בס"ד

המרכז האקדמי לב

מיני-פרויקט בארגון וניהול קבצים – תשע"ו

רכז: ד"ר משה גולדשטיין

הקדמה

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

המטרה העיקרית של העבודה במיני-פרויקט תהיה לכתוב חבילה של מחלקות (classes) שמיישמות סימולציה של דיסק. כלומר, באמצעות חבילה זו יהיה אפשר:
(א) ליצור דיסק מדומה על בסיס קובץ אמיתי שיהיה בדיסק הקשיח של המחשב שלך
(ב) לפרמט את הדיסק ולאחר מכן לנהל את שטח הדיסק המפורמט
(ג) לבצע פעולות על קבצים כגון יצירה, ביטול, פתיחה, סגירה, כתיבה, קריאה, וכו'.

חבילה זו תשמש ממשק תכנה (API) שבאמצעותו תכניתן 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 במחלקת 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) קורס זה מניח ידע תכנותי (וטכני) על הנושאים שמפורטים בשורות הבאות. בכל זאת, לפני ביצוע השלב הראשון של המיני-פרויקט, יהיה שלב מיוחד, שנקרא לו שלב 0 לצורך תרגול הנושאים האלה, , המהווים תנאי ליתר השלבים.. נ:

- קלט/פלט בשפת ++C
- איך משתמשים בצורה נכונה בקבצי h.*
- איך יוצרים ספרייה סטאטית (LIB) ואיך משתמשים בה – ראה בקישור http://msdn.microsoft.com/en-us/library/vstudio/ms235627(v=vs.100).aspx
- איך יוצרים ספרייה דינאמית (DLL) ואיך משתמשים בה – ראה בקישור http://msdn.microsoft.com/en-us/library/ms235636.aspx

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

- <u>כל שלב יממש שכבת תכנה שתשמש בסיס לשלב הבא</u>. בגלל זה, כל שלב יהיה תלוי בזה שקדם לו; כלומר, על מנת לבצע שלב מסוים יהיה צורך בהשלמת השלב הקודם לו.
- <u>בכל שלב תהיינה שתי גרסאות של המיני-פרוייקט</u>:
  - <u>גרסת פיתוח</u>
    - <u>על מנת לממש כל שלב במיני-פרוייקט</u>, תכתבו יישום מסוג console שישמש כתכנית ראשית (main). ה-main הזה יהיה תפריט טקסטואלי שבאמצעותו תוכלו 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. <u>אם התפריט לא יכסה את כל מה שנידרש באותו שלב, תהיינה הורדות ציון בהתאם</u>.
    - <u>על מנת לכתוב את ה-main הנ"ל</u>, תיצרו project של Visual Studio שבאמצעותו תיווצר תכנית הרצה (exe) מסוג console. project מסוג זה יכיל קובץ cpp עם ה-main של התכנית, קבצי h שיכילו את הגדרת (declaration) המחלקות של המיני-פרוייקט ללא המימוש, וקבצי cpp שיכילו את מימוש (implementation) המחלקות הללו.
  - <u>גירסת הדגמה</u>
    <u>על מנת להדגים את כל מה שימומש בכל שלב, תיצרו שני project-ים נוספים של Visual Studio:</u>

(א) תיצרו project של Visual Studio מהסוג שיוצר ספריה סטאטית (LIB) בה תהיינה כל המחלקות שכתבתם לאותו שלב. באמצעות project מהסוג הזה, תוכלו ליצור ספרייה סטאטית.
ראה בקישור http://msdn.microsoft.com/en-us/library/vstudio/ms235627(v=vs.100).aspx

(ב) תיצרו project נוסף של Visual Studio שבאמצעותו תיווצר תכנית הרצה (exe) שתריצו בזמן ההדגמה במעבדה. project מסוג זה יכלול את ה-main של התכנית, את כל קבצי h המתאימים, ואת הספרייה הסטאטית שבניתם עם ה-project מ-(א).
הערה: ניתן לעשות דבר דומה עם ספריה דינאמית. ראה בקישור http://msdn.microsoft.com/en-us/library/ms235636.aspx
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-cmd.

- למימוש כל השלבים האלה, עליכם להכיר את המחלקות המאפשרות טיפול בקבצים (קלט/פלט) בשפת ++C. כחומר התחלתי בנושא, ניתן להסתכל במצגות של פרק 12, פרק 14, פרק 19 ו-פרק 22 של הספר מאת Deitel&Deitel על תכנות בשפת ++C. חומר נוסף משלים אפשר למצוא בפרק 3 של הספר "Thinking in C++, Volume 1" ובפרק 2 של הספר "Thinking in C++, Volume 2", מאת Bruce Eckel.

(ג) בשלב האחרון של המיני-פרויקט, התפריט הטקסטואלי יוחלף בממשק גראפי (GUI) שייכתב בשפת #C או באמצעות QT. למימוש השלב הזה, עליך להכיר תכנות ממשקים גראפיים (GUI) בשפת #C או ב-QT. ב- Moodle של הקורס תוכלו למצוא הסברים ודוגמאות תכנותיות לשתי האפשרויות:
- אושרי כהן הכין מדריך מפורט בקשר לשילוב של תכנית ראשית בעלת GUI כתובה בשפת #C, עם ספרייה *דינאמית* של קוד כתוב ב-++C. בנוסף לזה, תוכלו למצוא דוגמאות נוספות, מאוד שימושיות, בשפת #C, באתר הבא: www.java2s.com/tutorial/csharp/catalogcsharp.htm
- עזרא דשט ושני תלמידים שלו הכינו מדריך מפורט לעבודה עם QT, לפיתוח GUI לתכנית ראשית כתובה בשפת ++C.
בשלב זה, השימוש בספרייה דינאמית הינו הכרחי.

הערות חשובות

(א) חובה להסתכל באתר הקורס ב-Moodle כי ייתכנו שינויים במשך הסמסטר. אם בפועל יהיו שינויים, הם יפורסמו בזמן השיעור ובאתר הקורס שב-Moodle.

(ב) יהיה פורום במסגרת האתר של הקורס ב-Moodle, שישמש כלוח לשאלות ותשובות בו כל תלמיד/תלמידה יוכל/תוכל לשלוח שאלות, הערות והארות, וגם לענות עליהן. מורי הקורס גם ישתתפו בפורום לפי הצורך. הפורום יהיה האמצעי היחיד לברור שאלות ובעיות, בנוסף לשיעור במעבדה. מורי הקורס לא יענו להודעות דוא"ל שתישלחנה ישירות לכתובת הדוא"ל שלהם.

(ג) בקשר להצטרפות לקורס: *לא יהיה ניתן להצטרף* לקורס הזה אחרי מועד הגשת שלב מס' 0 (כשלושה שבועות מתחילת הקורס), 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/אחת יעניק/תעניק לעניין הנידון. כלומר, כל אחד/אחת יצטרך/תצטרך 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/אחת יחליט/תחליט ליישם.

(ה) התפישה בקורס הזה היא שהשיעור במעבדה מיועד, בין היתר, לדיון ולניתוח של המשימות שעל הפרק. המורה יסביר/תסביר מה משמעותן של המשימות השונות של אותו שלב, וינחה/תנחה 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

סביבת הפיתוח בשימוש במיני-פרוייקט תהיה רק Visual Studio בגרסתה המותקנת במעבדות של המכון.
כל תלמיד/תלמידה חייב/חייבת לשמוע את הקורס מהמורה שהוא/היא רשום/רשומה אצלו/אצלה - פרט למקרים חריגים ביותר לא תתאפשרנה העברות בין קבוצות מעבדה.

העבודה *חייבת להיות בזוגות ושני/שתי בני/בנות הזוג חייבים להיות מאותה קבוצת מעבדה*; כלומר, לא יתאפשר מצב שבני/בנות זוג יהיו/תהיינה מקבוצות מעבדה שונות: אם יהיה מקרה כזה, העבודה לא תיבדק. כל זה נכון פרט למקרים חריגים שכל מורה (או רכז הקורס) יחליט לגופו של כל עניין ספציפי.
נוכחותם של שני/שתי בני/בנות הזוג בזמן השיעור במעבדה ***חובה***. בהתחלת כל שיעור, המורה יעביר/תעביר דף נוכחות. בלא נוכחות בקורס לא תתבצע הגנה על הפרויקט בסוף הקורס.

חשוב מאוד לציין *שלכל מי שיחסיר/תחסיר מעל 25% מהשיעורים, ללא סיבה מוצדקת ומאושרת, לא תהיה זכות להגיש את המיני-פרוייקט בסוף הסמסטר וגם לא להגן עליו*, אפילו אם מימש/מימשה אותו מ-א' עד ת' – משמעות הדבר שיצטרך/תצטרך לקחת את הקורס שוב.

אם תהיינה בעיות כלשהן במשך הסמסטר, ניתן יהיה לפנות באמצעות הדואר האלקטרוני למורה שהתלמיד/תלמידה רשום/רשומה אצלו/אצלה בלבד.

נוהלי הגשה ובדיקה

בהתאם ללוח הזמנים של הסמסטר, המפורט כאן, בתום ביצוע כל אחד מהשלבים במיני-פרוייקט, כל זוג יגיש את העבודה שלו לתא ההגשה שייפתח ב-Moodle של הקורס. ההגשה חייבת לעמוד בכללים הבאים:

(א) שמות, תעודות זהות וכתובות דוא"ל של שני/שתי בני/בנות הזוג *חייבים* להופיע כהערות בהתחלת קובץ התכנית הראשית (ה-main).

(ב) כל זוג יכין שני קבצי zip שיכילו את מצבו העדכני ביותר של כל המיני-פרויקט, נכון לרגע ההגשה. לתוך קובץ ה-zip הראשון תוכנס כל התיקייה של גרסת הפיתוח, על כל מרכיביה; לתוך קובץ ה-zip השני תוכנסנה שתי התיקיות של גרסת ההדגמה, על כל מרכיביה. בשני המקרים מדובר בכל ה-source files בשפת C++ (ובשפת C#, במקרה שבשלב האחרון תעשו את ה-GUI בשפה הזאת), כל ה-object files, קבצי הספריות (הסטאטית והדינאמית), קובץ ההרצה (exe) והדו"ח החלקי (ראה בהמשך) המתייחס לאותו שלב שמוגש באותו רגע.

(ג) כל זוג יגיש את שני קבצי zip הנ"ל לתא ההגשה המתאים שיהיה ב-Moodle. מומלץ שכל בן/בת זוג יגיש/תגיש לתא ההגשה שלו/שלה כאילו עשה/עשתה את העבודה לבד.

(ד) *להגשות באיחור יינתן ציון 0* פרט למקרים מוצדקים ומאושרים ע"י המורה של קבוצת המעבדה המתאימה.

(ה) קבצי zip הנ"ל חייבים להיות לפי הפורמט הבא:
גרסת פיתוח - fmslabdev-nnn-ID1-ID2.zip
גרסת הדגמה - fmslabdsp-nnn-ID1-ID2.zip
בשני המקרים, nnn – מס' השלב של המיני-פרויקט, IDi – מס' תעודת זהות של בן/בת הזוג i.

בתום השלבים מס' 0, מס' 1, מס' 3, ובסוף המיני-פרוייקט,

(א) הדגמת שלב של המיני-פרויקט: בזמן שיעור מעבדה שיתקיים מיד אחרי תאריך ההגשה ב-Moodle, כל זוג ידגים למורה קבוצת המעבדה שלהם/שלהן, את הרצת גרסת ההדגמה של המיני-פרוייקט שהוגשה ל-Moodle.

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

(ג) הגנה על אותו שלב: תוך כדי בדיקת ההרצה, תתבצע *הגנה* על אותו שלב: המורה ישאל/תשאל שתיים או שלוש שאלות, כולל התמצאות בקוד, וייתן/תיתן ציון בהתאם; הציון על ההגנה יהיה שבר בין 0 ל-1. ראה בסעיף "ציון" בקשר למשמעות ציון ההגנה.

(ד) בדיקת התיעוד ואיכות התכנות של אותו שלב: תיעשה באופן לא יסודי במיוחד; כל מטרתה להעיר עליהם על מנת שבשלב הבא יתוקנו כל הליקויים בהתאם להערות. אם מתברר שהליקויים לא תוקנו בהתאם להערות, תירשם הורדת ציון באותו שלב.

(ה) *דו"ח חלקי*: ביחד עם כל מה שתואר בסעיפים (א) עד (ד), תידרש כתיבת *דו"ח חלקי* שמתייחס לשלב/שלבים הספציפיים שההרצה שלהם נבדקת באותו רגע במעבדה (להנחיות על כתיבת הדו"ח, ראה כאן). בזמן בדיקת ההרצה, המורה יבדוק/תבדוק שהדו"ח החלקי נכתב, תוך הסתכלות יחסית שטחית, בלי בדיקת עומק של תכנו; כל מטרתה להעיר 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 ו/או האלגוריתם שמומש בפועל בתכנית.

(ו) הציון של אותו שלב (ראה סעיף "ציון" בהמשך): יבוא לידי רישום על ידי המורה רק אחרי שהמורה יברר/תברר שהשלב הוגש ל-Moodle לפי ההנחיות דלעיל.

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

- מועד הדגמת/הרצת המיני-פרויקט כולו ייקבע על ידי מדור בחינות כחלק ממועדי א' של הבחינות של סוף הסמסטר. מובן מאליו שלהדגמת/הרצת המיני-פרויקט אין מועד ב'.
- הרצת המערכת תיבדק באמצעות הממשק הגראפי, תוך הענקת ציון בדומה לשלבים הקודמים (ראה סעיף "ציון" בהמשך).
- כל ההיבטים האחרים של המיני-פרויקט (איכות התכנות, תיעוד ודו"ח סופי) ייבדקו לעומק, ויינתן בהתאם ציון למיני-פרויקט כולו.

הערה כללית חשובה: מובן מאליו שבעקבות ההגנה על כל שלב, וגם על המיני-פרוייקט כולו, יתכנן שיהיה הבדל בציון בין כל אחד/אחת מבני/בנות הזוג.

ציון

הקריטריונים לקביעת ציון יהיו לפי הטבלה הבאה:

משימה	משקל	הערות
ה-main כיישום Win32 Console (טקסטואלי)	**0%**	ה-main חייב לאפשר את בדיקת ההרצה של כל הפונקציונאליות הנדרשת בכל שלב ושלב של המיני-פרויקט.
שלב מס' 0	**15%**	
שלב מס' 1	**15%**	
שלב מס' 2	**10%**	
שלב מס' 3	**25%**	
שלב מס' 4	**10%**	
שלב מס' 5: ה-main כיישום חלונאי בעל GUI, כתוב בשפת #C, או ב-++C עם QT.	**15%**	הציון ייקבע לפי איכות תכנון הממשק, מבחינת השימושיות (usability) של הממשק, וכו'. ה-GUI חייב לאפשר את בדיקת ההרצה של כל הפונקציונאליות הנדרשת בכל שלב ובמיני-פרויקט כולו.
דו"ח מסכם של המיני-פרוייקט כולו	**10%**	הדו"ח המסכם יכלול: (א) צרוף הדו"חות החלקיים שהוגשו בזמני הגשת שלבי המיני-פרויקט. (ב) הערות ביקורתיות בקשר לקורס בכללותו (ג) הערות ביקורתיות בקשר לסגל ההוראה של הקורס (ד) הצעות לשיפור הקורס (ה) הצעות להוספת שלב/שלבים למיני-פרויקט. להנחיות מפורטות בקשר לדו"ח, תסתכל כאן. <u>מועד הגשת הדו"ח המסכם</u> יהיה מועד הדגמת/הרצת המיני-פרוייקט כולו.
הציון הכולל	**100%**	הציון הכולל של המיני-פרויקט יינתן אחרי שהמורה יקרא/תקרא את כל הדו"חות – בבקשה, תנסו להיות מאוד תמציתיים (לא יותר מ-15 עמודים, ב-12 Arial ורווח כפול).

הקריטריונים לקביעת ציון של כל שלב של המיני-פרויקט יהיו לפי הטבלה הבאה:

הרצת השלב עם ה-main הטקסטואלי.	**50%**	התכנית חייבת לבצע את משימותיה בצורה נכונה, בהתאם לדרישות ולמפרטים. ה-main חייב לאפשר את בדיקת ההרצה של כל הפונקציונאליות הנידרשת בכל שלב.
הגנה על השלב		ההגנה על שלב מסוים של המיני פרויקט תשמש לקביעת הציון של אותו שלב, לפי הנוסחה הבאה: (ציון ההגנה) * (ציון ההרצה), כאשר *ציון ההגנה* יהיה ערך בין 0.0 ל-1.0 ו*ציון ההרצה* יהיה ערך בין 0 ל-100. לדוגמה: אם הציון של ההרצה של שלב מסוים הוא 90, ובהגנה נותנים לך 1.0, הציון של ההרצה יהיה גם 90; לעומת זאת, אם בהגנה נותנים לך 0.95, הציון של ההרצה יהיה 85.5 במקום 90. בקשר לסוג השאלות שייתכן שתישאלנה, תסתכל כאן
איכות התכנות של השלב	**25%**	<u>איכות התכנות שלך יוערך לפי הקריטריונים הבאים</u>: ▪ 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 הקוד של השלב	**25%**	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.

תכנית הסמסטר

מטלות הקורס לאורך הסמסטר

שבוע	פעילות
1, 2	**מבוא למיני-פרוייקט** *משימתך* במיני-פרויקט הזה, לבנות מערכת תכנה שמיישמת סימולציה של דיסק. **התמונה הכוללת של המיני-פרויקט** על מנת ליישם את המערכת הזאת, נזהה הפשטות מתאימות לדיסק, מהסוג המתואר **כאן**. המרכיבים המבניים וההתנהגותיים שלפיהם אנחנו מציעים לממש את זה, יבואו לידי ביטוי כמחלקות בשפת C++ והקשרים המוגדרים ביניהן, שתבואנה לידי מימוש בחמישה שלבים: שלב מס' 0: בשלב זה ממומשת התשתית הפיזית של הדיסק, שהיא שכבה מעל שכבת ה-fstream. ברמה הפיזית, מסתכלים על הדיסק כרצף של בלוקים (סקטורים) בעלי מבנה בסיסי, ומאורגנים לפי מספר סידורי של הסקטור בדיסק (שהוא תרגום מהשלישייה [צילינדר, מסילה, סקטור]), בלי התייחסות לשום מבנה לוגי מעבר לתווית הזיהוי של הדיסק. שלב מס' 1: בשלב זה ממומשת התשתית הלוגית של הדיסק, שהיא שכבה מעל הרמה הפיזית. ברמה הלוגית מסתכלים על דיסק כרצף של סקטורים בעל מבנה שבא לידי ביטוי על ידי (א) ה-Disk Allocation Table (DAT) שתאפשר לייצג ולנהל שטח פנוי לעומת שטח תפוס, ו-(ב) התיקייה הראשית שתאפשר לנהל קבצים ותת-תיקיות בדיסק. שלב מס' 2: בשלב זה, על סמך התשתית שנבנתה בשני השלבים הקודמים, ממומש מנגנון המאפשר יצירת, מחיקת והרחבת קבצים שבתיקייה הראשית. שלב מס' 3: בשלב זה, על סמך מה שנעשה בשלבים הקודמים, ממומש מנגנון המאפשר פתיחת וסגירת קובץ וטיפול בתוכן של קובץ, כגון פעולות כתיבה וקריאה של רשומה. שלב מס' 4: בשלב זה נשלים את מלאכת המימוש של המנגנון המאפשר לנו לטפל בתוכן של קובץ, כגון פעולות עדכון ומחיקה של רשומה שנקראה קודם לכן לצורך עדכון באמצעות read-for-update. עד כאן, ממשק ה-main הינו טקסטואלי – הוא ממומש כ-console application. בדומה למימוש המחלקות עד עכשיו, הוא כתוב בשפת C++. שלב מס' 5: לאחר מימוש הסימולציה של הדיסק כמחלקות בשפת C++, ימומש השלב האחרון של המיני-פרויקט, שלב מס' 5, שמתוכנן ליישם את ה-main של התכנית כממשק גרפי (GUI).כלומר, הממשק הטקסטואלי יוחלף בממשק גרפי (GUI). אתם חופשיים להחליט לממש את ה-GUI בשפת C# באמצעות WPF של NET. או בשפת C++ תוך שימוש כלי מיוחד לפיתוח ממשקים גרפיים שנקרא QT. בשני המקרים סביבת הפיתוח תהיה Visual Studio. במקרה של שימוש ב-QT, אפשר להשתמש בו באופן עצמאי או ניתן להשתמש בו מתוך Visual Studio אם QT מותקן כתוסף (plug-in) ל-Visual Studio. 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. חשוב לציין שלמרות שהמיני-פרויקט מתוכנן על כל שלביו, 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. **הערה כללית, תקפה לכל המיני-פרויקט:** אם יש תקלה בביצועה של פונקציה כלשהי בחלק כלשהו של המערכת לניהול הדיסק המדומה של המיני-פרוייקט הזה, הטיפול בתקלה חייב לבוא לידי ביטוי באמצעות מנגנון ה-try-throw-catch של שפת C++ ולא באמצעות קוד שגיאה או הדפסת הודעת שגיאה מתוך הפונקציה. רק ברמת הממשק למשתמש (חלונאי או לא) התקלה תתבטא כהודעת שגיאה מתאימה למשתמש. בקשר לנושא של exception handling, תסתכלו בפרק מס' 1 של הספר **"Thinking in C++, Volume 2"** מאת Bruce Eckel ובמצגת

של **פרק 13** של הספר מאת Deitel&Deitel. בנוסף לזה, כדאי להסתכל בדוגמאות תכנותיות שיכולות לעזור לכם במשך הסמסטר: - דוגמאות בקשר להמרה של **מחרוזות מסיגנון C לסיגנון ++C**. - דוגמאות בקשר **לשימוש ב-buffer**. **התחלת המיני-פרויקט** **שלב מס' 0: תשתית הדיסק המדומה ברמה הפיזית שלו – הגשה: בשיעור מס' 3** משימתך בשלב זה, ובשלב הבא של המיני-פרויקט להגדיר וליישם את מחלקות התשתית (מבנה + פעולות) המתאימות למבנה שתואר **כאן**. ספציפית לשלב הזה, משימתך בשלב זה של המיני-פרויקט להגדיר וליישם את מחלקות התשתית (מבנה + פעולות) המיישמות את הדיסק המדומה ברמה הפיזית שלו, כרצף של סקטורים בהתאם למבנה שתואר **כאן**. הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא **כאן**.	
שלב מס' 1: תשתית הדיסק המדומה ברמה הלוגית שלו – הגשה: בשיעור מס' 5 משימתך בשלב זה של המיני-פרויקט להגדיר וליישם את התשתית (מבנה + פעולות) של הדיסק המדומה ברמה הלוגית על סמך התשתית ברמה הפיזית שמומשה בשלב מס' 0 של המיני-פרויקט. כלומר, בשלב הזה נממש פעולות הקשורות לניהול שטח הדיסק באמצעות ה-DAT. הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא **כאן**.	3, 4
שלב מס' 2 : ניהול קבצים ברמה הפיזית: יצירת, מחיקת, והרחבת קבצים – הגשה: בשיעור מס' 7 משימתך בשלב זה של המיני-פרויקט, ליישם יצירת והרחבת קובץ תוך הקצאת שטח דיסק בשבילו, ומחיקת קובץ תוך החזרת השטח שהוא תפס, לשטח הפנוי של הדיסק. בשלב הזה מסתכלים על קובץ כרצף של סקטורים, ללא התייחסות לרמה הלוגית שבה מסתכלים על קובץ כרצף של רשומות. הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא **כאן**	5, 6
שלב מס' 3: ניהול קבצים ברמה הלוגית: פתיחה וסגירה של קובץ, כתיבה וקריאה של רשומה, וגישה ישירה לרשומה, עדכון ומחיקה של הרשומה הנוכחית – הגשה: בשיעור מס' 9 משימתך בשלב זה של המיני-פרויקט, ליישם (א) פתיחת קובץ תוך הקצאת שטח זיכרון לחוצץ, (ב) סגירת קובץ תוך החזרת שטח החוצץ למערך החוצצים של המערכת, (ג) כתיבת רשומה לתוך המקום הנוכחי בקובץ, (ד) קריאת רשומה מתוך המקום הנוכחי בקובץ, (ה) גישה ישירה לרשומה כלשהי בקובץ, (ו) ביטול הנעילה של הרשומה הנוכחית במקרה שנקראה לצורך עדכון (read-for-update) ובהתאם ננעלה והיישום החליט לבטל את הנעילה ולשחרר את הרשומה, (ז) עדכון הרשומה הנוכחית, (ח) מחיקת הרשומה הנוכחית. פעולת עדכון ופעולת מחיקה של רשומה ניתנות לביצוע רק עם הרשומה נקראה לצורך עדכון ובהתאם ננעלת. שתי פעולות אלו משחררות את הנעילה. הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא **כאן**	7, 8
שלב מס' 4: מערכת ניהול הדיסק כולה: מימוש המחלקה DMS (Disk Management System): – הגשה: בשיעור מס' 11 משימתך בשלב הזה של המיני-פרויקט, ליישם את מערכת ניהול הדיסק כולה תוך מימוש המחלקה **DMS** המשמשת כשכבת תכנה העליונה. הגדרה מפורטת של משימתך לשלב זה של המיני-פרויקט אפשר למצוא **כאן** דיאגרמה שמתארת את הקשרים המבניים בין המחלקות העיקריות המרכיבות את המערכת לניהול הדיסק המדומה, ניתן למצוא **כאן**.	9,10

11,12	**<u>שלב מס׳ 5: מימוש הממשק הגראפי – הגשה: בשיעור מס׳ 13</u>** בשלב זה של המיני-פרוייקט, תיישם ממשק גראפי שיחליף את הממשק הטקסטואלי שהיה עד עכשיו, וישמש כ-main של מערכת לניהול הדיסק המדומה. הגדרה של משימתך לשלב זה של המיני-פרוייקט אפשר למצוא **כאן** .
13	**יום אחרון של הסמסטר: הדגמת והגנת המיני-פרויקט כולו**

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

משימתך בשלב זה, ובשלב הבא, של המיני-פרויקט להגדיר וליישם את מחלקות התשתית (מבנה + פעולות) המתאימות למבנה שתואר **כאן**. ספציפית לשלב הזה, משימתך בשלב זה של המיני-פרויקט להגדיר וליישם את מחלקות התשתית (מבנה + פעולות) המיישמות את הדיסק המדומה ברמה הפיזית שלו, כרצף של סקטורים (או סקטורים) בהתאם למבנה שתואר **כאן**.

מחלקת disk

- **בקשר לחלק המבני של המחלקה**

במחלקה זו נגדיר את השדות הבאים:

vhd

שדה שמיועד להיות המקום בו נתוני ה-Volume Header יועברו בזמן ביצוע mount של דיסק קיים, או ייווצרו בזמן createdisk של דיסק חדש. בשני המקרים, הנתונים ייקראו/ייכתבו מתוך/לתוך הסקטור מס' 0 של הדיסק המדומה.

dat

שדה שמיועד להיות המקום אליו ה-DAT יועבר בזמן ביצוע mount של דיסק קיים, או ייווצר בזמן createdisk של דיסק חדש. בשני המקרים, הנתונים ייקראו/ייכתבו מתוך/לתוך הסקטור מס' 1 של הדיסק, בו ה-DAT אמור להימצא.

rootdir

שדה שמיועד להיות המקום אליו נתוני התיקייה הראשית (ה-Root Directory) של הדיסק המדומה יועברו בזמן ביצוע mount של דיסק קיים, או ייווצרו בזמן createdisk של דיסק חדש. בשני המקרים, הנתונים ייקראו/ייכתבו מתוך/לתוך שני הסקטורים של ה-cluster מס' 1 של הדיסק, בהם התיקייה הראשית אמורה להימצא.

mounted

שדה בולאני שישמש כדגל: אם ערכו false, משמעותו שהדיסק המדומה עדיין לא זמין (כלומר, שעדיין לא בוצעה mountdisk; אם ערכו true, משמעותו שהוא זמין.

dskfl

אובייקט מסוג fstream המייצג את הקובץ שמכיל את הדיסק המדומה.

currDiskSectorNr

שדה מסוג unsigned int שערכו הוא המספר הסידורי של הסקטור בדיסק שכרגע בחוצץ של קובץ מסוים.

הערה 1: זה ברור שעל מנת לטפל באופן מסודר בכל אחד משלושת השדות vhd, dat ו-rootdir, יהיה צורך להגדיר מחלקות מתאימות גם לכל אחד מהם.
הערה 2: ייתכן שבמשך פיתוח המיני-פרויקט יהיה צורך להגדיר שדות נוספים, בהתאם להחלטות יישום של כל אחד ואחד מכם.

- **בקשר לפונקציות ליישום המחלקה הזאת**

  - **disk() - default constructor**

תפקידו של הבנאי הזה לאתחל אובייקט מסוג disk על כל השדות שלו בלי לקשר לו אף שם של קובץ (כלומר, שם הדיסק יהיה מחרוזת ריקה) ובלי לקשר לו אף בעל (כלומר, שם בעל הדיסק יהיה גם מחרוזת ריקה), וכו'.

- **disk(string &, string &, char)**

תפקידו של הבנאי הזה לאתחל אובייקט מסוג disk.

- האובייקט שנוצר יהיה קשור לקובץ ששמו הערך של *הפרמטר הראשון* (ברור שהקובץ הזה הוא האמצעי שלנו למימוש הדיסק המדומה).
- שם בעל הדיסק ששמו הערך של *הפרמטר השני*.
- ערך *הפרמטר השלישי* הוא קוד שקובע אחד משני דברים:

'c' - הפונקציה תפעיל את createdisk על מנת ליצור ממש את הדיסק המדומה ולאחר מכן תפעיל את mountdisk על מנת לפתוח אותו ולהפוך אותו לזמין.
'm' - הפונקציה תפעיל את mountdisk על מנת לפתוח את הדיסק המדומה ולהפוך אותו לזמין.
זה ברור שבמקרה הראשון הקובץ חייב להיות לא קיים, ובמקרה השני הוא חייב להיות קיים.

הערה: ברור שבשתי הגרסאות של הבנאי של המחלקה disk, הבנאיים (default constructors) של מחלקות מתאימות לתיקייה הראשית, ל-DAT, ו-FCB, תקראנה באופן אוטומטי בזמן יצירת אובייקט מסוג disk.

- **~disk( )**

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

- **void createdisk(string &, string &)**

תפקידה של פונקציה זו ליצור דיסק מדומה; כלומר, הפונקציה הזאת תיצור קובץ שאורכו כגודל הדיסק בבתים (או במספר סקטורים).
שני הפרמטרים שפונקציה זו מקבלת הם מחרוזות:
(א) *המחרוזת הראשונה* תהיה שם הדיסק
(ב) *המחרוזת השנייה* תהיה שם בעל הדיסק.
ראשית כל, הפונקציה הזאת תמספר את *כל* הסקטורים של הדיסק – תכתוב את כולם, מהראשון ועד האחרון, עם המספרים הסידוריים שלהם. מיד לאחר מכן, הפונקציה תאתחל את ה-Volume Header של הדיסק: הפרמטר הראשון יועתק לשדה diskName של vhd והפרמטר השני יועתק לשדה diskOwner של vhd. כל השדות האחרים ב-vhd יקבלו את ערכיהם בהתאם ל**תיאור מבנה הדיסק.** אחרי זה, השדה vhd יועתק לסקטור המיועד לו בדיסק המדומה. כפעולה אחרונה, הקובץ המכיל את הדיסק המדומה ייסגר.
במקרה של הצלחה, בסופו של התהליך יהיה לנו דיסק מדומה *לא* מפורמט (משמעותו של דבר שהשדה isFormatted חייב לקבל ערך בולאני false).
הערה: לפני מימוש הפונקציה הזאת כדאי לך מאוד להסתכל בדוגמה התכנותית המופיעה **במצגת של פרק 14** של הספר מאת Deitel&Deitel.

- **void mountdisk(string &)**

תפקידה של פונקציה זו לפתוח את הקובץ הממשמש את הדיסק המדומה, כאשר *הפרמטר הראשון* שהיא מקבלת הוא שם הקובץ. במקרה של הצלחה בפתיחת הקובץ, הפונקציה תיתן לשדה mounted את הערך true ותקרא את תוכנם של כל הסקטורים המכילים את המידע המבני של הדיסק המדומה, לתוך השדות המתאימים במחלקה disk.

- **void unmountdisk( )**

תפקידה של פונקציה זו לבצע את הצעדים הבאים:
(א) לעדכן את כל הסקטורים המכילים מידע מבני של הדיסק המדומה - זה יתבצע באופן חלקי או מלא בהתאם לשינוי שיתברר שהיה מאז שהסקטורים האלה נקראו בזמן mountdisk.
(ב) לסגור את הקובץ של הדיסק המדומה.
(ג) לתת לשדה mounted את הערך false.

- **void recreatedisk(string &)**

תפקידה של פונקציה זו לאתחל מחדש את הדיסק המדומה. כלומר, הפונקציה הזאת תבצע את כל מה שתואר בקשר לפונקציה createdisk, אבל בלי ליצור מחדש את הדיסק המדומה.

כמובן שזה יוכל לצאת לפועל אם שלושה תנאים מתקיימים:
(א) *הפרמטר* חייב להיות מחרוזת עם שם בעל הדיסק
(ב) הדיסק המדומה חייב להיות קיים
(ג) הערך של השדה mounted חייב להיות false – כלומר, פונקציה זו יכולה להתבצע רק אם התבצעה קודם לכן unmountdisk.
בדומה למה שעושים כאשר יוצרים דיסק עם createdisk, יצטרך להתבצע mountdisk על מנת שהדיסק יהיה זמין.

- **fstream* getdskfl()**

תפקידה של פונקציה זו להחזיר מצביע מסוג fstream לכתובת של dskfl שמייצג את הקובץ שמכיל את הדיסק המדומה. יוחזר מצביע ריק במקרה של בעיה כלשהי.

- **void seekToSector(unsigned int)**

תפקידה של פונקציה זו להתמקם בסקטור המבוקש, בדיסק המדומה.
הפרמטר הוא מספר שלם וחיובי; מספר זה אמור להיות המספר הסידורי של הסקטור המבוקש בדיסק המדומה.
הערה: כתוצאה מביצוע הפונקציה הזאת, הסקטור המבוקש הופך להיות הסקטור הנוכחי; כלומר, מייד אחרי ביצוע הפונקציה הזאת, יהיה אפשר לקרוא או לכתוב את הסקטור המבוקש.

- **void writeSector(unsigned int, Sector*)**

תפקידה של פונקציה זו לכתוב מהחוצץ שאליו מצביע הפרמטר השני, לתוך הסקטור המבוקש בדיסק המדומה.
הפרמטר הראשון הוא מספר שלם וחיובי; מספר זה אמור להיות המספר הסידורי של הסקטור המבוקש בדיסק המדומה.
הפרמטר השני הוא מצביע לחוצץ באורך של סקטור; כלומר, השטח שאליו מצביע הפרמטר הזה הוא רצף/מערך של תווים (שאינו מחרוזת!) המשמש כחוצץ.
הערה: מייד אחרי הכתיבה, הסקטור הנוכחי יהיה זה שמייד אחרי הסקטור שמספרו הסידורי התקבל כפרמטר ראשון.

- **void writeSector(Sector*)**

הפרמטר הוא מצביע לחוצץ באורך של סקטור; כלומר, השטח שאליו מצביע הפרמטר הזה הוא רצף/מערך של תווים (שאינו מחרוזת!) המשמש כחוצץ.
תפקידה של פונקציה זו לכתוב מהחוצץ שאליו מצביע הפרמטר, לתוך הסקטור הנוכחי בדיסק המדומה.

הערה: מייד אחרי הכתיבה של הסקטור שהתקבל כפרמטר, השדה currDiskSectorNr יקודם באחד.

- **void readSector(int, Sector*)**

הפרמטר הראשון הוא מספר שלם וחיובי; מספר זה אמור להיות המספר הסידורי של הסקטור המבוקש בדיסק המדומה.
הפרמטר השני הוא מצביע לחוצץ באורך של סקטור; כלומר, השטח שאליו מצביע הפרמטר הזה הוא רצף/מערך של תווים (שאינו מחרוזת!) המשמש כחוצץ.
תפקידה של פונקציה זו לקרוא את הסקטור המבוקש מהדיסק המדומה לתוך החוצץ שאליו מצביע הפרמטר השני.
הערה: מייד אחרי הקריאה, הסקטור הנוכחי יהיה זה שמייד אחרי הסקטור שמספרו הסידורי התקבל כפרמטר ראשון.

- **void readSector(Sector*)**

הפרמטר הוא מצביע לחוצץ באורך של סקטור; כלומר, השטח שאליו מצביע הפרמטר הזה הוא רצף/מערך של תווים (שאינו מחרוזת!) המשמש כחוצץ.
תפקידה של פונקציה זו לקרוא את הסקטור הנוכחי מהדיסק המדומה לתוך החוצץ שאליו מצביע הפרמטר.
הערה: מייד אחרי הקריאה של הסקטור הנוכחי לתוך החוצץ שהתקבל כפרמטר, השדה currDiskSectorNr יקודם באחד.

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

שלב זה הוא המשך טבעי של השלב הקודם. משימתך כאן להגדיר וליישם פונקציות תשתית נוספות לאלו שכבר מימשת בשלב הקודם, במסגרת המחלקה disk. מה שמאפיין את כל הפונקציות שעליך לממש כאן, שהן ניתנות לביצוע בתנאי שכבר קיים דיסק מדומה מהסוג שמבנהו תואר **כאן.**

- **בקשר לפונקציות ליישום בשלב הזה**
  - **void format(string &)**
    תפקידה של פונקציה זו לעשות פירמוט לדיסק המדומה.
    הפרמטר הוא שם בעל הדיסק.
    אם שם בעל הדיסק הרשום ב-vhd שונה מהערך של הפרמטר, לא יהיה אפשר לבצע את הפעולה ותיווצר תקלה (exception) באמצעות מנגנון ה--try-throw-catch; כלומר, בעל הדיסק הוא היחיד שיכול לבצע פירמוט לדיסק הספיציפי הזה.
    על מנת להשיג את משימתה, הפונקציה הזאת תבצע
    (א) איתחול ה-DAT: כל הביטים של ה-DAT המייצגים cluster-ים של נתונים יקבלו ערך 1 (משמעות הדבר שכל ה-cluster-ים של נתונים פנויים), וכל אלה המייצגים cluster-ים של מידע מבני של הדיסק המדומה, יקבלו ערך 0.
    (ב) איתחול התיקייה הראשית: כל כניסותיה של התיקייה הראשית יסומנו כריקות (כלומר, השדה entryStatus בכל אחת מהכניסות של התיקייה יקבל ערך 0 המעיד על כניסה ריקה).
    אם הייתה בעיה בביצועה, הפונקציה תיצור תקלה (exception) מתאימה באמצעות מנגנון ה-try-throw-catch
  - **int howmuchempty( )**
    תפקידה של פונקציה זו להחזיר את מספר סה"כ ה-cluster-ים הפנויים בדיסק המדומה.
  - **void alloc(DATtype &, unsigned int, unsigned int)**
    תפקידה של פונקציה זו לטפל בהקצאת שטח (לקובץ) בדיסק.
    הפרמטר *הראשון* חייב להיות ערך מסוג DATtype דימוי FAT, אליו מבקשים להקצאות שטח.
    הפרמטר *השני* חייב להיות מספר חיובי שקובע כמה *סקטורים* מבוקשים להקצאה.
    הפרמטר *השלישי* מייצג סוג האלגוריתם להקצאת שטח. פרמטר זה חייב להיות אחד משלושה ערכים: 0 – first fit, 1 – best fit, 2 – worst fit (בקורס התיאורטי למדנו על שלושת האלגוריתמים האלה).
    הקצאת השטח המבוקש תתבצע כדלהלן:
    א. מתבצע חיפוש שטח מתאים ב-DAT על מנת לספק את השטח המבוקש, תוך כדי הקפדה מרבית על יעילות גישה לפי [צילינדר, מסילה,סקטור].
    ב. אם א. הצליח, ה-cluster-ים שהוקצאו מסומנים בהתאם ב-DAT וגם ב-FAT שהתקבל כפרמטר ראשון.
    ג. אם א. לא הצליח, תיווצר תקלה (exception) מתאימה באמצעות מנגנון ה-try-throw-catch.
    רמז: הערך החדש של ה-DAT יהיה תוצאה של פעולה בוליאנית מסוימת בין ה-DAT לבין ה-FAT. למדנו על זה בקורס התיאורטי.

  - **void allocextend(DATtype &, unsigned int, unsigned int)**
    תפקידה של פונקציה זו לטפל בהוספת הקצאת שטח (לקובץ) בדיסק.
    בדומה למה שנאמר בהגדרת הפונקציה alloc, דלעיל, משמעות הפרמטרים כדלקמן:
    הפרמטר *הראשון* חייב להיות ערך מסוג DATtype דימוי FAT, אליו מבקשים להוסיף הקצאת שטח.

הפרמטר *השני* חייב להיות מספר חיובי שקובע כמה *סקטורים* מבוקשים להוסיף להקצאה הקיימת.
הפרמטר *השלישי* מייצג סוג האלגוריתם להקצאת שטח (ראה בהגדרת הפונקציה alloc דלעיל).
על מנת להקצאות את השטח, פונקציה זו פועלת בדיוק כמו הפונקציה alloc; ההבדל היחיד הוא שבמקרה של הצלחה בהקצאה, הערך מסוג DATtype שהתקבל כפרמטר ראשון, והוא דימוי FAT, מוחזר עם סימונים נוספים מתאימים (כמובן שב-DAT זה גם בא לידי ביטוי באמצעות סימונים נוספים).
רמז: הערך החדש של ה-DAT יהיה תוצאה של פעולה בוליאנית מסוימת בין ה-DAT לבין ה-FAT. למדנו על זה בקורס התיאורטי.

- **void dealloc(DATtype &)**

תפקידה של פונקציה זו לשחרר שטח תפוס (ע"י קובץ מסוים).
הפרמטר הינו ערך מסוג DATtype, דימוי FAT, שמבקשים להחזיר לשטח הפנוי.
שחרור השטח יתבצע כך שכל ה--cluster-ים המסומנים כתפוסים ב-FAT שהתקבל כפרמטר, יסומנו כפנויים בו וגם ב-DAT.
רמז: הערך החדש של ה-DAT יהיה תוצאה של פעולה בוליאנית מסוימת בין ה-DAT לבין ה-FAT. למדנו על זה בקורס התיאורטי.

- **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. יכול להיות מאוד טוב לזמן ההדגמה של כל מה שנעשה עד השלב הזה, שתהיינה פונקציות עזר שנותנות אינפורמציה על השטח הכללי, השטח התפוס, השטח הפנוי, מצב ה-DAT, וכו'. בכל זאת, אנו משאירים ליצירתיות שלכם להחליט איזה פונקציות מן הראוי שתהיינה בקטגוריה הזאת.

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

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

משימתך בשלב זה של המיני-פרויקט, ליישם יצירת והרחבת קובץ תוך הקצעת שטח דיסק בשבילו, ומחיקת קובץ תוך החזרת השטח שהוא תפס, לשטח הפנוי של הדיסק. בשלב הזה מסתכלים על קובץ כרצף של סקטורים, ללא התייחסות לרמה הלוגית שבה מסתכלים על קובץ כרצף של רשומות.

- **בקשר לפונקציות ליישום בשלב הזה (פונקציות השייכות למחלקה disk)**

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

תפקידה של פונקציה זו לבצע יצירת קובץ נתונים (שאינו תיקיה). על מנת לבצע את משימתה, הפונקציה הזאת בודקת פרמטרים, מכניסה את הקובץ לתיקיה, קוראת ל- alloc על מנת להקצאות את השטח הדרוש לקובץ, נותנת ערך 1 לשדה entryStatus, וכו'.
הפרמטר *הראשון* קובע את שם הקובץ (חייב להיות *לא* קיים).
הפרמטר *השני* קובע את שם בעל הקובץ.
הפרמטר *השלישי* קובע את סוג קובץ ("F" – עם רשומות בעלות אורך קבוע; "V" – עם רשומות בעלות אורך משתנה).
הפרמטר *הרביעי* קובע את אורך הרשומה במקרה שערך הפרמטר השלישי הוא "F"); הפרמטר הזה קובע את האורך המרבי של הרשומה במקרה שערך הפרמטר השלישי הוא "V". במקרה של "V", לכל רשומה יוצמד שדה לפניה, מסוג unsigned int, שיעיד על האורך בפועל של הרשומה.
הפרמטר *החמישי* קובע את מספר הסקטורים המבוקשים לקובץ שפונקציה זו יוצרת.
הפרמטר *השישי* קובע את טיפוס הנתונים של המפתח.
הפרמטר *השביעי* קובע את ה-offset של התחלת המפתח בתוך הרשומה.
הפרמטר *השמיני* קובע את אורך המפתח (הוא אופציונלי; הוא חובה רק אם מדובר במחרוזת).
בנוסף לכל הנ"ל, הפונקציה תיקח את תאריך היום ממערכת ההפעלה כתאריך יצירת הקובץ.
<u>הערה</u>: אם כבר קיים קובץ בעל אותו שם, או אין מקום בתיקייה, או אין מספיק מקום להקצאת השטח הדרוש, הקובץ *לא נוצר*, והפונקציה יוצרת exception מתאים באמצעות מנגנון ה-try-throw-catch של שפת ++C.

- **void delfile(string &, string &)**

תפקידה של פונקציה זו לבצע מחיקה של קובץ קיים. על מנת לבצע את משימתה, הפונקציה הזאת בודקת פרמטרים, מוחקת את הקובץ מהתיקייה, קוראת ל-dealloc על מנת להחזיר את השטח שתפוס ע"י הקובץ, נותנת ערך 2 לשדה entryStatus, וכו'.
הפרמטר *הראשון* קובע את שם הקובץ (שחייב להיות קיים)
הפרמטר *השני* קובע את שם המשתמש שמבקש למחוק את הקובץ.
<u>הערה</u>: רק בעל הקובץ יוכל למחוק אותו. אם מישהו שאינו בעל הקובץ, או אם הקובץ לא קיים, הפונקציה יוצרת exception מתאים באמצעות מנגנון ה--try-throw-catch של שפת ++C.

- **void extendfile(string &, string &, unsigned int)**

תפקידה של פונקציה זו לבצע הוספת שטח לקובץ קיים. על מנת לבצע את משימתה, הפונקציה הזאת בודקת פרמטרים, מעדכנת את הכניסה של הקובץ בתיקיה הנוכחית, קוראת ל- allocextend להקצאת השטח הנוסף הדרוש לקובץ, וכו'.
הפרמטר *הראשון* קובע את שם הקובץ (שחייב להיות קיים)
הפרמטר *השני* קובע את שם המשתמש שמבקש להוסיף שטח לקובץ.
הפרמטר *השלישי* קובע את מספר הסקטורים שהמשתמש מבקש להוסיף.
<u>הערה</u>: אם לא קיים קובץ בעל אותו שם, או אין מספיק מקום להקצאת השטח הנוסף הדרוש, או המבקש אינו בעל הקובץ, הפונקציה יוצרת exception מתאים באמצעות מנגנון ה-try-throw-catch של שפת ++C.

- **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 מן הראוי שתהיינה בקטגוריה הזאת.

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

בס"ד

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

בשלב הקודם יישמת את התשתית לטיפול בקבצים, מבחינה מבנית, פיזית, ברמת הדיסק.
משימתך בשלב זה של המיני-פרויקט להגדיר וליישם מחלקה בשם FCB (File Control Block) שתאפשר לטפל בקובץ מבחינת התוכן שבו, מבחינה לוגית, כרצף של רשומות. למעשה, כל אובייקט מסוג זה יאפשר לנהל את כל פעולות הקלט/פלט הלוגיות (לפי רשומות) הקשורות לפתיחה כלשהי של קובץ שבדיסק.
הגדרת המבנה של המחלקה FCB כדלקמן:

שם שדה	טיפוס	משמעות	אורך השדה
d	disk*	מצביע לאובייקט מסוג disk שמייצג דיסק מדומה. אם המצביע ריק, משמעות הדבר שאובייקט זה עדיין לא בשימוש לניהול קובץ.	4 bytes
fileDesc	dirEntry	העתק של ה-fileDesc (file descriptor) של ה-File Header של הקובץ (ולא של כניסת הקובץ בתיקייה)	72 bytes
FAT	DATtype	FAT	200 bytes
Buffer	sector	חוצץ קלט/פלט (I/O Buffer)	1024 bytes
currRecNr	unsigned int	מספר סידורי של הרשומה הנוכחית, בתוך הקובץ	4 bytes
currSecNr	unsigned int	מספר סידורי של הסקטור הנוכחי, בתוך הקובץ	4 bytes
currRecNrInBuff	unsigned int	מספר סידורי של הרשומה הנוכחית, בתוך הסקטור שבחוצץ	4 bytes

דלקמן רשימת פונקציות הנדרשות בשלב הזה של המיני-פרויקט.

- **FCB *openfile(string &, string &, string &)**

<u>פונקציה זו שייכת למחלקה disk</u> ותפקידה לבצע פתיחה של קובץ על כל מה שמשתמע מכך (בהתאם למה שלמדנו בקורס התיאורטי). תוצאה חשובה של ביצוע הבקשה לפתיחת קובץ היא האתחול של ה-"מצביע" קריאה/כתיבה של הפתיחה. אתחול זה מתבטא בפועל באתחול שלושת המשתנים המאפשרים ניהול הקלט/פלט הפיזי והלוגי של הקובץ (מדובר במשתנים currSecNr, currRecNr, currRecNrInBuff). אם אופן הפתיחה "I", "O", או "IO", ה-"מצביע" קריאה/כתיבה יהיה ממוקם בהתחלת הקובץ (ברשומה מס' 0); (ב) אם אופן הפתיחה "E", ה-"מצביע" קריאה/כתיבה יהיה ממוקם בסוף הלוגי של הקובץ.
הפרמטר *הראשון* הוא שם הקובץ (שחייב להיות קיים).
הפרמטר *השני* הוא שם המשתמש שמבקש לפתוח את הקובץ.
הפרמטר *השלישי* הוא אופן הפתיחה המבוקש (קלט – "I", פלט – "O", קלט/פלט – "IO", הוספה – "E")
<u>הערך המוחזר</u>: מצביע לאובייקט מסוג FCB שפונקציה זו יוצרת באופן דינאמי, ומאתחלת אותו בהתאם לערכי הפרמטרים שהתקבלו.
<u>הערות:</u>

- הפונקציה תחפש בתיקייה כניסה מתאימה לפרמטרים שהתקבלו. הפונקציה תקרא את ה-File Header ותעתיק את ה-FileDesc שלו ואת ה-FAT לשדות המתאימים באובייקט FCB. הפונקציה תעדכן את השדות המתאימים בכניסה שבתיקייה.
- <u>פתיחת קובץ לקריאה (I) בלבד</u>: כל משתמש יוכל לפתוח אותו באופן הפתיחה הזה. כלומר, לכל משתמש מותר לפתוח כל קובץ לקריאה.
- <u>פתיחת קובץ לכתיבה (O) בלבד, להוספה (E) או לקריאה/כתיבה (IO)</u> : אופני פתיחה אלה יהיו מותרים רק לבעל הקובץ.
- כל מצב שמנוגד לתנאי כלשהו מהנ"ל, יגרום ל-exception מתאים.

<u>הערה</u>: <u>יתר הפונקציות בשלב הזה של מיני-פרויקט, כולן שייכות למחלקה FCB</u>.

- **default constructor – FCB()**

הבנאי ברירת מחדל של FCB יצור אובייקט מסוג FCB ויאתחל את כל השדות עם ערכים התחלתיים רלוונטיים. במיוחד חשוב להבהיר ש-d מסוג מצביע ל-disk, יאותחל ל-NULL.

- **constructor – FCB(disk *)**

בנאי זה של FCB יצור אובייקט מסוג FCB, השדה d יאותחל עם המצביע מסוג disk שהתקבל כפרמטר, ויתר השדות יאותחלו עם ערכים רלוונטיים.

- **destructor – ~FCB()**

חשוב להזכיר שתפקיד מרכזי של פונקציה משמידה הוא לשחרר כל מה שהוקצא באופן דינאמי.

- **void closefile()**

תפקידה של פונקציה זו לסגור את הקובץ הספציפי. משמעות הדבר: כתיבה פיזית של תוכן החוצץ באמצעות הפונקציה flushfile (ראה בהמשך), עדכון כל מה שצריך לעדכן ב-File Header (כולל כתיבת ה-EOF, אם יש צורך בכך), הפיכת האובייקט FCB לזמין לשימוש חוזר על ידי פתיחה עתידית אחרת (השדה d מסוג *disk יקבל מצביע ריק), עדכון שדות אחרים באובייקט FCB לפי הצורך, וכו'.

- **void flushfile()**

תפקידה של פונקציה זו לכתוב פיזית את תוכן החוצץ בחזרה לסקטור המתאים אם הוא עבר שינוי מאז אותו סקטור נקרא לתוך החוצץ.

<u>הערות חשובות</u>:

- לפונקציה זו יש משמעות רק על קובץ שנפתח לפלט, לקלט/פלט, או להוספה.

- **void read(char *, [unsigned int])**

תפקידה של פונקציה זו לקרוא את הרשומה הנוכחית של הקובץ. אם ערך הפרמטר השני הוא 0, המשמעות היא לקריאה בלי כוונת עדכון. אם ערך הפרמטר השני הוא 1, המשמעות היא לקריאה לצורך עדכון (read for update). הפרמטר השני אופציונאלי (ערך ברירת המחדל שלו הוא 0). מיד אחרי קריאת הרשומה, הרשומה שאחריה הופכת להיות הנוכחית שיהיה אפשר לקרוא (או לכתוב) עם ה-read (או ה-write) הבא.

<u>הערות חשובות</u>:

- בקשת קריאה מקובץ שנפתח לכתיבה, היא טעות.
- הקריאה באה לידי ביטוי על ידי העתקה של הרשומה הנוכחית מהחוצץ לכתובת שהתקבלה כפרמטר.
- במקרה שהקובץ נפתח לקלט בלבד (או אם ערך הפרמטר השני הוא 0), אחרי ביצוע פעולה זו הרשומה הנוכחית היא זאת שמיד אחרי הרשומה שנקראה.
- במקרה שהקובץ נפתח לקלט/פלט, וערך הפרמטר השני הוא 1, אחרי ביצוע פעולה זו, הרשומה הנוכחית היא זאת שנקראה (ולא זאת שאחריה!!) והיא הופכת לנעולה עד שהנעילה משתחררת כחלק מביצוע של update, delete, או updateCancel (ראה בהמשך). <u>עד אחרי שחרור הרשומה מהנעילה, המערכת תמנע מהיישום לבצע כל פעולה שהיא על הקובץ</u>.
- בזמן שרשומה נעולה, אם היישום ינסה לבצע פעולה כלשהי על הקובץ, שאינה update, delete, או updateCancel, זה יגרום ל- exception מתאים.

- **void write(char *)**

תפקידה של פונקציה זו לכתוב את הרשומה שהתקבלה בפרמטר הראשון, לתוך המיקום הנוכחי של הקובץ.

<u>הערות חשובות</u>:

- בקשת כתיבה לקובץ שנפתח לקריאה, היא טעות.
- הכתיבה באה לידי ביטוי על ידי העתקה של הרשומה שהתקבלה כפרמטר לתוך המקום הנוכחית בחוצץ. אחרי ביצוע פעולה זו, הרשומה הנוכחית היא זאת שמיד אחרי הרשומה שנכתבה. יהיו מקרים שכתיבת הרשומה תגרום לכתיבה פיזית (flush) של הסקטור שבחוצץ, למקומו בקובץ בדיסק.

- **void seek(unsigned int, int)**

תפקידה של פונקציה זו לאפשר "לקפוץ" קדימה או אחורה מהמקום הנוכחי למקום אחר בתוך הקובץ. ברור שהבקשה "לקפוץ" מספר רשומות קדימה או אחורה תלויה בערך של הפרמטר השני. מיד אחרי ה-"קפיצה", המקום שאליו הגענו בקובץ הופך להיות המיקום של הרשומה הנוכחית שממנו נוכל לקרוא רשומה או לתוכו נוכל לכתוב רשומה.

הפרמטר *הראשון* הוא מספר שמסמל את סוג החיפוש (0 – מהתחלת הקובץ, 1 – מהרשומה הנוכחית, 2 – מהסוף הלוגי של הקובץ).

הפרמטר *השני* הוא מספר שלם המציין כמה *רשומות* המשתמש רוצה "לקפוץ" קדימה או אחורה (ערך חיובי מתכוון לקפיצה קדימה וערך שלילי מתכוון לקפיצה אחורה).

<u>הערות חשובות:</u>

- <u>פונקציה זו רלוונטית באופן מלא רק למקרה של קובץ בעל רשומות באורך קבוע ("F")</u>. בקשה לקפיצה קדימה כאשר הערך של הפרמטר הראשון 2 היא טעות, ובקשה לקפיצה אחורה כאשר הערך של הפרמטר הראשון 0 היא גם טעות. בשני המקרים זה יגרום ל-exception.
- <u>במקרה של קובץ בעל רשומות באורך משתנה ("V"), רק שתי קפיצות ניתנות לביצוע</u>: (א) קפיצה (0,0), להתחלת הקובץ, ו-(ב) קפיצה (2,0), לסוף הלוגי של הקובץ. כל קפיצה אחרת לא מוגדרת וניסיון לבצע כזאת יגרום ל-exception.
- אם הקפיצה לרשומה המבוקשת גורמת לקפיצה לסקטור שונה מזה שבחוצץ, זה יגרום ל-flush של החוצץ ולאחר מכן קריאה של הסקטור המבוקש לתוך החוצץ, והוא הופך להיות הסקטור הנוכחי שבתוכו נמצא מיקום הרשומה שהופכת לרשומה הנוכחית.

<u>הערה: שלושת הפונקציות הבאות, delete, update, ו-updateCancel ניתנות להפעלה רק על רשומה שנקראה לצורך עדכון ע"י הפונקציה read במצב read-for-update</u>.

- **void updateCancel()**

תפקידה של פונקציה זו לבטל את מצב הנעילה של הרשומה הנוכחית, שנוצר כתוצאת הביצוע של "קריאה לצורך עדכון".

<u>הערות חשובות:</u>

- כמובן שבקשה כזאת לקובץ שנפתח לקריאה, או לכתיבה, בלבד היא טעות.
- פונקציה זו תוכל להתבצע רק אם הרשומה הנוכחית נקראה קודם לכן לעדכון (read-for-update).
- אחרי ביצוע פעולה זו, הרשומה הנוכחית היא זאת שהייתה לפני ביצועה.

- **void delete()**

תפקידה של פונקציה זו למחוק רשומה במקום הנוכחי בקובץ. כמובן שבקשה כזאת לקובץ שנפתח לקריאה, או לכתיבה, בלבד היא טעות. הכוונה היא למחיקה לוגית; כלומר, הרשומה לא תבוטל פיזית, אלא שדה המפתח יימחק עם אפסים בינאריים, מה שיסמל מחיקה של הרשומה כולה.

<u>הערות חשובות:</u>

- פונקציה זו תוכל להתבצע רק אם הרשומה הנוכחית נעולה (היא נקראה קודם לכן לעדכון). אם תנאי זה לא מתקיים, תיגרם exception.
- פונקציה זו תוכל להתבצע רק אם הרשומה הנוכחית נעולה.
- לאחר ביצועה של פונקציה זו, הנעילה משתחררת.
- <u>אחרי ביצוע פעולה זו הרשומה הנוכחית היא זאת שמיד אחרי הרשומה שבוטלה</u>.

- **void update(char *)**

תפקידה של פונקציה זו לכתוב רשומה במקום הנוכחי בקובץ, לצורך עדכון.

הפרמטר הוא מצביע לרשומה שתכנית היישום מבקשת לכתוב כעדכון לרשומה הנוכחית.

<u>הערות חשובות:</u>

- בקשה כזאת לקובץ שנפתח לקריאה, או לכתיבה, בלבד היא טעות.
- פונקציה זו תוכל להתבצע רק אם הרשומה הנוכחית נעולה.
- <u>אחרי ביצוע פעולה זו, הרשומה הנוכחית היא זאת שמיד אחרי הרשומה שעודכנה.</u>

- לאחר ביצוע של פונקציה זו, הנעילה משתחררת.

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

בשלב הקודם יישמת פונקציות המאפשרות לטפל בקובץ ברמת התוכן שבו: יכולנו לפתח ולסגור קובץ באופנים שונים, יכולנו לקרוא ולכתוב רשומות בקובץ, יכולנו "לטייל" בין רשומות של הקובץ, ויכולנו לעדכן וגם למחוק רשומה.
פרט לפונקציה openfile ששייכת למחלקה disk ומחזירה אובייקט מסוג FCB שמייצג קובץ פתוח, כל יתר הפונקציות שכתבנו בשלב הקודם הן מימוש של פעולות שניתן לבצע על התוכן של קובץ; לכן, טבעי שכל אלה תהיינה שייכות למחלקה FCB.
כרגע יש לנו את כל מה שצריך על מנת להשלים את בניית המערכת שלנו לניהול דיסק מדומה, וזה ייעשה באמצעות שכבת תכנה נוספת שבאה לידי ביטוי באמצעות מחלקה בשם **DMS (Disk Management System)**.
במבנה הכולל של המערכת לניהול דיסק מדומה, שמיושם באמצעות מחלקה בשם DMS (Disk Management System), למחלקה FCB תפקיד מרכזי: ישנו מערך של אובייקטים מסוג FCB שיאפשרו לטפל במספר קבצים כמספר האיברים במערך. כל איבר במערך הזה מייצג קובץ פתוח במערכת. כפי שלמדנו בקורס התיאורטי, המערך הזה (א) מאפשר לממש את רעיון ניהול ה-System I/O Buffers שמשוייכים לקבצים בדיסק מייד עם פתיחתם, ו-(ב) מאפשר לממש את רעיון רשימת הקבצים הפתוחים ברגע נתון במערכת.
הגדרת המבנה של המחלקה DMS כדלקמן:

שם שדה	טיפוס	משמעות	אורך השדה
fcbArray	FCB*	מצביע לאובייקט מסוג FCB שישמש למערך של אובייקטים מסוג FCBמיישם דיסק מדומה. אם המצביע	
fcbArrSize	unsigned int	אורך המערך של אובייקטים מסוג FCB	72 bytes

דלקמן רשימת פונקציות הנדרשות בשלב הזה של המיני-פרויקט.

- **default constructor – DMS()**

הבנאי ברירת מחדל של DMS ויאתחל את המצביע fcbArray עם מערך של חמישה אובייקטים מסוג FCB (במצב ברירת המחדל שלהם) שיוקצאו באופן דינאמי. בנוסף לזה, השדה fcbArrSize יאותחל בערך 5.

- **constructor – DMS(unsigned int)**

הבנאי הזה יאתחל את המצביע fcbArray עם מערך של אובייקטים מסוג FCB (במצב ברירת המחדל שלהם) שיוקצאו באופן דינאמי. בנוסף לזה, השדה fcbArrSize יאותחל בערך של הפרמטר שהתקבל.

- **destructor – ~DMS()**

חשוב להזכיר שתפקיד מרכזי של פונקציה משמידה הוא לשחרר כל מה שהוקצא באופן דינאמי.

- **FCB *openfile(disk*, string &, string &, string &)**

הפרמטר *הראשון* הוא מצביע לאובייקט מסוג disk, שמבקשים לפתוח קובץ שבתוכו.
יתר הפרמטרים הם בדיוק אותם אלה שבפונקציה המקבילה במחלקה disk.
כמקבילה שלה במחלקה disk, תפקידה של פונקציה זו לבצע פתיחה של קובץ. הפונקציה הזאת תקרא לפונקציה lookforfcb (ראה בהמשך) על מנת לחפש במערך fcbArray אובייקט FCB זמין לשימוש (שהמצביע מסוג *disk ריק). לאחר מכן, תיתן את הערך של הפרמטר הראשון למצביע d שבאובייקט FCB שהפונקציה lookforfcb מצאה ותקרא לפונקציה openfile של אותו דיסק על מנת לפתוח את הקובץ בפועל. המצביע מסוג FCB שמוחזר על ידי הפתיחה בפועל יוחזר על ידי הפונקציה הזאת.

- **FCB *lookforfcb(disk*, string &)**

הפרמטר *הראשון* הוא מצביע לאובייקט מסוג disk, שמבקשים לפתוח קובץ שבתוכו.
הפרמטר *השני* הוא שם של קובץ.
תפקידה של פונקציה זו לסרוק את המערך fcbArray ולהחזיר אובייקט FCB פנוי. לפני שהפונקציה מחזירה אובייקט כזה היא חייבת לבדוק שקובץ בעל אותו שם, באותו דיסק, לא פתוח. אם הכל תקין אבל אין איבר פנוי, הפונקציה תחזיר מצביע ריק.
<u>הערות</u>:
- מובן מאליו שמספר הקבצים שיוכלו להיות פתוחים בו-זמנית במערכת יהיה מוגבל לאורך המערך fcbArray.
- אם קובץ כבר פתוח באופן פתיחה כלשהו, אסור שתתקבל פתיחה נוספת כלשהי של אותו קובץ. רק אחרי שהקובץ ייסגר יהיה אפשר לפתוח אותו שוב.
- כל מצב שמנוגד לתנאי כלשהו מהנ"ל, יגרום ל-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.

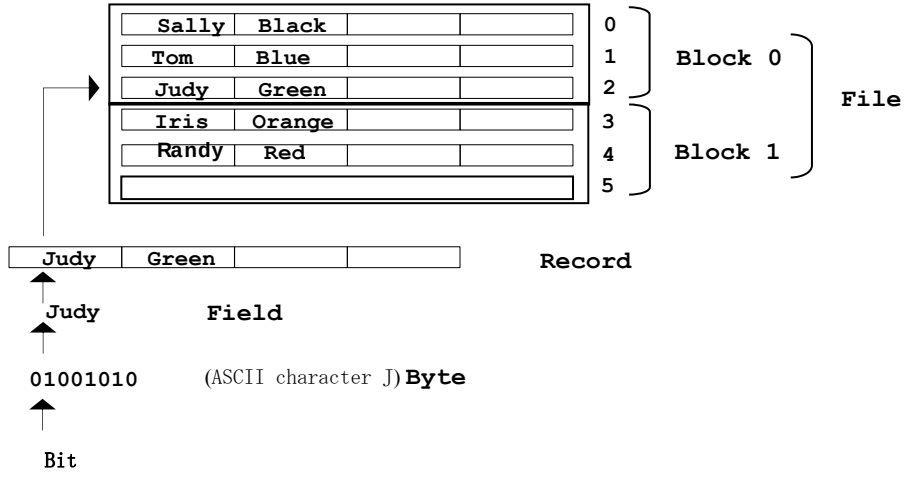
בס"ד

תאור של המשימות שעליך לבצע בשלב מס' 5 של המיני-פרוייקט

משימתך בשלב זה של המיני-פרוייקט, ליישם ממשק גראפי שישמש כ-main של מערכת לניהול דיסק מדומה.
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.

בס"ד

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

הערות חשובות:

(א) 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.

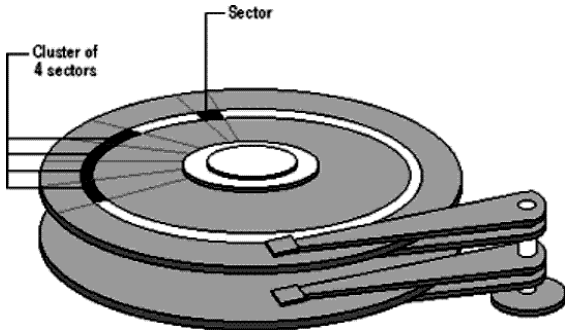
(ב) כמו כן, אם שדה מפתח של רשומה בקובץ מוגדר כמחרוזת, הוא גם יפורש כמחרוזות בסגנון שפת C.

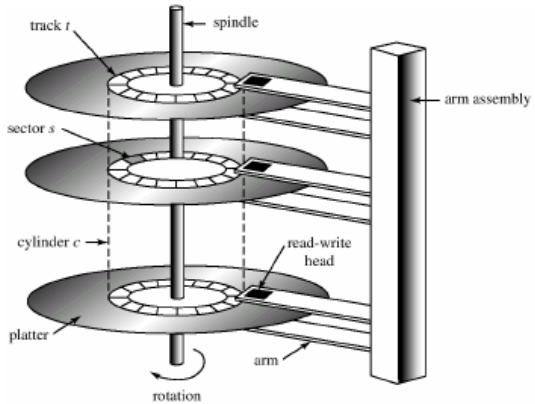
(ג) חשוב לציין שכל הקבצים שייווצרו ע"י מערכת ניהול הקבצים שנפתח במיני-פרויקט הזה, יהיו קבצים רגילים מבחינת מערכת ההפעלה המארחת (במקרה שלנו, Windows); כלומר, מערכת ניהול הקבצים שלך לא צריכה לדאוג לניהול FAT/DAT כי זאת אחראיות של מערכת ההפעלה המארחת.

בס"ד

תאור מפורט של מבנה הדיסק

באופן כללי, לדיסק קשיח יש המבנה הבא:





סקטור (Sector): יחידת קלט/פלט של מידע
אשכול (Cluster): יחידת הקצאת שטח, שהיא קבוצת סקטורים המופיעים ברצף ב*אותה* מסילה.

הכתובת הפיסית של כל סקטור בדיסק היא השלישייה [צילינדר, מסילה, סקטור].
הצורה הטבעי לגשת לכל סקטור באופן סידרתי בדיסק היא לפי צילינדר, ובתוכו לפי מסילה, ובתוכה לפי סדר הופעתם של הסקטורים באותה מסילה.
לפי הסדר הזה, ניתן להסתכל על דיסק משתי נקודות מבט:
(א) מבחינת קלט/פלט של מידע, דיסק הוא מערך של סקטורים ממוספרים בסדר עולה, מאפס ועד הסקטור האחרון בדיסק, שניתן לגשת לכל אחד מהם באופן ישיר לפי המספר הסידורי של הסקטור מהתחלת הדיסק.
(ב) מבחינת הקצאת שטח, דיסק הוא מערך של אשכולות ממוספרים בסדר עולה, מאפס ועד האשכול האחרון בדיסק.

בדיסק שאנו נממש את הסימולציה שלו יש 2 משטחים פעילים שלכל אחד מהם יש 200 מסילות, כאשר הקיבולת של כל מסילה היא 8192 בתים.
כל מסילה מחולקת ל-8 סקטורים, כאשר לכל סקטור יש כתובת, והיא מספרו הסידורי מהתחלת הדיסק; כתובת הסקטור הראשון היא 0.

- יחידת הקצאת שטח (cluster) היא 2 סקטורים רצופים.
- יחידת קלט/פלט היא סקטור אחד בלבד

ממה שתואר לעיל ברור שלדיסק מסוג זה יש 3200 סקטורים שהם 1600 cluster-ים.

מבנה הדיסק הוא כדלקמן:

- לכל סקטור המבנה הבא:

שם השדה	טיפוס	משמעות	אורך השדה
sectorNr	unsigned int	מספר סידורי של הסקטור בדיסק	4 bytes
rawData	char[1020]	נתונים	1020 bytes

- יש סקטורים בדיסק המיועדים למידע מבני (Volume Header, DAT ו-Root Directory) וכל היתר מיועדים לנתונים (ראה בהמשך הדף הזה).

- הסקטור הראשון (סקטור מס' 0) של הדיסק מכיל את תווית הזיהוי שלו - Volume Header. לתווית הזיהוי המבנה הבא:

שם השדה	טיפוס	משמעות	אורך השדה
sectorNr	unsigned int	מספר סידורי של הסקטור בדיסק	4 bytes
diskName	char[12]	שם זיהוי הדיסק	12 bytes
diskOwner	char[12]	שם בעל הדיסק	12 bytes
prodDate	char[10]	תאריך יצור הדיסק (ddmmyyyy)	10 bytes
ClusQty	unsigned int	סה"כ יחידות הקצאה (clusters) בדיסק	4 bytes
dataClusQty	unsigned int	מספר יחידות הקצאה לנתונים בלבד.	4 bytes
addrDAT	unsigned int	כתובת הסקטור שמכיל את ה-DAT	4 bytes
addrRootDir	unsigned int	כתובת ה-cluster שמכיל את התיקייה הראשית (Root Directory)	4 bytes
addrDATcpy	unsigned int	כתובת הסקטור שמכיל עותק שני של ה-DAT	4 bytes
addrRootDirCpy	unsigned int	כתובת ה-cluster שמכיל עותק שני של התיקייה הראשית (Root Directory)	4 bytes
addrDataStart	unsigned int	כתובת ה-cluster הראשון בדיסק המיועד לנתונים.	4 bytes
formatDate	char[10]	תאריך פירמוט (ddmmyyyy)	10 bytes
isFormated	bool	האם כבר מפורמט? (כן / לא)	1 byte
emptyArea	char[947]	שמור לשימוש עתידי	947 bytes

- מיקום ברירת המחדל המיועד לאחסון ה-DAT (Disk Allocation Table) הוא הסקטור השני (סקטור מס' 1) של הדיסק; למעשה, הדבר היחיד שיהיה מאוחסן באותו סקטור הוא ה-DAT. כלומר, ה-cluster מס' 0 מכיל את ה-Volume Header בסקטור הראשון שלו ואת ה- DAT בסקטור השני.
- התיקייה הראשית (Root Directory) של הדיסק תופסת שני סקטורים צמודים, השייכים לאותו cluster. המיקום ברירת המחדל ל-cluster של התיקייה הראשית הוא ה-cluster מס' 1.
- בזמן יצירת הדיסק השדות addrDAT ו-addrRootDir של ה-Volume Header יהיו מאותחלים עם ערכי ברירת המחדל שלהם. בזמן אתחול (format) הדיסק יהיה ניתן להחליט על מיקום אחר.
- שים לב שמטעמי ביטחון, חייב להיות עותק שני של ה-DAT ועותק שני של התיקייה הראשית במקום אחר בדיסק.
- ה-DAT הוא רצף של ביטים (אובייקט מסוג bitset) המייצג את הדיסק כולו, כך שמספר הביטים בו (האורך שלו) כמספר יחידות ההקצאה (clusters) בדיסק; כלומר, הוא מכיל 1600 ביטים. בפועל הוא תופס 200 הבתים הראשונים של הסקטור מס' 1. יתר הסקטור לא בשימוש. לכן, טיפוס הנתונים שנשתמש ל-DAT (וגם ל-FAT, שנדבר עליו בהמשך) הוא כדלקמן:

```
typedef bitset<1600> DATtype;
```

הביט i ב-DAT מייצג את ה-Cluster i בדיסק. cluster פנוי מיוצג ע"י הערך 1 בביט המתאים לו ב-DAT, ו-cluster תפוס מיוצג ע"י הערך 0 (להסבר נוסף בקשר ל-DAT, ראה את החומר של הקורס התיאורטי).
לכן, לסקטור של DAT יהיה המבנה הבא:

שם שדה	טיפוס	משמעות	אורך השדה
sectorNr	unsigned int	מספר סידורי של הסקטור בדיסק	4 bytes
DAT	DATtype	DAT	200 bytes
emptyArea	char[820]	שמור לשימוש עתידי	816 bytes

הערה: שימו לב שה-DAT הוא מסוג bitset. ניתן ללמוד על טיפוס נתונים זה (א) מפרק 7 של הספר **"Thinking in C++, Volume 2"** מאת Bruce Eckel (תסתכלו ב**דוגמת קוד** מהספר הזה), ו-(ב) **מדף שבאתר של חברת מיקרוסופט**.

- כאמור לעיל, התיקייה הראשית של הדיסק תופסת שני סקטורים צמודים (cluster אחד), וכך גם כל תת-תיקייה. למעשה, כל תיקייה בדיסק הינה מערך כניסות שכל אחת מהן מיועדת לייצג קובץ בדיסק שיכול להיות קובץ נתונים או תיקיית-משנה (sub-directory). המבנה של כל כניסה כזאת (נקרא לה dirEntry) הוא כדלקמן:

שם שדה	טיפוס	משמעות	אורך השדה
Filename	char[12]	שם הקובץ	12 bytes
fileOwner	char[12]	שם בעל הקובץ	12 bytes
fileAddr	unsigned int	כתובת הסקטור הראשון של הקובץ	4 bytes
crDate	char[10]	תאריך יצירת הקובץ	10 bytes
fileSize	unsigned int	גודל הקובץ, כמספר סקטורים	4 bytes
eofRecNr	unsigned int	מיקום "רשומת" ה-end-of-file (המספר הסידורי של מיקומה מהתחלת הקובץ)	4 bytes
maxRecSize	unsigned int	אורך רשומה מרבי	4 bytes
actualRecSize	unsigned int	אורך רשומה בפועל	4 bytes
recFormat	char[2]	סוג: - אם מדובר בקובץ נתונים, זה מסמל סוג רשומה כלומר, אורך קבוע או משתנה : "F" או "V" - אם מדובר בתת-תיקיה, ערך השדה הזה יהיה האות "D".	2 bytes
keyOffset	unsigned int	Offset של התחלת המפתח בתוך הרשומה	4 bytes
keySize	unsigned int	אורך המפתח, כמספר בתים	4 bytes
keyType	char[2]	טיפוס נתונים של ערך המפתח: "I" - מספר שלם (int) "F" – מספר ממשי (float) "D" – מספר ממשי כפול (double) "C" - מחרוזת תווים	2 byte
entryStatus	unsigned char	שדה זה מעיד על מצב הכניסה הספציפית בתיקייה. המצב יכול להיות אחד מתוך שלושה: 0 - כניסה ריקה (empty): הכניסה עדיין לא הייתה בשימוש מאז שבוצע format על הדיסק . 1 - כניסה פעילה (active): הכניסה מייצגת קובץ קיים ופעיל. 2 - כניסה לא פעילה (inactive): הכניסה מייצגת קובץ מחוק.	1 byte

הערה: תיקייה תופסת שני סקטורים השייכים ל-cluster אבל היא מאוחסנת בשני חלקים שכל אחד מהם מכיל 14 כניסות של התיקייה. כלומר, לא ייתכן כניסה שחלק ממנה בסקטור אחד וההמשך שלה בסקטור הצמוד.

- כל יתר הסקטורים בדיסק מיועדים להכיל קבצי נתונים או תיקיות-משנה (sub-directories).
  - במקרה של תיקיית-משנה, גודל הקובץ יהיה תמיד cluster אחד בלבד, והמבנה שלו יהיה בדיוק כמו זה של התיקייה הראשית.
  - במקרה של קובץ נתונים, המבנה כדלהלן:

1. לסקטור הראשון של הקובץ, שהוא תווית הזיהוי שלו (File Header), יהיה המבנה הבא:

שם שדה	טיפוס	משמעות	אורך השדה
sectorNr	unsigned int	מספר סידורי של הסקטור בדיסק	4 bytes
fileDesc	dirEntry	העתק של כניסתו של הקובץ בתיקייה (file descriptor)	72 bytes
FAT	DATtype	FAT	200 bytes
emptyArea	char[744]	שמור לשימוש עתידי	744 bytes

2. ליתר הסקטורים בקובץ, המיועדים להכיל נתונים, יהיה המבנה הכללי של כל סקטור:

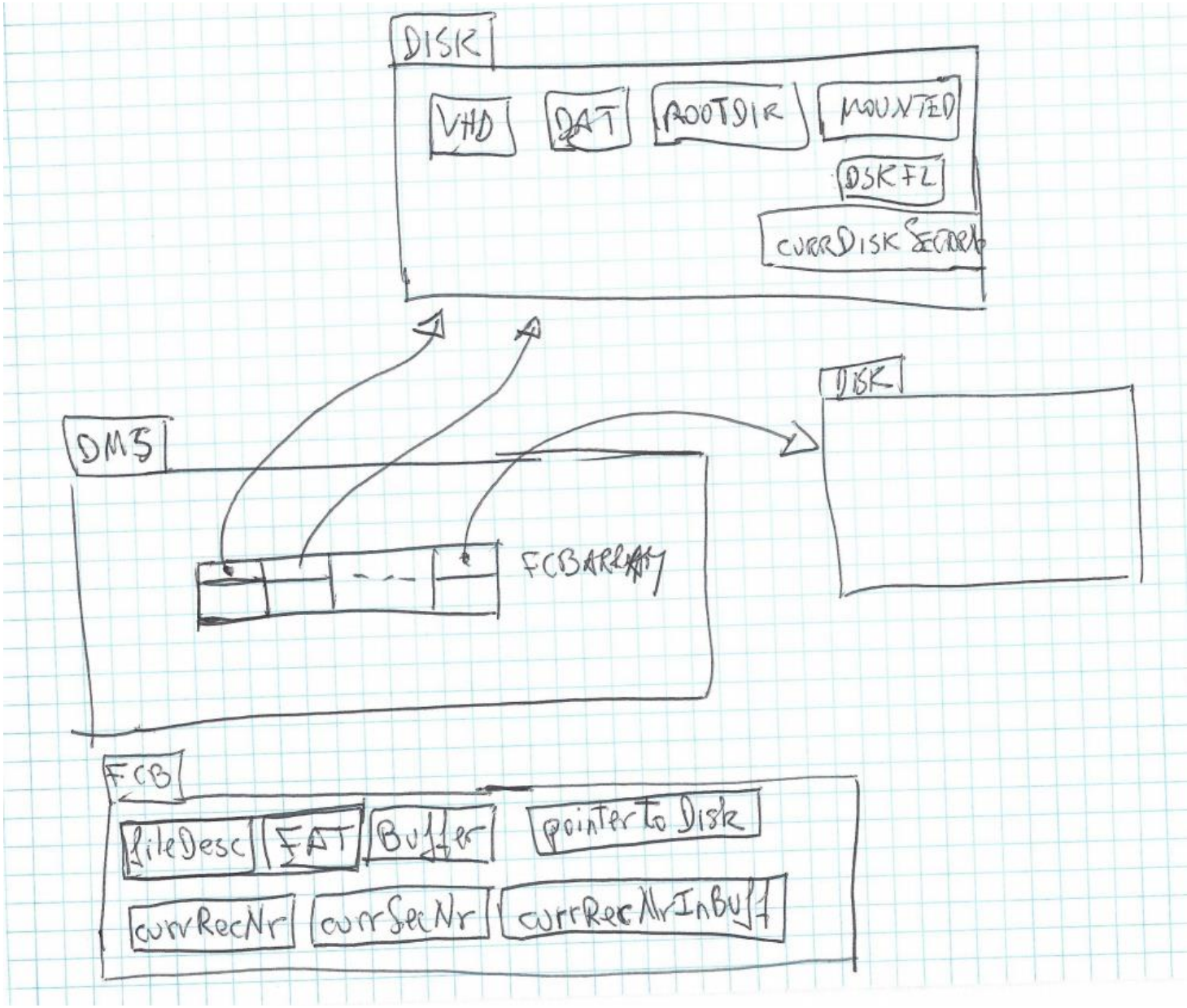
שם השדה	טיפוס	משמעות	אורך השדה
sectorNr	unsigned int	מספר סידורי של הסקטור בדיסק	4 bytes
rawData	char[1020]	נתונים	1020 bytes

מיקומם של הסקטורים האלה בדיסק ייקבע ע"י ה-FAT של הקובץ שהוא אובייקט מסוג bitset בדומה ל-DAT; ברור שלביטים ב-FAT יש משמעות שונה מזאת שב-DAT (ראה את החומר של הקורס התיאורטי).

<u>הערה: במיני-פרויקט הזה נממש דיסק שיש לו רק תיקייה ראשית, ללא תיקיות-משנה, למרות שמבחינה מבנית הכל מוכן לכך.</u>

בס"ד

<u>דיאגרמה המתארת את הקשרים המבניים בין המחלקות העקריות המרכיבות את המערכת לניהול הדיסק המדומה</u>


DISK
VHD
DAT
ROOTDIR
MOUNTED
DSKFL
currDiskSector
DISK
DMS
FCBARRAY
FCB
fileDesc
FAT
Buffer
pointerToDisk
currRecNr
currSecNr
currRecNrInBuff

בס"ד

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

הדו"ח הסופי:

- מבוא קצר (כחצי עמוד) על מטרות המיני-פרויקט
- ניתוח שלבים של המיני-פרויקט (לציין כל שלב ולפרט מה נעשה בו)
- בקשר למערכת לניהול דיסק מדומה, 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')
  - תיאור השיטות השונות להקצאת שטח בדיסק: ניתוח יתרונות וחסרונות של כל שיטה, וכו'.
  - תיאור מימושם של כל השלבים, במיוחד אלגוריתמים ומבנים שונים שהשתמשת; אם בשלב מסוים היו לך מספר מבנים ואלגוריתמים אלטרנטיביים, 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.
- <u>סיכום כללי</u> בו תכתוב מה להערכתך למדת במשך המיני-פרויקט, דעותיך בקשר לקורס, צוות ההוראה, הצעות לשיפור, הערות, הארות, ביקורות וכו'.
- <u>נספח</u> בו מעין מדריך למשתמש של המערכת בגרסתה עם GUI, עם הסברים על היבטים תכנוניים ותכנותיים.
- <u>נספח</u> שיכלול הצעה לשלב נוסף למיני-פרויקט על סמך השלבים הנוכחיים, ונימוק להצעתך.
- <u>אל תדפיס את הקוד</u> – כבר ראינו אותו בזמן הבדיקה.
- <u>אל תדפיס את הדו"ח</u> – תגיש אותו לתא ההגשה ב-moodle, שייפתח בשבילו.
- 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-פרויקט 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-פרויקט: הפונקציות קצרות ומשתמשות זאת בזאת
- טיפול בשגיאות (exception handling)
- מימוש פתרונות לבעיות תכנותיות בהן נתקלת במיני-פרוייקט
- להלן רשימה של סוגי השאלות שיכולות להישאל על המיני-פרויקט:

1. השוואה בין שיטות שונות לפתרון בעיה מסוימת במיני-פרוייקט לבין השיטה שבחרת; למשל, השיטות השונות להקצאת שטח בדיסק, או שיטת שונות למחיקת רשובה, וכו'.
2. מדוע הפונקציה flush היא פונקציה נפרדת? (האם היה צורך – או טעם - להגדיר פונקציה נפרדת לביצוע כתיבה פיזית של סקטורים?)
3. הצעת רעיון לשלב נוסף למיני-פרויקט על סמך השלבים הנוכחיים.
4. …