

פרויקט באירגון וניהול קבצים

כי לא באמת מבינים תאוריה עד למימוש

מגשים:
רועי מרגלית
נפתלי הרשלר

מבוא קצר

פרויקט בארגון וניהול קבצים בשפת C++ נועד על מנת לקבל תחושה ממשית של איך מתבצעים הדברים מתחת לפני השטח. כלומר, כיצד עובדת מערכת ניהול הקבצים על כונן במחשב, במילים אחרות, איך בפועל נשמר קובץ בדיסק קשיח. כמו גם, איך הוא נקרא (נפתח). הדבר נעשה על ידי מימוש מערכת קבצים פשוטה אשר נלמדה בקורס ארגון וניהול קבצים ויישומה בשפת C++.

דבר זה כלפי המשתמש נראה פשוט עד מאוד, לוחצים על צור קובץ ריק, ונוצר קובץ ריק. בפועל, מתחת לפני השטח יש לא מעט דברים שרצים, החל מחיפוש מקום פנוי לקובץ, הקצאתו וכלה בהגדרת מבנה הקובץ כך שיתאים לצורת העבודה עליו. וכל פרט מסתעף לעוד חלקים ועוד חלקים. הרבה חלקים קטנים היוצרים תמונה שלמה.

אף על פי שמדובר בפרויקט שאדם אחד יכול לנהל, הרי שיש כאן פן של ניהול ותכנון פרויקט, עבודה מסודרת בשלבים, הנחת יסודות ובניה על גביהם. דבר שנועד על מנת לתת תחושה לתהליך פיתוח תוכנה עם החלקים הכלולים בו.

על כן נראה שבתור מטרות משנה לפרויקט, אף על פי שהינן חשובות עד מאוד, ניתן למנות למידת מטודולגיות פיתוח, תכנון נכון, הגדרת סדר תכנות ויצירת קוד בנוי בשלבים המשתמשים בשלבים הקודמים להם.

ערך מוסף היה לפי בחירת הסטודנטים בהכרת סביבת פיתוח נוספת בשם QT או בכתיבת קוד ב C# המשתמש בספריות C++ (חיבור קוד Managed ו-Unmanaged).

ניתוח שלביהמיני-פרויקט

הפרויקט חולק למספר שלבים הנעים משלב 0 ועד לשלב 5 (בכל זאת, ב C++ מתחילים לספור מ 0).

שלב 0:

בשלב זה נוצרו רוב המבנים הבסיסיים אשר משמשים לייצוג הכונן ברמת הפיזית והלוגית. זאת כדוגמת הכונן, סקטור, מבנה תיקיה, מבנה קובץ ועוד. ניתן למצוא רשימה מלאה בצורה מתועדת בקובץ התייעוד. כמו כן, בשלב זה נוצרו פונקציות ליצירה, עיגון וסגירה של דיסק. בנוסף טיפול ברמת הסקטור הבודד.

חשוב לציין כי יש קשר הדוק בין המחלקה Sector למחלקות FileHeader ו VolumeHeader. מחלקות אלו אמורות להראות כמו סקטור לכל דבר ועניין. ככה שבעת שהקובץ נשמר בכונן, הרי הוא כסקטור, ובעזרת ידע מוקדם, אנו יודעים כי סקטור זה הוא באמת משהו אחר.

כך גם לגבי הצורה בה נשמרת התיקיה בכונן וע"כ יש לה צורה מיוחדת ליצוא ויבוא, עקב כי בזיכרון היא מוחזקת בצורה אחרת.

שלב 1:

בשלב זה מומשו פונקציות תשתית נוספות המנהלות את הדיסק שנוצר בשלב 0. שלב זה מאפשר אתחול הדיסק, הקצאת שטח בדיסק (למה שבעתיד יהווה מקום לקבצים) וכן שחרורו.

שלב 2:

שלב זה מטפל בקובץ ברמה הפיזית. כלומר, היכן וכיצד מאוחסן כסקטורים בדיסק. שלב זה כולל יצירת קובץ, הגדלתו ומחיקתו.

שלב 3:

בשלב זה נוצר ה File control block להלן FCB המטפל בקובץ ברמה הלוגית כהמשך טבעי לשלב 2. ה FCB מאפשר פתיחת קובץ על מנת לעבוד עליו בתור רצף של רשומות (מבנה נתונים פשוט) בפורמט הנבחר (כדוגמת סטודנט). פונקציות לדוגמא בשלב זה הינן, פתיחת קובץ קיים וסגירתו וכן טיפול ברשומותיו (ע"י קריאה, כתיבה, עדכון, ניווט ומחיקה).

שלב 4:

בשלב זה נוצר ה Disk Management System אשר נועד לפתוח מספר קבצים בדיסקים שונים (או באותו הדיסק) ושימוש בהם. ובין היתר משמש למניעת פתיחה כפולה של קובץ, דהיינו לוודא נעילת הקבצים בעת שימוש בהם.

שלב 5:

שלב זה נועד לבניית ממשק משתמש גרפי לתפעול, שימוש והצגה של השלבים הנ"ל. בשלב זה נלמדה צורת העבודה עם Qt וכן פרדיגמת Qt Model / View. נבחר בתור סביבה גרפית עקב העובדה שאז אין צורך לערבב שפות תכנות שונות ביחד. הסביבה הזאת כמו גם הפרויקט כתובים ב C++, כך שחיבור הדברים היה קל יותר לעומת חיבור קוד C# עם ספריית C++. החסרון היה בכך שנלמדה סביבה חדשה מאפס. הרצון היה לקבל כמה שיותר כלים מפרויקט זה.

הסברים והגדרות לכלל השלבים:**קבועים**

ראשית בחרנו להשתמש בקבועים עבור כל הגדלים המוגדרים בדיסק שאינם חלק ממבנה נתונים לשם קריאות התוכנית וכן על מנת להקל על עדכון עתידי (כלומר באם נרצה לשנות גודל הסקטור, כמות הסקטורים בקלאסטר או כמות הקלאסטרים בדיסק).

יתר על כן, השתדלנו בהגדרת פונקציות כ const היכן שניתן, זאת על מנת לאפשר זיהוי קל של פונקציות שאינן משנות את האובייקט לו הן שייכות. כמו גם למנוע שינוי של העצם בטעות כאשר אינו אמור להשתנות.

Typedef

בחרנו להשתמש ב Typedef על מנת להקל בזיהוי משתנים (האם unsigned int זה הינו מספר קלאסטר? מספר סקטור? מספר רשומה?). כמו כן, השתמשנו ב uint32_t במקום unsigned int על מנת להכריח גודל מוגדר. שהרי בשינוי הקומפילר וסביבת המטרה, יתכן כי יוגדר אחרת.

שיום

שינינו את שמות הפונקציות בפרויקט (באישור כמובן) לצורה קבועה ושיטתית זאת על מנת לאפשר זיהוי קל של מה הינו קבוע, מהי פונקציה ומה הוא משתנה.

צורת השיום אשר בחרנו הינה:

- PascalCase עבור מחלקות.
- camelCase עבור משתנים ופונקציות.
- UPPERCASE עבור קבועים.

מצביעים

השתדלנו להימנע ככל הניתן מ Raw Pointers על מנת להימנע ככל הניתן מדליפות זיכרון. זאת עשינו על ידי שימוש ב `std::unique_ptr`. כלי זה לוקח בעלות על ההקצאה הדינמית ודואג לשחררה ברגע שהוא נהרס ובכך מאפשר להחזיר הקצאות דינמיות מפונקציות מבלי לדאוג שהמקבל ישכח למחוק את ההקצאה.

Namespace

ראינו לנכון ליצור Namespace אשר קראנו לו Fms על מנת לא להשחית את מרחב השמות של מי שעתיד להשתמש בספריה ולמנוע התנגשויות (ע"כ גם נמנענו משימוש ב `using namespace` בקבצי ה header).

Namespace נוסף אשר יצרנו הינו FmsUtils. שכן מבחינה לוגית, הרשומה סטודנט לא שייכת לספריה. מי שעתיד להשתמש בספריה מחליט איך יהיה מבנה הרשומה שלו. מאידך היה צריך כי ניתן יהיה להשתמש בסטודנט המדובר גם מקוד ה Cli וגם מקוד ה Gui, בנוסף לדרישה כי כל הלוגיקה תהיה אך ורק בקוד הספריה ולא בממשקים. הגענו למסקנה שהדרך הכי נכונה לפעול במקרה זה הינה להוסיף את הסטודנטים בתור כלי עזר לבדיקת הספריה וע"כ הינו מהודר הימה ונמצא בפרויקט שלה אך בעל מרחב שמות אחר.

Pragma

הגדרות אלו הגדרנו על מנת להכריח את המהדר להתנהג בצורה מסויימת בעת ההידור.

- על מנת להכריח את הקומפיילר לא לעשות אופטימיזציה לטיפוסים, זאת בכדי לשמור על גודל מדויק המשמש להמרת הטיפוסים השונים ביניהם (סקטורים כללים למקרה הפרטי שלהם).
- על מנת להעיף אזהרות של ה - Visual Studio לגבי פונקציות שהינן לא בטוחות בשימוש שגוי.

אלגוריתמים ומימושים בשלבים השונים:**הסבר פעולות ההקצאה בהן השתמשנו בתוכנית:****האלגוריתם First fit**

אלגוריתם זה מקצה את השטח הראשון (מתחילת הדיסק) שגודלו גדול אושווה למספר הסקטורים המבוקש. יתרונו במהירות שכן אין צורך לסרוק את כל הדיסק ובעיקר בעת המצאות מקום בתחילתו. חסרונו, בכך שאופן פעולתו (רעיון ההקצאה בראשית) אינו אולטימטיבי בהקצאת שטח, שכן אינו מתיחס לכדאיות מקום זה על פני מקום אחר.

האלגוריתם Best fit

אלגוריתם זה מקצה את השטח המתאים ביותר למספר הסקטורים המבוקש. יתרונו בכך שהוא מאפשר לכל קובץ "לתפוס" בדיוק את המקום לו הוא זקוק. בנוסף, שומר על המקטעים הפנויים הגדולים עבור קבצים גדולים אשר באמת זקוקים להם. חסרונו, בכך שמבזבז מקום כאשר המקום הנמצא אינו בגודל המבוקש בדיוק לדוגמא, כאשר התבקש 400 והמקום המתאים ביותר הינו 420, הרי שיש "20" שמבזבזים אם לא שנקבל קובץ מספיק קטן המקום הלך לאיבוד.

האלגוריתם Worst fit

אלגוריתם זה מקצה את השטח המירבי (מתחילת הדיסק) הפנוי ללא קשר למספר הסקטורים המבוקש. יתרונו בכך שהסיכוי לשטח מבוזבז קטן יותר מאשר ב Best fit אך מכנגד חסרונו בכך שהוא מאבד את היכולת לקבל קבצים גדולים.

האלגוריתם Fill fit

אלגוריתם זה הינו אלגוריתם "גיבוי". כאשר יש מקום בדיסק לאחסן את הקובץ, אך מקום זה אינו רציף וע"כ האלגוריתמים האמורים לעייל יכשלו (עקב חיפושם מקום רציף), אלגוריתם זה יכנס לפעולה. האלגוריתם פועל על ידי סריקה של הדיסק והקצאת המקום המתאים ביותר עבור הקובץ, וריצה חוזרת. כלומר, כל עוד המקטע הפנוי הגדול ביותר שנמצא קטן מהגודל המבוקש, האלגוריתם יקצה מקטע זה לקובץ. ברגע שיש מקטעים יותר גדולים מהגודל המבוקש הנוותר, האלגוריתם יקצה את המקום המתאים ביותר כמו Best fit. יתרונו כי בהכרח יצליח באם יש מקום פנוי בדיסק. חסרונו בזמן הפעולה הגדול שלו, שהרי כנגד כל מקטע שימצא, יש לסרוק מחדש את כל הדיסק על מנת למצוא את המקום המתאים הבא עבור הקובץ (כי הוא מחפש את המקום האופטימלי עבור כל חלק וחלק). בנוסף, הקובץ יהיה מפוצל עד מאוד ויסבול מזמני ריצה פחות טובים.

הסבר פעולות לאחר ההקצאה, ובעיית ההקצאה המעגלית:

בעיה אשר נתקלנו בה בעת ניתוח שלבים 1 ו 2, הינה כי כאשר אנו מקצים שטח לקובץ מסוים ולאחר מכן מחליטים להרחיבו מספר פעמים, הרי שהוא עלול להגיע למצב, שההקצאה החדשה שלו הינה באמצעו או לפניו. בעוד שלפניו הינה בעיה קלה וניתן לפותרה ע"י שימוש במודולו הרי שמקום שהתווסף באמצעו הקובץ קוטע את רצפו.

בעיה זאת ניתן לפתור בכמה דרכים, האחת לא לאפשר מצב זה. כמובן שהחיסרון ניכר, שהרי לא תמיד אפשר יהיה להרחיב הקובץ. פתרון אחר, אשר הוצע ע"י המרצה, הינו לשמור טבלת הקצאות לכל קובץ וקובץ ולאחר מכן הקובץ יסרק לפני טבלה זו. פתרון זה לא התקבל על ידינו עקב מספר בעיות שראינו בו:

1. עדיף להימנע מהבעיה מלכתחילה מאשר ליצור אותה ואז לפתור אותה.
2. בכל התוכנית יש שימוש ב-unsigned int והמבנה שהוצע לפתרון משתמש ב-short כך שבכל השוואה עתידית שתהיה אנו נשווה טיפוסים שונים (לא מומלץ). דבר זה גם גורם לבעיה בגנריות התוכנית באם נרצה לשנות את גודל הדיסק לכזה ש short אינו יכול להכיל.
3. הגדלת הסיבוכיות ב"שכבות" העליונות של התוכנית, כל קריאה של קובץ או כתיבה תמיד נצטרך לעבור על מערך המיקומים ולנוע קדימה ואחורונה על גבי הדיסק בצורה לא עקבית כאשר סורקים סדרתית את הקובץ.
4. מפת הקובץ "FAT" מאבדת את משמעותה כאשר יש לנו מערך ששומר את המיקומים.

בכדי לפתור את הבעיה הנ"ל מומשה הפונקציה "Update FAT to File" אשר תפקידה לאחר כל "הרחבה" של קובץ מסוים ליישר את הקובץ. כך הבעיה אינה נוצרת מלכתחילה, שכן הקובץ תמיד יהיה בכיוון אחד. הפונקציה משווה את ההקצאה הישנה והחדשה של הקובץ ומזיזה את הנתונים הישנים למקום החדש שנוצר באמצע הקובץ. בצורה כזאת אם נשטח את ה FAT, (כלומר נמחק לוגית את השטחים שלא שייכים לקובץ), נשאר עם קובץ אחד רציף – כלומר סדר הקובץ מצד הנתונים הנו ישר. יתרונות הפתרון הנ"ל הינם כי הוא עקבי עם הטיפוסים המוגדרים וכן אינו חורג מההבנה המקורית בדבר סדר הקובץ.

קיים שיקלול תמורות בפתרון זה, שכן עלות הרחבת הקובץ גדלה, זאת בעקבות הצורך אשר עלול להיווצר בהזזת הקובץ בעת הרחבתו. מאידך, החיסכון נוצר בעת עבודה סדרתית על הקובץ, שכן כיוון ההתקדמות בכל שלב הינו רק קדימה ואין צורך לזגזג את ההתקדמות, כך נוצר ניצול טוב יותר של יכולת מחט הדיסק להתקדם בכיוון אחד בלבד. חישוב כדאיות זה נובע מכך כי יותר פעמים עוברים על הקובץ מאשר מרחיבים אותו, שכן אם לא שהנתונים לא נדרשים אף פעם ורק נכתבים, הרי שהם צריכים להקרא לפחות כמספר פעולות הכתיבה אם לא יותר. ע"כ העדפנו לייקר את הפעולה הפחות נפוצה ע"מ להוזיל את הפעולה הנפוצה יותר.

ספריה:

ראינו לנכון ליצור את הספריה בצורה כזאת שלא תהיה תלויה בכמות הסקטורים עליה היא נפרסת (גודל ספריה כגודל קלאסטר). ע"כ מימשנו אבסטרקציה מסוימת וכאשר עובדים עם תיקיה, הרי שהיא נראית כרצף אחד ארוך של כניסות תיקיה. ובעת אשר רוצים לשומרה כסקטורים, הרי שיש לה פונקציה ייצוא אשר מאפשרת שמירה בהתאם לדרישות על הכוון (כניסת תיקיה יכול להמצא בסקטור אחד בלבד ואסור עליה לגלוש לסקטור שלאחריה). וכן הכיוון השני אשר מאפשר לטעון הסקטורים ע"י בנאי מתאים למחלקה ספריה.

כמו כן, נוכחנו לדעת, כי לא פעם המטרה הינה לסרוק את כל התיקיה על מנת לקבל מידע על הפריטים השונים הנמצאים בה או על למנת לחפש פריט מסויים. ע"כ ראינו לנכון לממש Random Access Iterator אשר יאפשר לעבור על כל התיקיה בפשטות. כאשר משלבים זאת עם C++11 הרי שניתן בקלות לסרוק את התיקיה ע"י `for (auto entry : dir)` וכך לעבור על כל הפריטים במהירות.

Fat reader

על מנת לאפשר לשכבות העליונות יותר להתעלם מאי רציפות הקובץ בשכבות התחתונות, מומשה מחלקה בשם Fat Reader אשר יודעת לקבל Fat ולשטח אותו לרצף לוגי עבור השכבה העובדת עם הקובץ לוגית. כך לשכבה העליונה אין צורך בהבנת מבנה התנועה של הקובץ. כל שהיא עושה הוא לבקש מה Fat Reader את מיקמו הפיזי של קלאסטר לוגי מסויים, ה Fat Reader עובר על ה Fat ומיידיע את השכבה מהו המקום אותו היא מחפשת. כך אם נרצה לשנות את מבנה השמירה בדיסק למערכת קבצים מסוג אחר, הרי שנוכל להחליף את ה FAT בכל שיטה אחרת ולהחליף את ה Fat Reader בכלי אחר מתאים עם אותם פונקציות ציבוריות, השכבות העליונות לא תדענה מכך וגם לא יהיה להן איכפת.

Defrag

ניסינו לממש פונקציה איחוי בסיסית על מנת להבין את גודל הקושי. פונקציה זאת מנסה ליצור מצב שכל קובץ בכוון הקשיח הינו רציף וע"כ זמן העבודה הרציף שלו הרבה יותר מהיר. הקושי הגדול ביותר בו נתקלנו הינו, איך מעבירים קובץ ממקום למקום מבלי לפגוע בתוכנו גם כאשר מזיזים אותו על עצמו, כלומר להזיז את כל הקובץ קלאסטר אחד לכיוון כלשהו. בעיה זה פתרנו בסופו של דבר, על ידי מיפוי כל העברות הדרושות על מנת להזיז את הקובץ. כל הזזה שכזאת שיעד הכתיבה אינו שייך לקובץ בשום צורה שהיא (מקום חדש למשל) הינה העברה חינום ואין עימה בעיה. לאחר מכן, מתחילים במשחק של החלפת נתונים ממקום למקום, ובאם הנתון במקום היעד מיועד לכתיבה במקום אחר, הרי שאנו מציינים לכתיבה זאת כי מקורה הינה מהמקום החדש. כעין כיסאות מוזקלים כאשר המטרה להגיע לכך שכל אחד ישב במקום הראוי לו.

את האיחוי עצמו ניסינו לממש ע"י כך שבכל פעם ננסה לדחוף קובץ בצורה לא אפקטיבית לסוף הכוון, כך יפנה לנו שטח עבודה בתחילתו. כעת ננסה ליישר את כל שאר הקבצים לתחילת הכוון בעזרת המקום שיתפנה. בעזרת תהליך זה אנו גורמים לכך שהמקום הפנוי שהיה בין הקבצים השונים יעבור לסופם, בסיום תהליך זה אנו מיישרים את הקובץ שהוזז חזרה להתחלה (או לפחות מנסים אם ביכולתנו הדבר).

תהליך זה נעשה על כל אחד מהקבצים על מנת להזיז את הקבצים שמפריעים לרצף, יצירת רצף והחזרת הקובץ למקום רציף באם ניתן.

במקרים שבהם יש מספיק מקום משחק, התוצאה היא כי הקבצים הופכים רציפים, באם אין מספיק מקום משחק להזזת הקבצים, התוצאה היא כי הדיסק נשאר במצב לא מאוחה עקב חוסר היכולת להזיז הקבצים בלי להרוס אותם.

סיכום כללי

בקורס זה למדנו להשתמש ב Qt כמו גם את שיטת Qt .Model / View הינה Framework שלם ומקיף ב ++C. אף על פי שנגענו רק במקצתו, הרי שחווינו את תחושת השימוש בו. חלקו מזכיר את std שכן יש מימושים אחרים לחלק מהדברים שם (למשל string). הרגשנו בתופעה הלא נעימה של עירבוב Frameworks שכן התוצאה גוררת המרות חוזרות ונישנות של משתנים על מנת שיוכלו לעבוד אחד עם השני.

שיטת מודל - תצוגה, לעומת זאת מזכירה את מודל השכבות אליו נחשפנו בקורס מיני פרויקט בחלונות. הרעיון הינו חלוקת תפקידים ברורה, כך שכל חלק אחראי על תפקידו שלו כאשר יש צורת תקשורת ברורה בין החלקים. הייתרון בכך הינו, שכאשר היינו זקוקים למידע כלשהו, יכלנו לכתוב מודל אחד שינהל את המידע כנגד מקור הנתונים בעוד שכמה 'תצוגות' שונות ניגשו לאותו מודל והציגו את הנתונים השונים. כלי זה חסך כתיבה של קוד, גם אם לוקח יותר זמן לבנות אותו.

אף על פי שהפרויקט עבר כבר כמה שנים במכון, מ"מ ראינו להציע מספר הצעות לשיפורו להבא.

- שינוי שמות הפונקציות והמחלקות במפרט על מנת להיות עקביות. בדומה למה שעשינו אנו בפרויקט.
- חישוב מחודש של הגדלים, שכן כעת יש כמה אי התאמות בגדלים השונים בפרויקט.
- החלפת כל המקומות אשר כתוב בהן unsigned int ל uint32_t ודומיו. זאת בעקבות העבודה שהגודל של המשתנים חשוב מאוד לחישוב מבנה המבנים השונים. במפרט השפה ל unsigned int מוגדר גודל מינימלי, אך לא מוגדר גודל מרבי. דבר זה יכול לגרום לבעיות תאימות בעת המרת המחלקות השונות.
- הוספת מידע מסודר על Qt לתלמידים המעוניינים לפתח בסביבה זו.
- שינוי שם הפונקציה delete במחלקה Fcb. שם פונקציה זאת הינה מילה שמורה ב ++C וע"כ לא ניתן להשתמש בה.
- שיכתוב הפרויקט מנקודת מבט של ++C. כרגע הפרויקט נראה כפרויקט C שעבר המרה ל ++C, ואינו מנצל או משתמש בגישות השפה.

תוספת לפרויקט

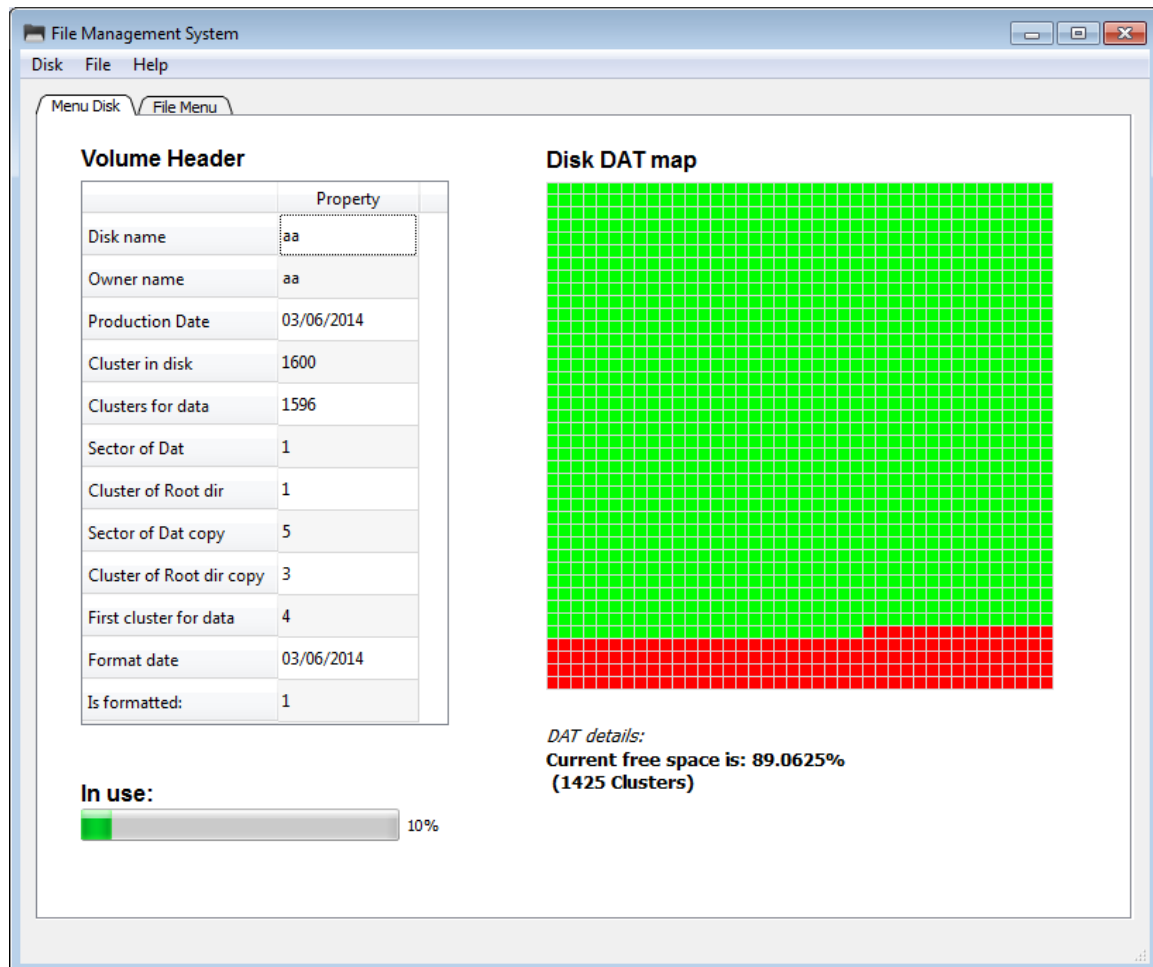
שלב נוסף אשר ניתן להוסיף לפרויקט, בהינתן זמן מתאים, הינה האפשרות לעבוד עם תתי תיקיות. רוב הקוד לכך קיים, אם כי יש קטעים במפרט אשר לא תוכננו למספר תיקיות כדוגמת openFile אשר מחפש קובץ רק בתיקיה הראשית. בצורה כזאת ניתן יהיה להרגיש קצת יותר את המבנה ההיררכי של מערכת הקבצים ועבודה ברמה של תיקיות ולא רק עבודה פרטנית על קובץ בודד.

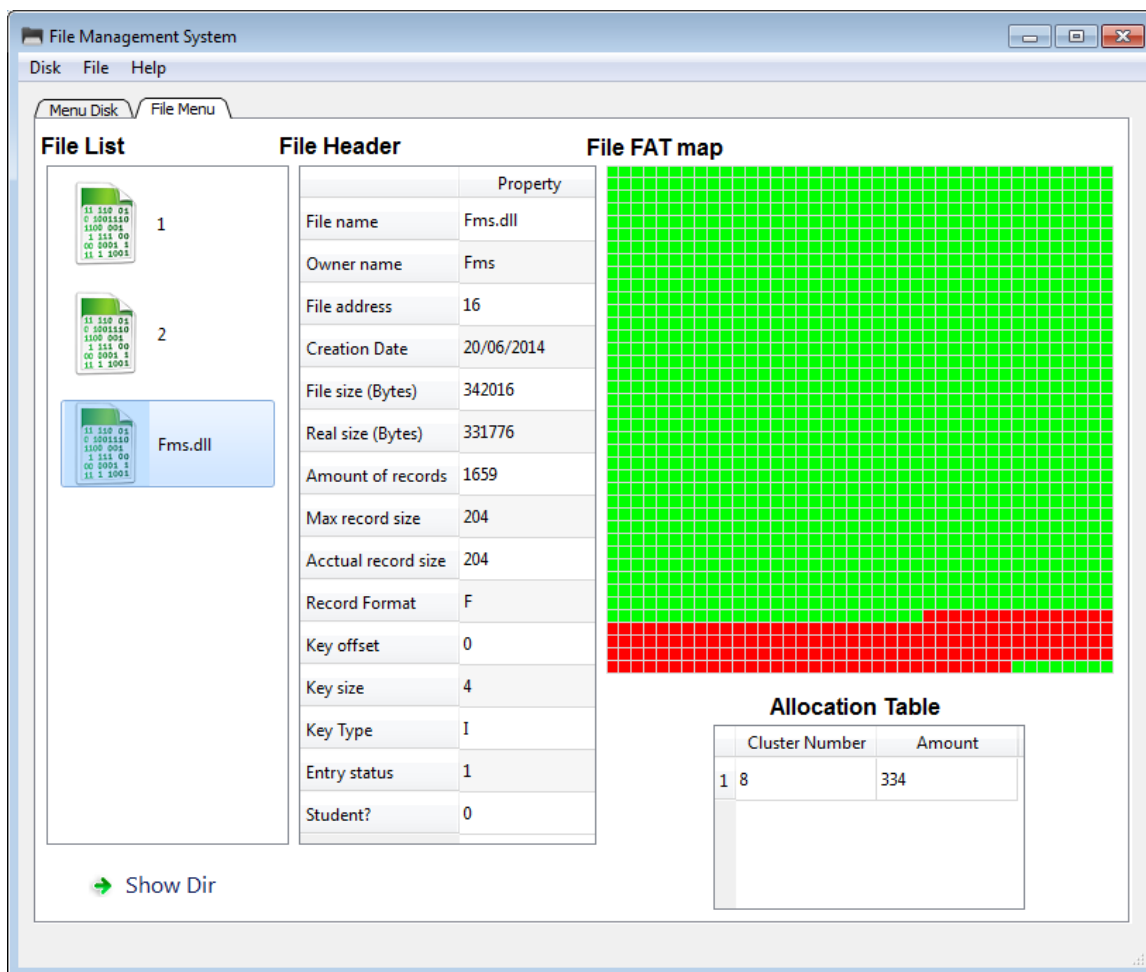
מדריך לממשק משתמש:

הממשק שנבנה מתחלק לשני אזורים עיקריים. האחד, התצוגה בחלון שמפרטת את כל הנתונים הרלוונטים על הדיסק וכן אינפורמציה על כל הקבצים. השני, התפריט בסרגל העליון של החלון המאפשר לבצע פעולות שונות השייכות למצב הנתון.

חלונות התצוגה

תצוגת התכנית מחולקת עם שני "טאבים" (חוצצים). האחד, לאינפורמציה על הדיסק הכולל בתוכה את ה-Volume Header (נתוני הדיסק) וכן את מפת הקיבולת (DAT) ופירוט נוסף עם "בר התקדמות גרפי" על מס' המקומות התפוסים וכן במילים ובאחוזים. השני, לאינפורמציה על הקבצים, החלון מכיל את התיקה הראשית עם הקבצים, אופן הצגתם מופיע בצורה גרפית עם אייקונים ואפשרות "דפדוף" עם העכבר והמקלדת. כמו כן, תצוגה המכילה את ה-File Header (נתוני הקובץ) וכן את מפת קיבולתו (FAT). בנוסף, חלון עם מיקומי ההקצאות של כל קובץ נתון. המידע בחלונות הנ"ל משתנה בהתאם לקובץ הנבחר מחלון התיקה הראשית.





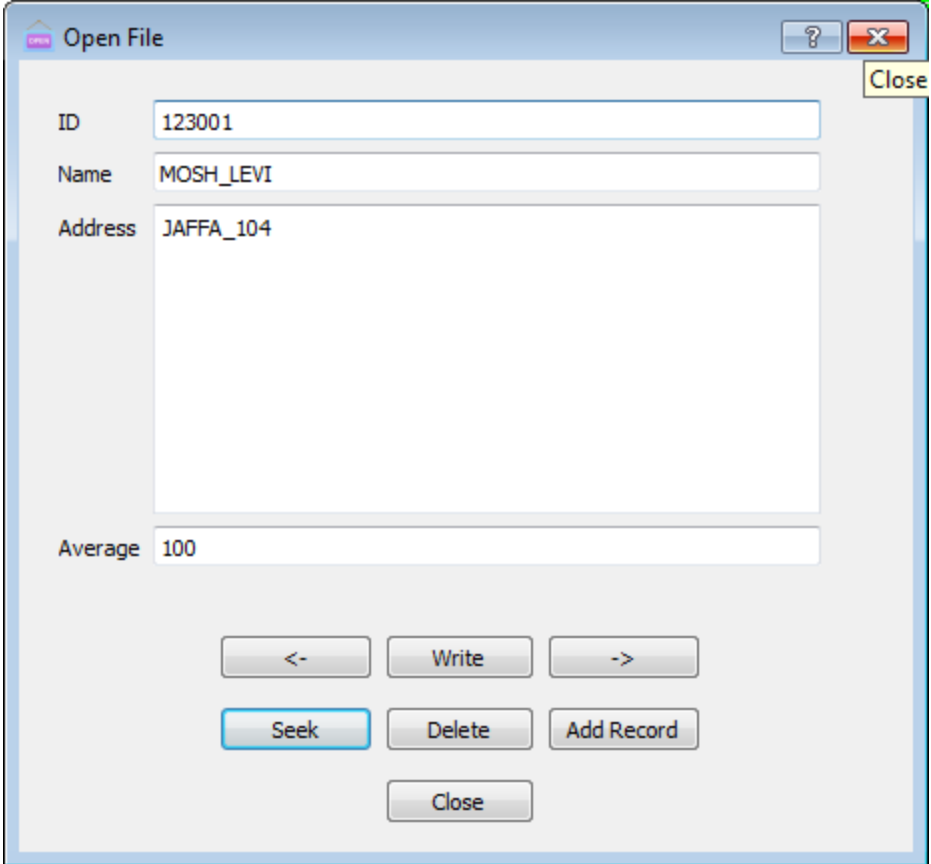
ניתן לבחור לבצע פעולה על קובץ קיים גם מהתפריט File כמו גם ע"י לחיצה של לחצן ימיני על הקובץ.

מצב רשומות

קובץ סטודנטים ניתן לפתוח כקובץ סטודנטים ולערוך אותו בהתאם. האפשרויות הקיימות הן מעבר קדימה ואחורה על פני הרשומות, שמירת שינויים ברשומה, מחיקת רשומה וקפיצה לסוף הקובץ ע"מ להוסיף רשומות. כמו כן יש אפשרות תנועה לפי רשומות בתוך הקובץ (3 רשומות קדימה מהמיקום הנוכחי).

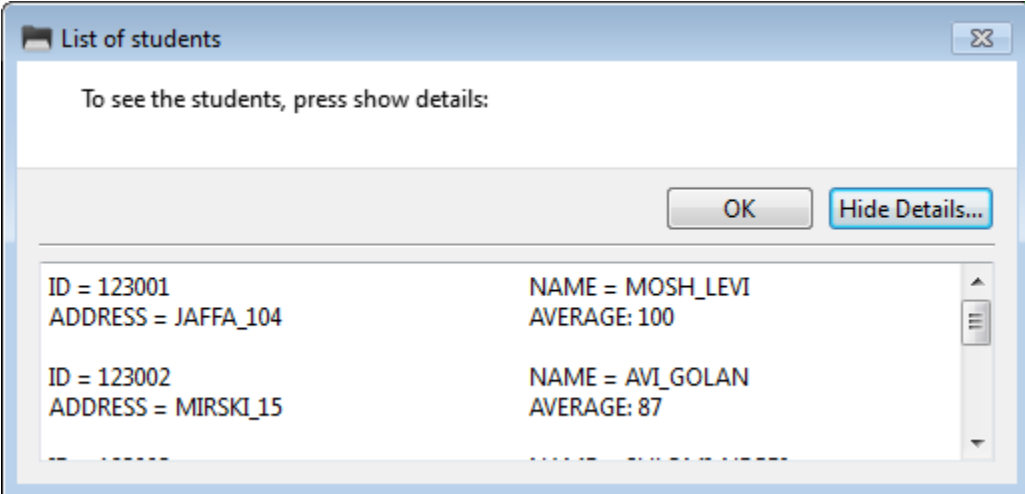
בחרנו להציג רק רשומה אחת בכל פעם על מנת להמחיש את צורת העובדה עם הקובץ, כאשר כל פעם נטען חוצץ לזיכרון ועובדים רק על רשומה אחת בזמן נתון, שכן כביכול אין לנו מספיק זיכרון להחזיק את כל הרשומות בזיכרון בבת אחת.

מ"מ איפשרנו להדפיס את כל הרשומות ע"י לחיצה על הכפתור Print All. דבר זה נועד ע"מ לעמוד בדרישות המרצה שאפשר יהיה לראות את כל הרשומות בבת אחת. אפשרות זאת נמצאת מהתפריט הראשי ולא במצב רשומות שכן היא תגרור פתיחה נוספת של הקובץ.



The 'Open File' dialog box contains the following fields and buttons:

- ID:** 123001
- Name:** MOSH_LEVI
- Address:** JAFFA_104
- Average:** 100
- Buttons:** <-, Write, ->, Seek, Delete, Add Record, Close.



The 'List of students' dialog box displays a list of student records. The text 'To see the students, press show details:' is at the top. Below the list are 'OK' and 'Hide Details...' buttons. The list contains the following data:

ID = 123001 ADDRESS = JAFFA_104	NAME = MOSH_LEVI AVERAGE: 100
ID = 123002 ADDRESS = MIRSKI_15	NAME = AVI_GOLAN AVERAGE: 87

כמו כן, הוספנו מצב להדפסת ה Root Directory, הוא בצורה נפרדת שכזאת כי מצד יופי הממשק לא ידענו איך להראות אותו, היה נראה לנו שהעיקר החשוב בו הוא רשימת הקבצים ולא יותר מזה. ע"כ זה כל מה שמופיע במסך הראשי. גם לפי מפרט הפרויקט היא לא בהכרח מעודכנת כראוי וממילא המידע שמופיע שם פחות שימושי (על כך הוצרך בפרויקט לקרוא את ה FileHeader שבדיסק כל פעם).

