecs)	unsigned int
מספר רשומות בפועל בבלוק (NrOfRecsInBlock)	unsigned char
שמור לשימוש אחר (Filler)	char[11]
נתונים (Data)	1000 bytes

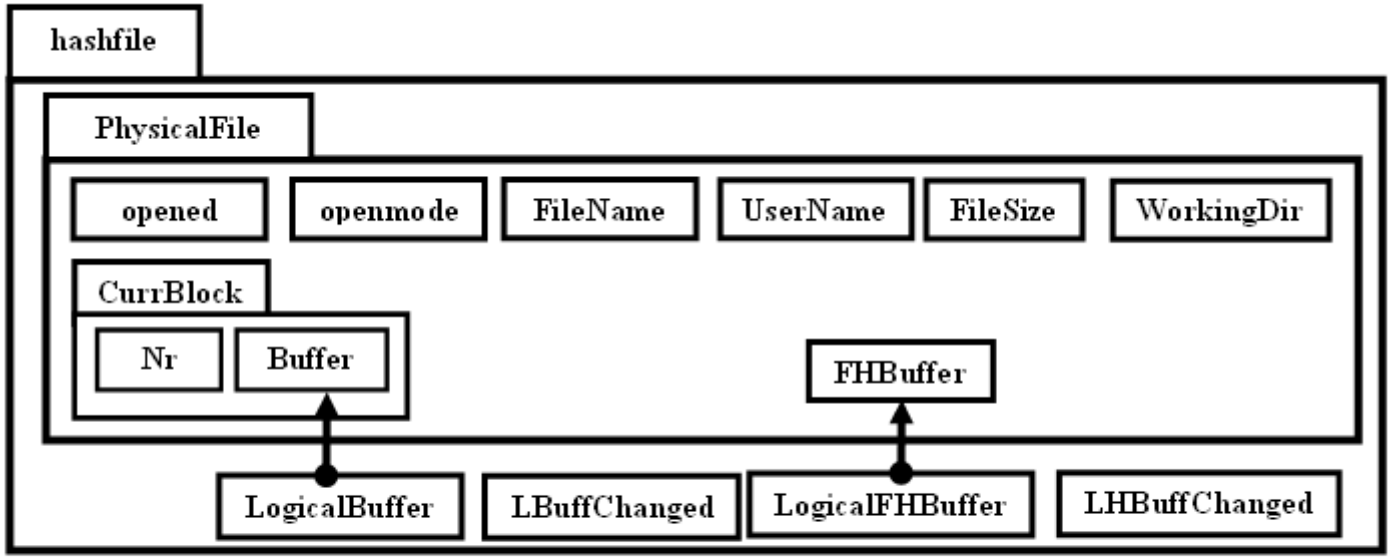
באופן בסיסי, ברמה הלוגית, לכל בלוק יש ארבעה שדות בעלי משמעות מבחינת הקובץ:
(א) ערכו של השדה BlockNr יהיה המספר הסידורי של הבלוק בקובץ (מספר שלם וחיובי); (ב) ערכו של השדה NrOfRecsInBlock יהיה סה"כ הרשומות שבכל רגע נתון מאוחסנות בבלוק; (ג) ערכו של השדה NrOfOverflowedRecs יהיה סה"כ הרשומות שהיו אמורות להיות בבלוק (לפי ערך ה-hash של המפתח שלהן) אבל גלשו לבלוק אחר; (ד) ערכו של השדה Data הוא מערך של בתים שבו תאוחסנה הרשומות המיועדות לאותו בלוק (לפי פונקצית ה-hash).

כפי שכבר נאמר דלעיל, לבלוק הראשון של כל קובץ (בלוק מס' 0), ה-File Header, יש תפקיד מיוחד והוא לאחסן מידע על הקובץ. המבנה המיוחד שלו כדלהלן:

מספר סידורי של הבלוק בקובץ (0 כמובן)	unsigned int
שם הקובץ (FileName) (ללא סיומת!)	(מחרוזת באורך מקסימאלי 11)char[12]
של בעל הקובץ (OwnerName)	(מחרוזת באורך מקסימאלי 9)char[10]
סה"כ מספר הבלוקים בקובץ (FileSize)	unsigned int
תאריך יצירת הקובץ (CreationDate)	(dd/mm/yy) (char[10]
אורך רשומה (RecordSize)	unsigned int
סה"כ הרשומות שכרגע בקובץ (NrOfRecsInFile)	unsigned int
מיקום התחלת המפתח ברשומה (KeyOffset)	unsigned int
טיפוס נתונים של המפתח (KeyType): "I" - הכוונה ל-integer (מספר שלם); במקרה זה, אורך המפתח ידוע והשדה KeySize לא רלוונטי. "S" – הכוונה למחרוזת בסגנון שפת C. במקרה זה, KeySize מציין את האורך המרבי האפשרי של שדה המפתח. למשל, אם ערכו של KeySize הוא 10, משמעות הדבר שהמפתח יכול להיות מחרוזת באורך 1 עד 9.	(1מחרוזת באורך) char[2]
אורך המפתח (KeySize)	unsigned int
שמור לשימוש אחר (Filler)	char[958]

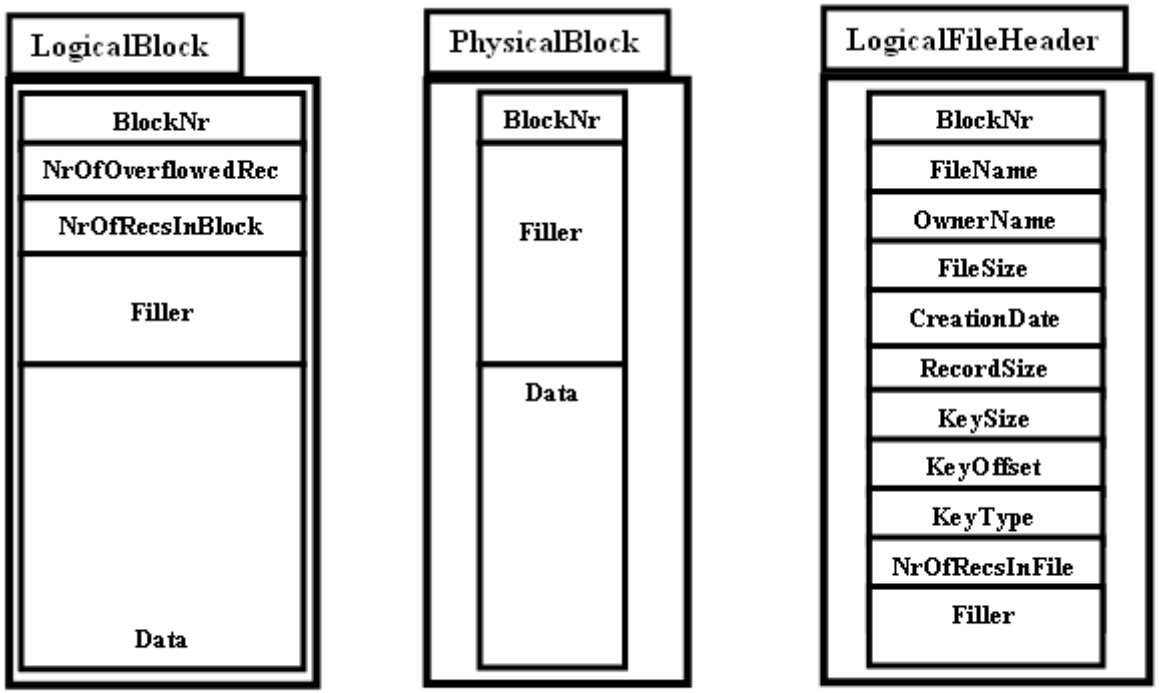
הערה חשובה: שים לב ששם הקובץ מאוחסן ללא סיומת ב-File Header, אבל הקובץ חייב להיווצר עם הסיומת hash.

המחלקות למימוש ארגון הקבצים HASH – ההיבט המבני



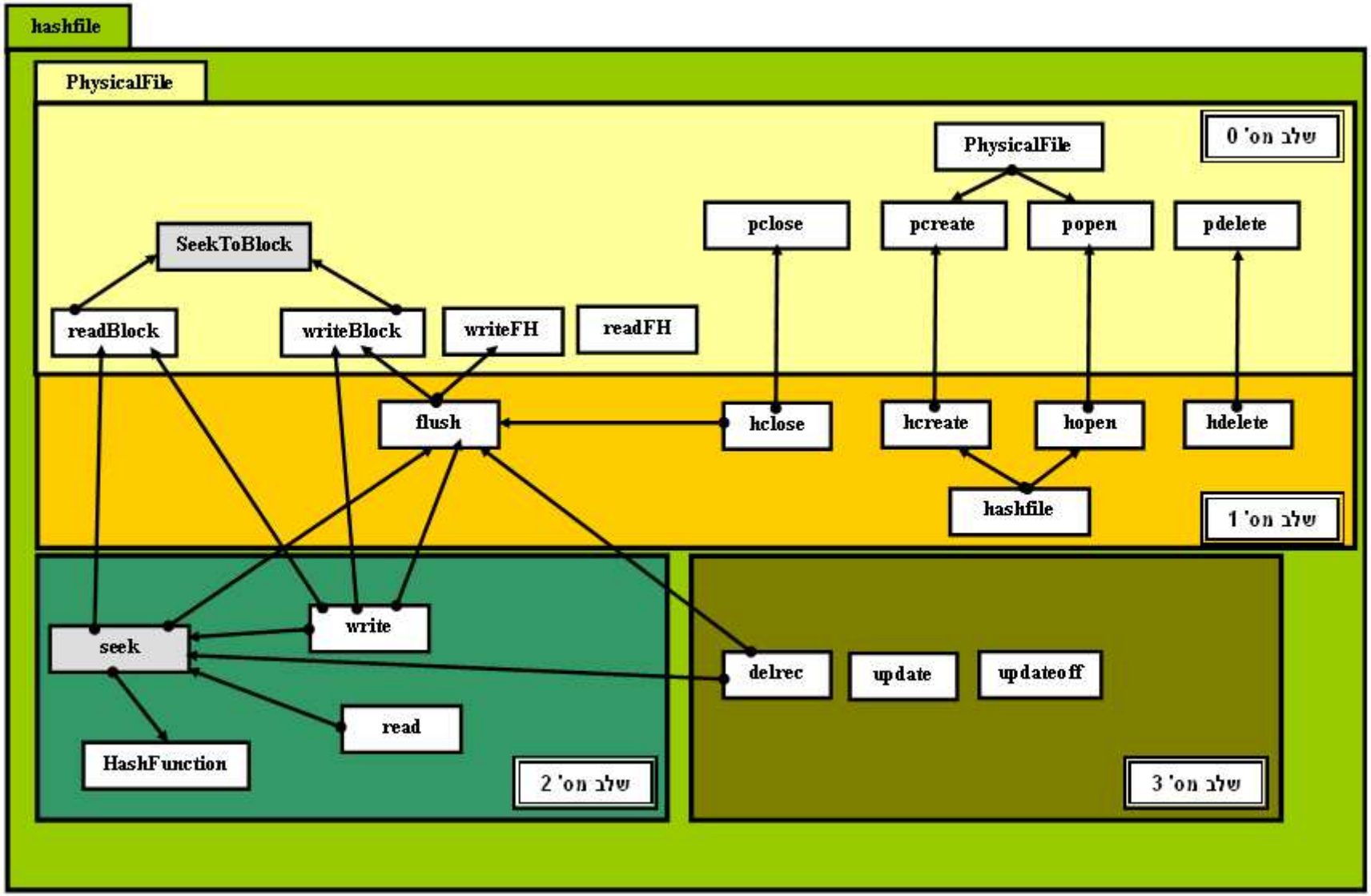
תיאור הקשר המבני שבין המחלקה PhysicalFile, שמייצגת את הרמה הפיזית של קובץ ה-hash,
לבין המחלקה hashfile, שמייצגת את הרמה הלוגית של קובץ ה-hash.

מבנה של בלוק ושל ה-File Header של קובץ – ברמה הפיזית וברמה הלוגית

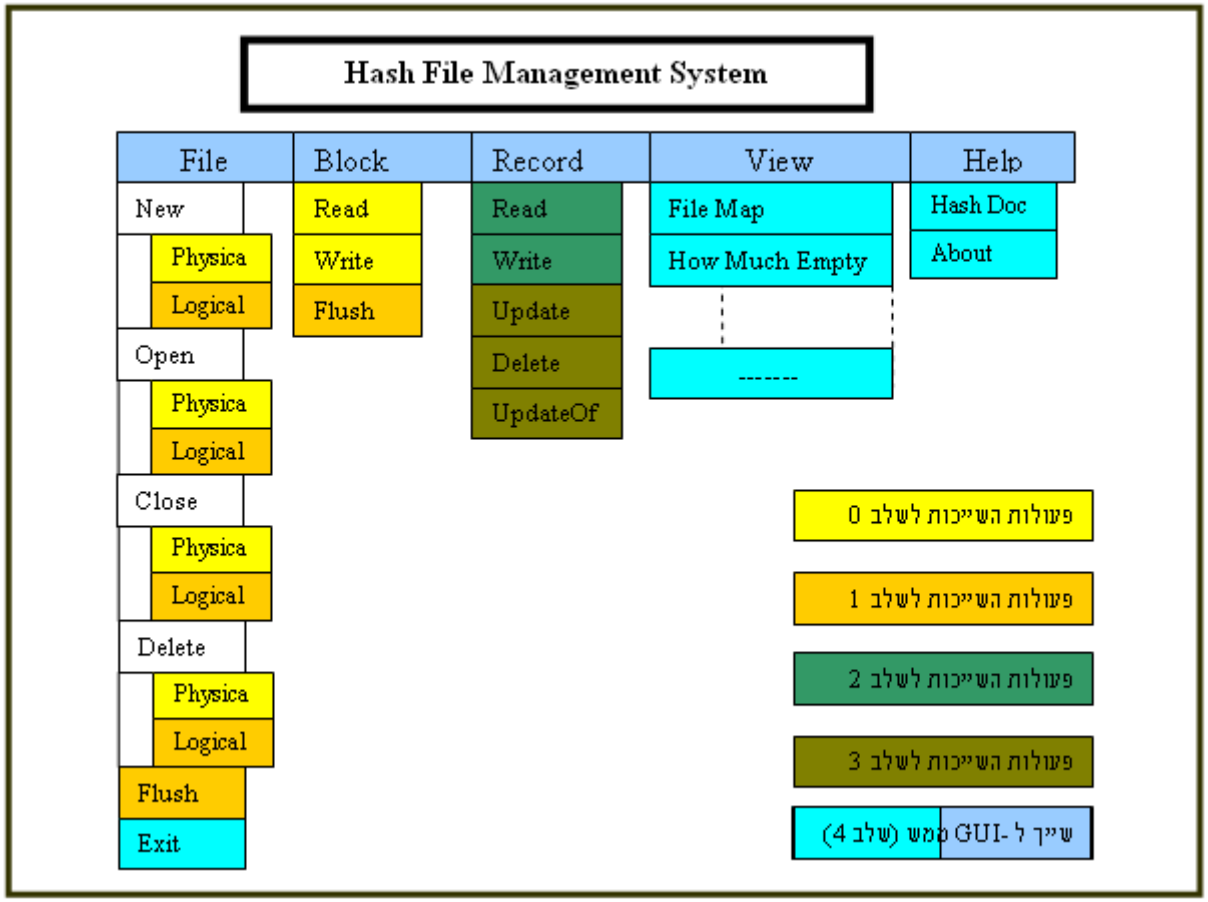


בלוק ברמה הלוגית | בלוק ברמה הפיזית | בלוק ה-File Header ברמה הלוגית

המחלקות למימוש ארגון הקבצים HASH – ההיבט ההתנהגותי



הצעה לתפריט לממשק הגראפי של המערכת לניהול ארגון קבצים ישיר (HASH)



כמה נקודות למחשבה בקשר לדו"ח הסופי ולהגנה על המיני-פרוייקט

הדו"ח הסופי:

- מבוא קצר (כחצי עמוד) על מטרות המיני-פרוייקט
- ניתוח שלבים של המיני-פרוייקט (לציין כל שלב ולפרט מה נעשה בו)
- בקשר למערכת לניהול ארגון קבצים hash שבנית:
  - תיאור מימושם של כל השלבים, במיוחד אלגוריתמים ומבני נתונים שהשתמשת. חשוב מאוד לנמק את החלטתך תוך ניתוח יתרונות וחסרונות של השיטה שבחרת יחסית לאלטרנטיבות ששקלת אבל לבסוף לא השתמשת.
  - תיאור מבני של המחלקות, במיוחד הקשר ביניהן (ירושה, וכו')
  - תאור השיטות השונות לטיפול בהתנגשויות בארגון קבצים hash : ניתוח יתרונות וחסרונות של כל שיטה, ונימוק בחירתך.
  - תאור מימושם של כל השלבים, במיוחד אלגוריתמים ומבנים שונים שהשתמשת; אם בשלב מסוים היו לך מספר מבנים ואלגוריתמים אלטרנטיוויים, תנתח את יתרונות וחסרונות של כל אחד מהם, ותנמק למה בחרת במה שבחרת.
  - מפרטים של כל הפונקציות public שכתבת; זה יכול להיות לקוח מהתיעוד הפנימי של התכנית עצמה. מפרט חייב לכלול: שם הפונקציה, סוג הערך המוחזר, פרמטרים, ותיאור קצר מה הפונקציה עושה. קיימות מערכות שונות אשר עוזרות לעשות זאת בצורה אוטומטית. ראה למשל doc++ בעזרת המנוע החיפוש המועדף עליך.
- סיכום כללי בו תכתוב מה להערכתך למדת במשך המיני-פרוייקט, דעותיך בקשר לקורס, צוות ההוראה, האם חשבון הדוא"ל של הקורס עזר, הצעות לשיפור, הערות, הארות, ביקורות וכו'.
- נספח בו מעין מדריך למשתמש של המערכת בגרסתה עם GUI, עם הסברים שהיבטים תכנוניים ותכנותיים.
- נספח שיכלול הצעה לשלב נוסף למיני-פרוייקט על סמך השלבים הנוכחיים, ונימוק להצעתך.
- אל תדפיס את הקוד – כבר ראינו אותו בזמן הבדיקה.
- הדו"ח חייב להיות תמציתי ולעניין – לא יותר מעשרים עמודים; עדיף שרוב הטקסט יהיה כתוב בגופן Arial בגודל 12 וברווח כפול.

הרצה והגנה על השלבים השונים ועל המיני-פרוייקט כולו:

- הדגמת המיני-פרוייקט בכל שלב; בשלב האחרון, זה צריך לכלול הסברים על ה-GUI מבחינה תכנונית ותכנותית.
- בדיקת יציבות כל הפונקציות (כלומר שהתוכנית "לעולם" לא נופלת)
- תכנון טוב של המיני-פרוייקט: הפונקציות קצרות ומשתמשות זאת בזאת
- טיפול בשגיאות (exception handling)
- מימוש פתרונות לבעיות תכנותיות בהן נתקלת במיני-פרוייקט
- שאלות שונות על המיני-פרוייקט. להלן רשימה של סוגי השאלות שיכולות להישאל:

1. השוואה בין שיטות שונות לפתרון בעיה מסוימת במיני-פרוייקט לבין השיטה שבחרת; למשל, השיטות השונות לטיפול בהתנגשויות והשיטה.
2. מדוע הפונקציה flush היא פונקציה נפרדת? (האם היה צורך – או טעם - להגדיר פונקציה נפרדת לביצוע כתיבה פיסית של בלוקים לקובץ hash?)
3. הצעת רעיון לשלב נוסף למיני-פרוייקט על סמך השלבים הנוכחיים.
4. ...