בית הספר הגבוה לטכנולוגיה בירושלים

מיני-פרויקט באירגון וניהול קבצים – התשע"ב

מורים:
ד"ר משה גולדשטיין, מר עזרא דשט, מר אריה ויזן, מר אושרי כהן, מר אליעזר גינסבורגר
גב' חני נדלר, גב' עדינה מילסטון, אורית רוזנבליט

הקדמה

ככל מיני-פרויקט, עיקר הקורס הזה יהיה מעשי-תכנותי. משימתך במשך הסמסטר תהיה בניית מערכת תכנה שמיישמת מערכת לניהול מסמכים. המטרה העיקרית של המערכת תהיה אחזור מידע ממאגר מסמכי טקסט. למטרה זו נשתמש במבנה נתונים דומה למה שבספרות נקרא **trie** (מהמילה re**trie**val). כל התכנות, כולל הממשק הגראפי (GUI), חייב להיכתב בשפת ++C. לא תתקבלנה עבודות כתובות בשפה כלשהי אחרת. סביבת הפיתוח היחידה שבה כל העבודה תתבצע חייבת להיות Visual Studio 2010.
הרעיון הוא לכתוב חבילה של מחלקות (classes – בשפת ++C) שמיישמות את מערכת ניהול המסמכים הנ"ל. כלומר, באמצעות חבילה זו יהיה אפשר לבצע פעולות על מאגר מסמכי (קבצי) טקסט: הוספת מסמך למאגר, מחיקת מסמך מהמאגר, עדכון מסמך, ובמיוחד אחזור מידע ממאגר המסמכים באמצעות שאילתות טקסטואליות דומות לאלו שאנחנו רגילים לבצע עם מנועי חיפוש כגון google.

המוצר הסופי יהיה ישום בעל ממשק גראפי שבאמצעותו המשתמש יוכל להוסיף, למחוק וגם לעדכן מסמך, ולהציג שאילתות חיפוש. לצורך זה, נבדיל בין סוגים שונים של משתמשים: משתמשים רגילים שיורשו לחפש בלבד, ומשתמשים מיוחדים (דומה למעין super user) שיוכלו לבצע את כל הפעולות.
למימוש כל הנ"ל, הקורס הזה מניח שלך יש ידע קודם בתחומים הבאים:

1. ++Cמחלקות המאפשרות טיפול בקבצים (קלט/פלט) בשפת . ראה במצגות מהספר מאת Deitel&Deitel: **chp12**, **chp14**, **chp19**, **chp22** ; ראה גם במצגות מהספר מאת Bruce Eckell: **chp2**, **chp3**-1.
2. GUIתכנות ממשקים גראפיים ().

למיני-פרויקט יהיו מספר שלבים התלויים זה בזה: כלומר, על מנת לבצע שלב מסוים יש צורך בהשלמת השלב הקודם לו. לפיכך לא נרשה להצטרף לקורס אחרי הגשת שלב מס' 1.

הערות חשובות

(א) חובה להסתכל מדי פעם באתר הקורס במודל.
(ב) לגרסה הסופית של המיני-פרויקט נדרוש יישום בעל ממשק גראפי, אבל לאורך רוב שלבי פיתוח המערכת, נסתפק בממשק טקסטואלי. _

נוהלי עבודה

העבודה תעשה בזוגות; נוכחותם של שני בני הזוג בזמן השיעור במעבדה ***חובה***. אם תהינה בעיות כלשהן במשך הסמסטר, ניתן יהיה לפנות למורי הקורס באמצעות דואר אלקטרוני, בהתאם לקבוצה שכל אחד מהם אחראי לה.

נוהלי הגשה ובדיקה

בתום ביצוע כל אחד מהשלבים במיני-פרויקט, כל זוג ידגים את הרצת אותו שלב למורה האחראי לקבוצה שלו, בזמן השיעור במעבדה; הציון על ההרצה יינתן על המקום ע"י המורה. בנוסף לזה, כל זוג יהיה חייב להגיש למודל קובץ ZIP שיכיל את עבודתו; ביחד עם הגשת כל שלב, תידרש הגשת דו"ח בעבור השלב הנוכחי. ללא ההגשה במועד במודל, לציון ההרצה של כל שלב לא יהיה תוקף.
מומלץ מאוד שכל זוג ישמור כל דבר רלוונטי לכל אחד מהשלבים במיני-פרויקט: שיבנה תיקיה במחשב שלו שהולכת וגדלה עד סוף הסמסטר. כדאי לשמור את כל זה כי ישמש חומר גלם להכנת הדו"ח המסכם ולהגנת המיני-פרויקט.
שבועיים לפני סוף הסמסטר ייקבע מועד מועד להגנה על המיני-פרויקט כולו, מול המורים המתאימים לקבוצות השונות.

ציון

הציון הסופי ייקבע לפי הטבלה הבאה:

משימה	משקל	הערות
ה-main כיישום Console (טקסטואלי)	0%	

ה-main כיישום חלונאי בעל GUI	25%	
כל שלבי המיני-פרויקט	60%	הציון לכל שלב ייקבע לפי הקריטריונים הבאים: ▪ <u>תכנות</u>: 40% – שימוש בשיטות תכנות "נכונות", ובמבני נתונים מתאימים לבעיה. מה הכוונה של תכנות "נכונה"? ▪ הכתיבה חייבת להיות מודולרית: ▪ כל פונקציה חייבת להיות באורך סביר (לא יותר מעשרות בודדות של שורות לכל היותר), מה שיאפשר לקרוא ולהבין אותה בקלות, עד כדי כך שלא יהיה צורך בתיעוד בתוך גוף הפונקציה עצמו. ▪ שמות משמעותיים יוענקו למשתנים, לפונקציות ולקבועים, כך שמשמעותם ותפקידם יהיו מובנים מאליהם, מה שיתרום לקריאות התכנית. ▪ שמות הקבועים ייכתבו באותיות גדולות, לעומת שמות המשתנים והפונקציות שייכתבו באותיות קטנות. ▪ indentationלעשות *אינדנטציה* () של גוף לולאה ושל גוף if-else. ▪ לא מומלץ להשתמש בכל מיני "טריקים" של השפה על מנת לחסוך בכתיבת שורת קוד או במשתנה אחד. ▪ <u>תיעוד</u>: 20% – לתעד באופן ברור כל פרט חשוב בתכנית. למשל: ▪ classלכל מחלקה (), לתאר את מטרתה, ואיפה אפשר להשתמש בה. ▪ לכל פונקציה, לכתוב מפרט (ספסיפקציה) מייד לפניה; כלומר, לכתוב תיאור כללי של הפונקציה, מהם הפרמטרים (הקלטים) ומשמעותם, סוג הערך המוחזר 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, לכתוב תיעוד קצר ליד הגדרתו. ▪ <u>הרצה</u>: 40% – התכנית חייבת לבצע את משימותיה בצורה נכונה, בהתאם לדרישות ולמפרטים.
דו"ח מסכם	15%	הדו"ח המסכם של כל המיני-פרויקט על כל שלביו, יכלול הסברים רלוונטיים לכל שלב ושלב, וכל דבר שראוי לציין על מה שנעשה במשך הקורס. אם בשלב מסוים של המיני-פרויקט חשבתם על מבני נתונים אלטרנטיביים לפתרון הבעיה הנידונה באותו שלב, הדו"ח המסכם הוא המקום להביא אותם ולנתח יתרונות וחסרונות של כל אחד מהם (מבחינת הערכת זמן ריצה ומקום), ולהסביר למה נבחר מבנה הנתונים שמומש בפועל בתכנית.
הגנה		ההגנה על המיני פרויקט תשמש לקביעת הציון הסופי של הקורס לפי הנוסחה הבאה: (ציון ההגנה) * (ציון משוקלל של כל המיני פרויקט), כאשר (ציון ההגנה) יהיה ערך בין 0.0 ל-1.0 (ציון משוקלל של כל המיני פרויקט) יהיה ציון בין 0 ל-100. לדוגמה: אם הציון המשוקלל של כל השלבים והדו"ח המסכם אמור להיות 90, ובהגנה נותנים לך 1.0, הציון הסופי יהיה גם 90; לעומת זאת, אם בהגנה נותנים לך 0.95, הציון הסופי יהיה 85.5 במקום 90.

תכנית הסמסטר

בקשר למטלות הקורס לאורך הסמסטר

שבוע	פעילות
1	הרצאה: *משימתך* במיני-פרוקט הזה, ליישם מערכת לניהול וחיפוש מסמכים. הבסיס למערכת הזאת חייב להיות מיושם כ-DLL המכיל מחלקה ב-++C בשם triedocs; ה-DLL ייקרא triedocs.dll על מנת ליישם את triedocs, נדון בשיעור הזה בנושאים הבאים: (א) trieעץ רב-בנים ושמו (מהמילה retrieval). חלק זה של השיעור יהיה מבוסס על המצגת "**Search Engines and Tries**", המתארת את העקרונות של מבנה הנתונים trie והשימוש בו לצורך מנועי חיפוש של מאגרי מסמכים. (ב) מבנה מאגר מסמכים מהסוג שניישם במיני-פרוקט: תיאור של מבנה של מאגר מסוג זה ניתן למצוא כאן. על בסיס מבנה זה, לך יש חופש להוסיף, לשנות וכו', אפילו להציע מבנה משלך (הסברים עליו ביחד עם הנימוקים המתאימים, יצטרכו להופיע בדו"ח המסכם של המיני-פרוקט). מה שחייב להיות אחיד לכולם הוא שם המחלקה ושמות הפונקציות שיופיעו במטלות של שלבי המיני-פרוקט. למרות שבמטלות המיני-פרוקט יופיע תיאור מפורט של הפרמטרים הדרושים לכל פונקציה, כל אחד/אחת מכם חופשי/חופשיה להוסיף פרמטרים, לפי שיקול דעתו/דעתה האישית; במקרה כזה, כל פרמטר נוסף יצטרך להיות פרמטר אופציונאלי עם ערך ברירת מחדל מוגדר מראש על ידיכם. התנאי לחופש הבחירה הזה הוא שכל מה שנידרש בשלבים השונים של המיני-פרוקט יעבוד בדיוק לפי הדרישות. מובן מאליו שאם תוך כדי פיתוח המיני-פרוקט יש צורך להגדרת מבני נתונים כלשהם נוספים, זה יהיה עניין של החלטות תכנוניות/תכנותיות שלכם. הערה חשובה: שים לב שבמיני-פרוקט הזה משתמשים במחרוזות בשני מובנים (וייצוגים) שונים. מחרוזות מאוחסנות ברשומות של קבצי trie כמחרוזות בסגנון שפת C (מערך של תווים שמסתיים ב-'\0'). לעומת זאת, רוב המחרוזות שמתקבלות כפרמטרים בפונקציות השונות של המחלקות שנממש במיני-פרוקט, הינן אובייקטים מסוג המחלקה string של ++C. כלומר, ברוב המקרים, ברמת האפליקציה, המחרוזות אמורות להיות מהסוג של המחלקה string של ++C, וברמת המערכת לניהול קבצי trie 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 דוגמה שנמצאת כאן.
2,3	**שלב מס' 1: יצירה, מחיקה, פתיחה וסגירה של מאגר מסמכים – הגשה: 22/3/2012 – כ"ח באדר תשע"ב** כוונתנו העיקרית בשלב זה של המיני-פרוקט, להגדיר ולממש: שתי פונקציות בונות של המחלקה triedocs, פונקציה הורסת אחת, פונקציה (create) ליצירת מאגר חדש, פונקציה (delete) למחיקת מאגר קיים, פונקציה (mount) לפתיחת מאגר קיים, פונקציה (unmount) לסגירת מאגר פתוח. במקרה של הפונקציה delete, המחיקה תוכל להיות משני סוגים: (א) מחיקה לוגית – מחיקת מנגנון ניהול המסמכים של המאגר, בלבד (ב) מחיקה פיזית – מחיקת המאגר כולו, כולל המסמכים עצמם. הגדרה מפורטת של משימתך לשלב זה של המיני-פרוקט אפשר למצוא כאן.
4,5,6	**שלב מס' 2 : העלאת והורדת מסמכים, הוספת, ומחיקת מסמכים – הגשה: 19/4/2012 – כ"ז ניסן תשע"ב** כוונתנו העיקרית בשלב זה של המיני-פרוקט, להגדיר ולממש: ארבע פונקציות עיקריות: פונקציה (docupload) שבאמצעותה יהיה אפשר להעלות מסמך למאגר בלי עדיין להוסיף אותו למנגנון האינדוקס של המערכת, פונקציה (docdownload) שבאמצעותה יהיה אפשר להוריד מסמך ספציפי מהמאגר, פונקציה (docinsert)שבאמצעותה יהיה אפשר להוסיף מסמך למאגר, ופונקציה (docdelete) שבאמצעותה יהיה אפשר למחוק מסמך מהמאגר (מחיקה זו תהיה מחיקה לוגית – פיזית המסמך ימשיך להתקיים אבל לא יהיה נגיש). הגדרה מפורטת של משימתך לשלב זה של המיני-פרוקט אפשר למצוא כאן
7,8	**שלב מס' 3: חיפוש במאגר מסמכים – הגשה: 10/5/2012 – י"ח אייר תשע"ב** כוונתנו העיקרית בשלב זה של המיני-פרוקט, להגדיר ולממש: שתי פונקציות חיפוש עיקריות: פונקציה (docexists) בולאנית שבאמצעותה יהיה אפשר לדעת האם קיימים מסמכים שעונים לשאילתא מסוימת, פונקציה (doclookup) שבאמצעותה יהיה אפשר לקבל רשימת מסמכים שעונים לשאילתא מסוימת. הגדרה מפורטת של משימתך לשלב זה של המיני-פרוקט אפשר למצוא כאן
9,10	**שלב מס' 4: עדכון מאגר המסמכים – הגשה: 25/5/2012 – ד' סיוון תשע"ב** כוונתנו העיקרית בשלב זה של המיני-פרוקט, להגדיר ולממש: שלוש פונקציות עיקריות: פונקציה (idxupdate) שבאמצעותה יהיה אפשר לעדכן אינדוקס של מסמכים קיימים במאגר, פונקציה (docupdate) שבאמצעותה יהיה אפשר לעדכן מסמך קיים שמחברו החליט לשנותו (במקרה כזה, המסמך הקיים ימחק מהמאגר, ואז המסמך המעודכן יוכנס כמסמך חדש), פונקציה (reorg) שבאמצעותה יהיה אפשר לארגן מחדש את כל המאגר (במקרה הזה, כל המסמכים שנמחקו לוגית, יימחקו פיזית). הגדרה מפורטת של משימתך לשלב זה של המיני-פרוקט אפשר למצוא כאן.
11,12	**שלב מס' 5: מימוש הממשק הגראפי – הגשה: 14/6/2012 – כ"ד סיוון תשע"ב**

	כוונתנו העיקרית בשלב זה של המיני-פרויקט, להגדיר ולממש את הממשק הגראפי למערכת לניהול וחיפוש מסמכים שבניתם במשך ארבעת השלבים הקודמים. אתם חופשיים לתכנן את הממשק בצורה שתמצאו לנכון. חשוב מאוד שהשימושיות שלו תהיה גורם מרכזי בתכנונו. הצעה לממשק כזה אפשר למצוא כאן.
13	**הגשת דו"ח המיני-פרויקט: עד סוף הסמסטר** **הגנה על המיני-פרויקט כולו: במפגש האחרון של הסמסטר**

הערה כללית חשובה

בכל שלבי המיני-פרויקט, אם תתעורר בעיה/תקלה כלשהי בביצועה מתאימה- באמצעות מנגנון ה (exception) של פונקציה כלשהי במערכת שלך, הפונקציה הנידונה תיצור תקלה try-throw-catch, שתטופל על ידי גורמים מתאימים. רק באמצעות המנגנון הזה, הגורם המבקש לבצע פעולה מפעולות המערכת יידע על תקלה כלשהי שהתרחשה.

תאור מפורט של מבנה מאגר המסמכים

מבנה כללי

מאגר מסמכים מנוהל בתוך תיקיה (ראה איור מס' 1), ושם המאגר כשם התיקייה; תיקיה זו הינה *התיקייה הראשית* של המאגר. מבחינת מערכת ניהול המסמכים, כל מסמך במאגר מיוצג כשלישיה של קבצים שמאוחסנת בתוך תת-תיקיה של התיקייה הראשית. שמה של כל תת-תיקיה כזו יהיה כשם המסמך המאוחסן בה. ברור שמספר המסמכים המאוחסנים במאגר כמספר תת-התיקיות של התיקייה הראשית. בנוסף, בתיקיה הראשית יכול להיות קובץ מסוג stop ושמו stop.lst (ראה בהמשך הסבר מפורט).

איור מס' 1: מבנה של מאגר מסמכים

שלישיית הקבצים המייצגת מסמך, כדלהלן:

- קובץ טקסט גולמי (המסמך):

קובץ בשם ***docname.xxx***: ***docname*** הוא שם כלשהו של מסמך כלשהו, ו-**xxx** סיומת כלשהי, פרט לסיומות השמורות לקבצי המערכת (**stop, lst, trie**) ולסיומות שבאופן סטנדרטי אינם קבצי טקסט במערכת ההפעלה Windows, כגון **exe** ו-**obj**.

- קבצי מערכת:
  - **trie**קובץ מסוג

docname*.trie**: קובץ זה יכיל את האינדקס למסמך בשם ***docname.xxx. אינדקס זה מיושם כעץ רב-בנים מסוג **trie**.

  - **stop**קובץ מסוג

docname*.stop**: קובץ טקסט (אופציונאלי) שמכיל רצף מלים, מילה אחת בלבד בכל שורה, שמשמש כמסנן מלים במסמך; כלומר, בזמן אינדוקס המסמך ***docname.xxx, המערכת תדלג על המלים שמופיעות בקובץ מסוג **stop**. מילים מהסוג הזה נקראות **stop words**.

בנוסף לקבצי **stop** השייכים למסמכים במאגר, יכול להיות קיים קובץ מסוג **stop**, שהוא כללי למאגר כולו. שמו של הקובץ הזה יהיה **stop.lst**. ראה בהמשך תיאור של השימוש בקובץ **stop** הכללי הזה.

שלצורך אינדוקס של מסמך כלשהו במאגר *חייב* להיות קיים קובץ מסוג **stop**.

המערכת תאנדקס מסמך תוך יצירת עץ **trie**:

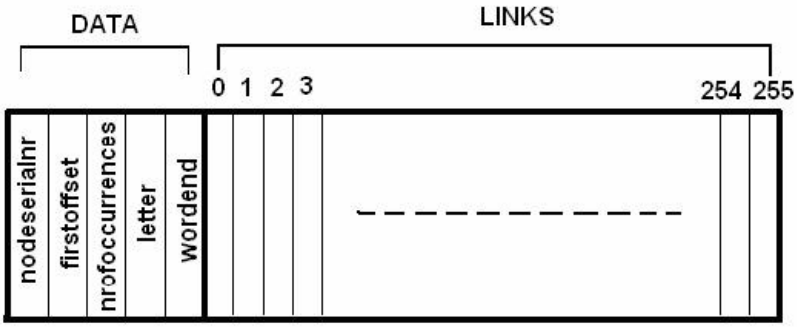
(א) המערכת תאנדקס מסמך מסוים בשם ***docname.xxx***, תוך דילוג על המלים שמופיעות בקובץ בשם ***docname*.stop**. אם המשתמש מעוניין לא לדלג על אף מלה במסמך, הוא יהיה חייב ליצור קובץ *ריק* מסוג **stop** (קובץ שאין בו אפילו מלה אחת).

(ב) אם קובץ בשם ***docname*.stop** לא קיים, המערכת תשתמש בקובץ הכללי **stop.lst**.

(ג) אם גם הקובץ stop.lst לא קיים, המערכת תיצור קובץ ***docname*.stop** *ריק* ותאנדקס את המסמך בלי לדלג על אף מלה בטקסט.

בהתאם למה שלמדנו על מבנה הנתונים **trie**, המימוש שלנו יהיה בצורה הבאה:

נייצג עץ **trie** כמערך שקודקודי העץ הם איבריו. לכל קודקוד (**trienode**) יהיו שני שדות, שדה DATA ושדה LINKS.



איור מס׳ 2: מבנה של קודקוד בעץ trie

- DATAהשדה מורכב מחמישה שדות:

שם השדה	טיפוס	משמעות
nodeserialnr	long int	מספר סידורי של הקודקוד במערך הקודקודים של עץ ה–trie
firstoffset	long int	מיקום של המופע הראשון של המילה בקובץ הטקסט
nrofoccurrences	long int	מספר מופעים של המילה בקובץ הטקסט
letter	unsigned char	תו מה–256 תוים של ה–ASCII
wordend	boolean	האם התו הזה מציין סוף מילה?

- LINKS השדה הוא מערך של 256 איברים. המספר הסידורי של כל איבר במערך הזה הוא הערך הבינארי של התו המתאים בטבלת ה–extended ASCII (מ–0 עד 255); כלומר, תו כלשהו מ–256 התווים של ה–ASCII ישמש כמעין מפתח חיפוש במערך הזה. כל ערך מה–256 שבמערך ה–LINKS, מציין את המיקום של קודקוד הבן במערך הקודקודים. ייתכנו תוים שלא מצביעים לאף קודקוד בן – במקרה כזה הערך המאוחסן בהם יהיה 1- . כלומר, בכל אחד מהאיברים של המערך הזה יהיה מאוחסן ערך מסוג long int שישמש כמעין מצביע לקודקוד בן של הקודקוד הנוכחי (ה–״מצביע״ הזה הוא המספר הסידורי של קודקוד הבן במערך קודקודי עץ ה–trie). התו המתאים למספר הסידורי של האיבר במערך ה–LINKS יהיה ערך השדה letter שבשדה DATA של קודקוד הבן.

איור מס׳ 3: עץ trie של המלים ״abab״, ״abax״, ו–״axb״ (מספור הקודקודים מציין את מיקומם במערך הקודקודים)

תיאור המבנה של קובץ מסוג **trie**:
מערך קודקודי עץ ה–**trie** יאוחסן בקובץ מסוג **trie** בו כל רשומה היא קודקוד של העץ. רשומות אלה תאוחסנה בתוך בלוקים של קובץ ה–**trie**, כאשר גורם הגושיות (ה–blocking factor) של כל בלוק בקובץ מסוג זה, הוא 10. משמעות הדבר שכל אחת מעשר הרשומות בבלוק, תהיה קודקוד בעץ ה–**trie**.
האורך של כל בלוק בבתים יהיה (**trienode**)sizeof * 10. על מנת לקרוא בלוק מהקובץ או לכתוב בלוק לקובץ, חייבים לדאוג להקצאת חוצץ בגודל בלוק. בזמן איחסון עץ ה–**trie**, יהיה צורך בלחלק את מערך הקודקודים בהתאם, לפי בלוקים, ולכתוב כל בלוק אחד אחרי השני. בזמן הבאת עץ ה–**trie** מהקובץ, יהיה צורך לשחזר את מערך הקודקודים.

הערה: זה ברור שעל מנת לטפל באופן מסודר בקריאה/כתיבה פיסית (רמת הבלוק), לעומת קריאה/כתיבה לוגית (רמת ה–**trienode**), יהיה צורך להגדיר struct (או מחלקה) מתאים.

תאור מפורט של המשימות שעליך לבצע בשלב מס׳ 1 של המיני–פרויקט

משימתכם בשלב זה של המיני-פרויקט להגדיר וליישם את התשתית (מבנה + פעולות) של מאגר המסמכים.

מחלקת triesite (שמאפיינת מאגר של מסמכים)

- החלק המבני של המחלקה

לכל אובייקט מסוג זה יהיו לפחות השדות הבאים:

- sitename – מסלול מלא (fullpathname) של התיקייה בה המאגר יאוחסן – string
- doclist – רשימה (list) של אובייקטים מסוג triedoc, של כל המסמכים במאגר.
- mounted – boolean
- mountype – char (ראה פונקציה mount)
- כל שדה נוסף, לפי הצרכים התכנוניים/תכנותיים שלכם.

- פונקציות ליישום המחלקה

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

- triesite () (default constructor)

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג triesite על כל השדות שלו בלי לקשר לו אף שם של מאגר (כלומר, ערכו של השדה sitename יהיה מחרוזת ריקה). השדה doclist יהיה רשימה ריקה. כל יתר השדות יותחלו עם ערכי default מתאימים.

- triesite (string, char, char)

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג triesite על כל השדות שלו, לפי הפרוט הבא:

– הפרמטר הראשון: שם המאגר. המחרוזת הזאת יכולה להיות

(א) שם של תיקייה ב-current directory (למשל, "mydocsite").

(ב) מסלול מלא של תיקייה (למשל, "D:\home\docs\mydocsite") שהחלק הבסיסי שלו הוא שם המאגר ("mydocsite" בדוגמה שלנו).

השם הזה, בפורמט של מסלול מלא יהיה הערך של השדה sitename.

– הפרמטר השני: קוד שקובע אחד משני דברים:

(א) cאם הקוד '' (או 'C'), הפונקציה create (ראה בהמשך) תופעל על מנת ליצור תיקייה ששמה שם המאגר.

(ב) mאם הקוד הוא '' (או 'M') הפונקציה mount תופעל על מנת לפתוח מאגר קיים, לשימוש.

בשני המקרים, אם שם המאגר שהתקבל כפרמטר ראשון הוא שם של תיקייה ב-current directory, הוא יומר למסלול המלא (full pathname) שלו ויועבר לפונקציה המתאימה (create או mount). אם שם המאגר שהתקבל כפרמטר ראשון הוא מסלול מלא של תיקייה, הוא יועבר כמות שהו לפונקציה המתאימה.

- הפרמטר השלישי: פרמטר אופציונאלי (רלוונטי רק במקרה שהפרמטר השני הוא 'm'). פרמטר זה הוא קוד שיועבר כפרמטר השני של הפונקציה mount (ראה פרוט בהמשך).

הערה: יש פונקציות בנויות בתוך Visual Studio C++ שמאפשרות לבצע פעולות שונות על תיקיות. הסבר ודוגמאות עליהן ניתן למצוא ב-MSDN.

- void create(string)

תפקידה של פונקציה זו ליצור בפועל תיקיה ששמה שם המאגר.

הפרמטר חייב להיות שם של מאגר בצורתו של מסלול מלא (full pathname). שם זה לא יכול להיות שם של מאגר שכבר קיים; אחרת, תיבצר exception מתאים.

מתפקידיה של פונקציה זו לאתחל את השדה doclist כרשימה ריקה.

- void mount(string,[char])

תפקידה של פונקציה זו לפתוח מאגר *קיים*.

- הפרמטר הראשון חייב להיות שם של מאגר בצורתו של מסלול מלא (full pathname). המאגר הזה *חייב* להיות קיים.

– הפרמטר השני אופציונאלי, וקובע אחד משני דברים:

(א) qאם הקוד '' (או 'Q'), הפונקציה הזאת תפתח את המאגר לשאילתות (**q**ueries) בלבד.

(ב) mאם הקוד הוא '' (או 'M') הפונקציה הזאת תפתח את המאגר לתחזוקה (**m**aintenance), מה שמאפשר לבצע כל פעולה על המאגר ועל כל אחד מהמרכיבים שלו.

אם פרמטר זה לא יופיע בקריאה לפונקציה זו, ערך ה-default שלו יהיה 'q'.

כחלק מתפקידיה של הפונקציה הזאת, כל מסמך שבמאגר חייב להיות מיוצג ע"י אובייקט מסוג triedoc ברשימה doclist; כלומר, לכל מסמך שבמאגר ייווצר אובייקט מסוג triedoc שיוכנס לרשימה doclist.

- void unmount()

תפקידה של פונקציה זו לסגור את המאגר הקשור לאובייקט הזה.

- triedoc* docexists(string)

תפקידה של פונקציה זו לבדוק האם מסמך מסוים קיים במאגר.

הפרמטר הוא שם אפשרי של מסמך, ללא סיומת.

הפונקציה תחפש האם יש איבר ברשימה doclist ששם המסמך בו הינו המחרוזת שהתקבלה כפרמטר. אם החיפוש מוצלח, הפונקציה תחזיר מצביע לאובייקט מסוג triedoc שמצאה ברשימה; אחרת, תחזיר NULL.

- void docupload(string,char)

תפקידה של פונקציה זו להעלות מסמך למאגר.

הפרמטר הראשון חייב להיות שם של מסמך שמעלים למאגר. מסמך עם שם כזה לא יכול להיות קיים במאגר.

הפרמטר השני הוא קוד בעל שני ערכים אפשריים: 'c' (או 'C') שמבקש להעתיק את הקובץ למאגר, או 'm' (או 'M') שמבקש להעביר את הקובץ למאגר.

הפונקציה הזאת תסרוק את הרשימה doclist על מנת לבדוק עם מסמך עם שם כזה לא קיים במאגר; אם קיים, תיווצר exception מתאים.

לאחר בדיקה זו, הפונקציה הזאת תיצור אובייקט מתאים מסוג triedoc; אובייקט זה ייצג את המסמך החדש שמוכנס למאגר, בלי עדיין לאנדקס אותו. פעולה זו תתבצע תוך שימוש בפונקציה putdoc של triedoc. לאחר מכן, האובייקט הזה יוכנס לרשימה doclist (בלי פעולה זאת, מבחינה לוגית המסמך לא נמצא במאגר).

- string docdownload(string,[string])

באמצעות פונקציה זו יהיה אפשר להעתיק מסמך ספציפי מהמאגר ל-current directory או לתיקיה שמצוינת על ידי הפרמטר השני. הפונקציה הזאת תחפש ב-doclist את המסמך המבוקש ותעתיק (היא לא מעבירה !) אותו באמצעות getdoc.
הפרמטר הראשון: מחרוזת שצריכה להיות שם של מסמך שקיים במאגר.
הפרמטר השני הוא אופציונאלי. אם לא מופיע, המסמך יועתק ל-current directory. אם הוא מופיע, חייב להיות מסלול מלא (full pathname) של תיקיה אליה המסמך יועתק. לצורך זה, הפונקציה הזאת תשתמש בפונקציה getdoc של האובייקט המתאים מסוג triedoc.
ערך מוחזר: המסלול המלא של הקובץ אחרי הורדתו.

מחלקת triedoc

- החלק המבני של המחלקה

לכל אובייקט מסוג זה יהיו לפחות השדות הבאים:
- docname – שם קובץ המסמך (ללא סיומת).
- trierootnode – קודקוד השורש של ה-trie.
- triebuff – חוצץ בגודל בלוק, שיכיל את הבלוק הנוכחי של ה-trie.
- trienodesarray – מערך דינאמי של קודקודים, לשימוש בזמן יצירת עץ ה-trie בלבד (בזמן חיפוש, למערך הזה לא יהיה שימוש כלל; כלומר, הוא יהיה ריק מתוכן). ראה כאן דוגמאות של מערך דינאמי בשפת C++.
- כל שדה נוסף, לפי הצרכים התכנוניים/תכנותיים שלכם.

- פונקציות ליישום המחלקה
  - triedoc () – default constructor

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג triedoc על כל השדות שלו בלי לקשר לו אף שם של קובץ (כלומר, ערכו של השדה docname יהיה מחרוזת ריקה). כל יתר השדות יותחלו עם ערכי default מתאימים.

- void triedoc (string, string)

תפקידה של הפונקציה הבונה הזאת לאתחל אובייקט מסוג triedoc על כל השדות שלו, לפי הפרוט הבא:
- הפרמטר הראשון: מסלול מלא (full pathname) של תיקיית מאגר המסמכים.
- הפרמטר השני: שם של מסמך.

הפונקציה putdoc תופעל על מנת להוסיף למאגר את המסמך ששמו התקבל כפרמטר השני.

- void putdoc(string,string,[char])

תפקידה של הפונקציה הזאת להוסיף מסמך למאגר. הוספה זו תעשה *ללא* אינדוקס; משמעותו של דבר שכל מסמך שהועלה למאגר אבל עדיין לא אונדקס, אינו נגיש דרך מנגנון החיפוש שאנחנו נפתח בשלב 3 של המיני-פרויקט.
- הפרמטר הראשון: מסלול מלא של תיקיית מאגר המסמכים.
- הפרמטר השני: שם של מסמך. השם הזה חייב להיות *עם* סיומת. השם הזה יכול להתייחס למסמך ב-current directory (למשל, myfile.txt) או יכול להיות מסלול מלא של קובץ שהמרכיב הבסיסי שלו הוא שם המסמך (למשל, D:\home\usr\haim\myfile.txt).
- הפרמטר השלישי אופציונאלי: הוא קוד בעל שני ערכים אפשריים: 'c' (או 'C') שמבקש להעתיק את הקובץ למאגר, או 'm' (או 'M') שמבקש להעביר את הקובץ למאגר. ברירת המחדל של הפרמטר הזה היא 'c'.

הפונקציה הזאת תבצע את הדברים הבאים:
(א) תיצור תיקיה ששמה כשם המסמך, בתוך תיקיית המאגר.
(ב) תעתיק (או תעביר) את קובץ המסמך לתיקיה שנוצרה ב-(א).
(ג) תעתיק (או תעביר) קובץ עם שם המסמך, עם סיומת stop, לתיקיה שנוצרה ב-(א). קובץ זה אמור להימצא באותה תיקיה ממנה המסמך מועתק (או מועבר). אם קובץ stop לא נמצא, הפונקציה הזאת תבצע מה שמוסבר כאן בקשר לקובצי stop.
(ד) השדה docname יקבל את שם המסמך *ללא* סיומת; כל יתר השדות יותחלו עם ערכי default מתאימים.
הערה: פונקציה זו *לא* בודקת את קיומו של המסמך במאגר (כי אין לה גישה לרשימה doclist שבאובייקט מסוג triesite) בהנחה שבדיקה כזאת נעשתה לפני קריאתה. אם בכל זאת מתברר שקיימת תיקיה ששמה שם המסמך, הפונקציה תזרוק exception מתאימה.

- void getdoc(string,[string])

תפקידה של פונקציה זו להעתיק את המסמך שבמאגר לתיקיה מתאימה.
- הפרמטר הראשון: מסלול מלא של תיקיית מאגר המסמכים.
- הפרמטר השני (אופציונאלי): מסלול מלא של תיקיה אליה יועתק המסמך מהמאגר. ברירת המחדל של הפרמטר הזה היא ה-current directory.

הערה: בשלבים הבאים נוסיף פונקציות לשתי המחלקות הנ"ל. כמובן שגם אתם תוכלו להוסיף פונקציות משלכם בהתאם להחלטותיכם התכנוניות.

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

משימתכם בשלב זה של המיני-פרויקט להגדיר וליישם פונקציות נוספות במחלקות triesite ו-triedoc.

במחלקת triesite

- void del([char])
תפקידה של פונקציה זו למחוק את המאגר.
- הפרמטר אופציונאלי והוא קוד עם שני הערכים האפשריים הבאים: 'l' (או 'L') אם הכוונה למחיקה לוגית, 'p' (או 'P') אם הכוונה למחיקה פיסית (ברירת המחדל היא 'l').

הפונקציה הזאת תבצע מחיקה של כל המסמכים שבמאגר בהתאם לערך הפרמטר. כמו שנאמר בשלב מס׳ 1, כל המסמכים שבמאגר מיוצגים ע״י אובייקטים מסוג triedoc, שברשימה doclist. המחיקה של כל אחד מהם תתבצע תוך שימוש בפונקציה del של triedoc. אם המחיקה פיסית, גם התיקייה של המאגר כולו חייבת להימחק. שים לב שמחיקת תיקיית המאגר כולו תוכל להתבצע רק אם קודם לכן התבצע unmount על המאגר.
שים לב שמחיקת המאגר *לא* גורמת להריסת האובייקט; אלא, לאתחול מחדש של כל השדות הרלוונטיים. אם הייתה בעיה כלשהי בביצועה ,הפונקציה תיצור exception מתאים.

- void docdel(string,[char])
תפקידה של פונקציה זו למחוק מסמך אחד מהמאגר.
- הפרמטר הראשון: שם של מסמך שחייב להיות במאגר.
- הפרמטר השני אופציונאלי (ברירת המחדל היא הערך 'l') והוא קוד עם שני הערכים האפשריים הבאים: (א) 'l' (או 'L') אם הכוונה למחיקה לוגית או 'p' (או 'P') אם הכוונה למחיקה פיסית.

אם המסמך המבוקש קיים במאגר, המחיקה תתבצע תוך הפעלת הפונקציה del של triedoc על האובייקט מסוג triedoc שמתקבל מהפעלת docexists של triesite. לאחר המחיקה, אם המחיקה פיסית, הפונקציה תמחק את אותו אובייקט triedoc מהרשימה doclist.

- void putstopfl(string)
תפקידה של פונקציה זו להוסיף (או להחליף) את הקובץ stop.lst של המאגר.
- הפרמטר יהיה שם של קובץ מסוג stop. שם זה יכול להיות משני סוגים:
(א) שם של קובץ מסוג stop שנמצא ב-current directory (למשל, mywords.stop).
(ב) מסלול מלא של קובץ (למשל, D:\home\docs\mywords.stop) שהחלק הבסיסי שלו הוא שם של קובץ מסוג stop (mywords.stop בדוגמה שלנו).
בשני המקרים, הפונקציה תעתיק את הקובץ מסוג stop ששמו התקבל כפרמטר, לתיקיית המאגר תוך כדי שינוי שמו לשם הסטנדרטי stop.lst.
כידוע לנו, הקובץ stop.lst ישמש לאינדוקס מסמך שעבורו לא הועלה קובץ stop משלו.

- void docidx(string)
תפקידה של פונקציה זו לאנדקס מסמך שכבר הועלה למאגר.
- הפרמטר: שם של מסמך (ללא סיומת) שחייב להיות במאגר.

הפונקציה הזאת תפעיל את הפונקציה idx על האובייקט מסוג triedoc שהוחזר על ידי הפונקציה docexists.

במחלקת triedoc

- void del(string,[char])
תפקידה של פונקציה זו למחוק מסמך מהמאגר.
- הפרמטר הראשון: מסלול מלא של תיקיית מאגר המסמכים.
- הפרמטר השני הוא אופציונאלי (ברירת המחדל היא הערך 'l') והוא קוד עם שני הערכים האפשריים הבאים:

(א) 'l' (או 'L') אם הכוונה למחיקה לוגית: במחיקה לוגית מוחקים את קובץ ה-trie בלי למחוק את המסמך עצמו; כלומר, המסמך ימשיך להתקיים במאגר אבל לא יהיה נגיש באמצעות שאילתות. אם ה-trie כבר לא קיים, הפונקציה תאתחל את כל השדות הרלוונטיים באובייקט, עם ערכי default מתאימים.
(ב) 'p' (או 'P') אם הכוונה למחיקה פיסית: במחיקה פיסית, גם המסמך עצמו נמחק, כולל התיקייה בה הוא נמצא בתוך המאגר. כל השדות הרלוונטיים באובייקט יותחלו עם ערכי default מתאימים.

- void idx(string)
הפרמטר הוא מסלול מלא של תיקיית מאגר המסמכים.
תפקידה של פונקציה זו ליצור trie למסמך המיוצג על ידי האובייקט מסוג trie, ולאחסן אותו בקובץ trie מתאים.
בהתאם לקובץ stop, הפונקציה תיצור את ה-trie תוך שימוש בשדות trierootnode ו-trienodesarray של האובייקט מסוג triedoc. לאחר יצירת ה-trie, תופעל הפונקציה flush על מנת לאחסן את ה-trie בתיקיה של המסמך שבתוך המאגר.

- void flush(string)
הפרמטר הוא מסלול מלא של תיקיית מאגר המסמכים.
תפקידה של פונקציה זו להעתיק את ה-trie שבשדה trienodesarray לקובץ trie מתאים בתיקיה של המסמך המיוצג על ידי האובייקט מסוג triedoc. על מנת לבצע את זה, הפונקציה תצטרך לשים ב-triebuff מספר קודקודים של ה-trie כמספר ה-blocking factor של קבצים מסוג trie.

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

ביטוי חיפוש:
מרכיב עיקרי במה שצריך לעשות בשלב הזה של המיני-פרוייקט הוא *ביטוי חיפוש*.

ביטוי חיפוש הוא מחרוזת שבאמצעותה מוגדר תנאי החיפוש של שאילתא.
התחביר של ביטוי מסוג זה, כדלהלן:

```
<searchexp> ::= <searchexp1> | <searchexp2>
<searchexp2> ::= '(' <searchexp1> ')' | <searchexp2> <logical operator> <searchexp2> | '(' <searchexp2> ')'
<searchexp1> ::= <word> | <wildcard> | <searchexp1> <word> | <searchexp1> <wildcard>
<word> ::= <letter> | <word> <letter>
<wildcard> ::= <word> '*'
<logical operator> ::= '&' | '|'
<letter> ::= ['a'-'z'] | ['A'-'Z']
```

כלומר:
ביטוי חיפוש יכול להיות:
(א) wildcardרצף של מילים כלשהן, רצף של -ים, או רצף של ערבוביה של שניהם.
לדוגמה:
1. נניח שאנו רוצים לברר האם לפחות אחת מהמילים student, course, ו-home נמצאות במסמך. ביטוי חיפוש מתאים לזה הוא המחרוזת

"student course home"

2. studנניח שאנו רוצים לברר האם קיימות במסמך מילים שמתחילות ב- או מילים שמתחילות ב-cou. ביטוי חיפוש מתאים לזה הוא המחרוזת

"stud* cou* "

3. homeנניח שאנו רוצים לברר האם קיימת במסמך המילה או מילים שמתחילות ב-stud. ביטוי חיפוש מתאים לזה הוא המחרוזת

"home stud* "

(ב) רצף מסוג (א) בתוך סוגריים. למשל:

"(home stud*)"

(ג) ביטויים מסוג (ב) קשורים באמצעות פעולה לוגית '&' או '|'; ביטויים לוגיים כאלה יכולים להיות ממורכבות כלשהי והיררכיית סוגריים כלשהי. למשל:

"((home stud*) & (course name room))"
"(((name teach*) | (cour* name)) & (room building))"

<u>משמעותו של דבר</u> <u>היא שמסמך יענה לשאילתא אם יהיה ניתן למצוא בעץ ה-trie שלו את כל המילים שביטוי החיפוש מייצג.</u>

<u>בקשר למימוש של ביטוי החיפוש</u>:
הגדרת ה-searchexp1 אומרת שביטוי חיפוש יכול להיות רצף של מילים, רצף של wildcard-ים או רצף של ערבוביה של שניהם. משמעות הדבר היא שמסמך יענה לביטוי מסוג זה אם לפחות מילה אחת (או wildcard אחד) מתוך הרצף שבביטוי נמצאת במסמך.
הגדרת ה-searchexp2 אומרת שביטוי חיפוש יכול להיות ביטוי מורכב מתת-ביטויים מחוברים ע"י אופראטורים לוגיים. משמעות הדבר היא שמסמך יענה לביטוי מסוג זה אם אפשר למצוא במסמך מילים שביחד הופכות את הביטוי לנכון מבחינה לוגית.

<u>הציון הכולל של השלב הזה תלוי במימוש הטיפול בביטוי החיפוש</u>:
כל מי שיממש את הטיפול בביטוי החיפוש בצורתו כ-searchexp1 *בלבד* יקבל 90% מהציון הכולל של השלב הזה.
כל מי שיממש *<u>גם</u>* את ביטוי החיפוש בצורתו כ-searchexp2 יקבל 100% מהציון הכולל של השלב הזה.

<u>משימתכם בשלב זה של המיני-פרויקט להגדיר וליישם פונקציות חיפוש במאגר המסמכים, במחלקת triesite וגם ב-triedoc.</u>

<u>במחלקת triesite</u>

- list<string> listdoc([int])
תפקידה של פונקציה זו לאפשר קבלת רשימת שמות של מסמכים שבמאגר.
  - <u>הפרמטר, אופציונאלי,</u> הוא מספר שערכו יכול להיות אחד משלושה: [0,1,2]
  1 – בקשה לרשימת שמות של מסמכים שכבר נעשה להם אינדוקס; כלומר, מסמכים שניתן לבצע שאילתות עליהם.
  2 – בקשה לרשימת שמות של מסמכים שעדיין לא נעשה להם אינדוקס.
  0 – בקשה לרשימת שמות של כל המסמכים שבמאגר; לכל מסמך שכבר נעשה לו אינדוקס, המחרוזת "**" תוצמד לשמו.
  ברירת המחדל של ערך הפרמטר תהיה 0.
  הפונקציה תחזיר רשימת שמות המסמכים המבוקשים.
<u>הערה</u>: ברור שבאמצעות docdownload יהיה אפשר להוריד כל מסמך ששמו מופיע ברשימת הפלט של הפונקציה הזאת.

- string expsearch (string,string)
תפקידה של פונקציה זו לאפשר קבלת שורת טקסט בה ביטוי חיפוש מופיע בפעם הראשונה במסמך ספציפי.
  - <u>הפרמטר הראשון</u> הוא מחרוזת שערכה שם של מסמך שבמאגר.
  - <u>הפרמטר השני</u> הוא מחרוזת שערכה ביטוי חיפוש.
אם ביטוי החיפוש לא נמצא במסמך, הפונקציה תחזיר מחרוזת ריקה; בכל מקרה אחר, הפונקציה תחזיר שורת טקסט בה ביטוי החיפוש מופיע בפעם הראשונה במסמך.
פונקציה זו תחפש ברשימת ה-doclist את האובייקט triedoc המתאים לשם המסמך שהתקבל כפרמטר ראשון, ותשתמש בפונקציה expsearch של המחלקה triedoc על מנת לבצע את החיפוש בפועל במסמך המבוקש.

- int expcount (string,string)
תפקידה של פונקציה זו לאפשר לדעת כמה פעמים ביטוי חיפוש קיים במסמך ספציפי.
- הפרמטר הראשון הוא מחרוזת שערכה שם של מסמך שבמאגר.
- הפרמטר השני הוא מחרוזת שערכה ביטוי חיפוש.

אם ביטוי החיפוש לא נמצא במסמך, הפונקציה תחזיר 0; בכל מקרה אחר, הפונקציה תחזיר את מספר הפעמים שניתן למצוא את ביטוי החיפוש במסמך.
פונקציה זו תחפש ברשימת ה-doclist את האובייקט triedoc המתאים לשם המסמך שהתקבל כפרמטר ראשון, ותשתמש בפונקציה expcount של המחלקה triedoc על מנת לבצע את החיפוש בפועל במסמך המבוקש.

- list<string> doclookup(string)
תפקידה של פונקציה זו לאפשר קבלת רשימת שמות של מסמכים שעונים לשאילתא מסוימת.
- הפרמטר הוא מחרוזת שערכה ביטוי חיפוש.

הפונקציה תחזיר רשימת שמות של מסמכים שעונים לשאילתא; כלומר, מסמכים שעונים לביטוי החיפוש שהתקבל כפרמטר.
הערה: ברור שבאמצעות docdownload יהיה אפשר להוריד כל מסמך ששמו מופיע ברשימת הפלט של הפונקציה הזאת.

במחלקת triedoc

- string expsearch (string, string)
תפקידה של פונקציה זו לאפשר קבלת שורת טקסט בה ביטוי חיפוש מופיע בפעם הראשונה במסמך המיוצג ע"י אובייקט מסוג triedoc.
- הפרמטר הראשון הוא מסלול מלא של תיקיית מאגר המסמכים.
- הפרמטר השני הוא מחרוזת שערכה ביטוי חיפוש.

אם ביטוי החיפוש לא נמצא במסמך, הפונקציה תחזיר מחרוזת ריקה; בכל מקרה אחר, הפונקציה תחזיר שורת טקסט בה ביטוי החיפוש מופיע בפעם הראשונה במסמך.

- int expcount (string, string)
תפקידה של פונקציה זו לאפשר לדעת כמה פעמים ביטוי חיפוש קיים במסמך המיוצג ע"י אובייקט מסוג triedoc.
- הפרמטר הראשון הוא מסלול מלא של תיקיית מאגר המסמכים.
- הפרמטר השני הוא מחרוזת שערכה ביטוי חיפוש.

אם ביטוי החיפוש לא נמצא במסמך, הפונקציה תחזיר 0; בכל מקרה אחר, הפונקציה תחזיר את מספר הפעמים שניתן למצוא את ביטוי החיפוש במסמך.

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

משימתכם בשלב זה של המיני-פרוייקט להגדיר וליישם פונקציות נוספות במחלקות triesite ו-triedoc.

במחלקת triesite

- void idxupdate(string, string, int)
תפקידה של פונקציה זו לעדכן אינדוקס של מסמך קיים במאגר, או לאנדקס את המסמך מחדש.
הפרמטר הראשון הוא שם של מסמך (ללא סיומת) שחייב להיות במאגר.
הפרמטר השני הוא מחרוזת המכילה מילה אחת או יותר מופרדות ע"י רווחים או פסיקים.
הפרמטר השלישי הוא קוד בעל ארבעה ערכים אפשריים: 1,2,3,4.
קודם כל, הפונקציה תפרק את מחרוזת הפרמטר השני לרשימה של מילים.
תלוי בערך של הפרמטר השלישי, הפונקציה תפעל כך:
(א) אם ערך הפרמטר השלישי הוא 1 או 2, הפונקציה תפעיל את הפונקציה docidxupdate על האובייקט מסוג triedoc שמוחזר על ידי הפונקציה docexists.
(ב) stopאם ערך הפרמטר השלישי הוא 3, הפונקציה תוסיף את המילים לקובץ של המסמך באמצעות הפונקציה docstopupdate של האובייקט מסוג triedoc שמוחזר על ידי הפונקציה docexists, תמחק את המסמך לוגית, ותאנדקס אותו מחדש.
(ג) docstopupdateאם ערך הפרמטר השלישי הוא 4, הפונקציה תשתמש בפונקציה של האובייקט מסוג triedoc שמוחזר על ידי הפונקציה docexists על מנת למחוק מילים מהקובץ stop של המסמך שנמצאות ברשימת מילים, תמחק את המסמך לוגית, ותאנדקס אותו מחדש.

- void docupdate(string,string, int)
תפקידה של פונקציה זו לעדכן מסמך קיים במאגר, שמחברו החליט לשנותו.
הפרמטר הראשון הוא שם של מסמך (ללא סיומת) שחייב להיות במאגר.
הפרמטר השני הוא מסלול מלא של הגרסה החדשה של המסמך (ברור ששם קובץ ללא סיומת חייב להיות כשם המסמך הקיים במאגר).
הפרמטר השלישי הוא קוד בעל שני ערכים אפשריים: 1,2.
תלוי בערך של הפרמטר השני, הפונקציה תפעל כך:
(א) אם ערך הפרמטר השני הוא 1, הפונקציה תמחק פיסית את המסמך מהמאגר ותעלה את המסמך המעודכן כמסמך חדש באמצעות docuploadהפונקציה , בלי לאנדקס אותו מחדש.
(ב) אם ערך הפרמטר השני הוא 2, הפונקציה תמחק פיסית את המסמך מהמאגר, תעלה את המסמך המעודכן כמסמך חדש באמצעות docuploadהפונקציה , ותאנדקס אותו מחדש.

- void reorg(int)
תפקידה של פונקציה זו לארגן מחדש את כל המאגר. (במקרה הזה, כל המסמכים שנמחקו לוגית, יימחקו פיזית)

הפרמטר הוא קוד בעל שני ערכים אפשריים: 1,2.
תלוי בערך של הפרמטר, הפונקציה תפעל כך:
(ג) אם ערך הפרמטר השני הוא 1, הפונקציה תמחק פיסית את כל המסמכים שמחוקים לוגית.
(ד) אם ערך הפרמטר השני הוא 2, הפונקציה תאנדקס מחדש את כל המסמכים שמחוקים לוגית.

במחלקת triedoc

- void docstopupdate(string, list<string>, int)

הפרמטר הראשון הוא מסלול מלא של תיקיית המאגר.
הפרמטר השני הוא רשימת מילים.
הפרמטר השלישי הוא קוד בעל ערכים אפשריים: 1,2,3.
תלוי בערך של הפרמטר השלישי, הפונקציה תפעל כך:
(א) אם ערך הפרמטר השלישי הוא 1, הפונקציה תוסיף את המילים שברשימה לקובץ stop של המסמך תוך שמירת אי כפילות של מילים בקובץ.
(ב) אם ערך הפרמטר השלישי הוא 2, כל מילה מהמילים ברשימה שתימצא בקובץ ה- stop של המסמך, תמחק ממנו.
(ג) אם ערך הפרמטר השלישי הוא 3, קובץ ה- stop הכללי של המאגר יועתק במקום קובץ ה- stop של המסמך.

- void docidxupdate(string, list<string>, int)

הפרמטר הראשון הוא מסלול מלא של תיקיית המאגר.
הפרמטר השני הוא רשימת מילים.
הפרמטר השלישי הוא קוד בעל שני ערכים אפשריים: 1,2.
תלוי בערך של הפרמטר השלישי, הפונקציה תפעל כך:
(א) אם ערך הפרמטר השלישי הוא 1, הפונקציה תעדכן את עץ ה- trie כך שהאינדוקס לפי המילים שברשימה יבוטל מהעץ; כל מילה מהמילים האלה שלא תימצא בעץ לא תגרום לשום שינוי בו. לאחר עדכון העץ, הפונקציה תשתמש בפונקציה docstopupdate על מנת להוסיף את המילים לקובץ stop של המסמך.
(ב) אם ערך הפרמטר השלישי הוא 2, הפונקציה תעדכן את עץ ה- trie כך שאינדוקס לפי המילים שברשימה יוכנס לעץ. לאחר עדכון העץ, הפונקציה תשתמש בפונקציה docstopupdate על מנת למחוק כל מילה מהמילים ברשימה שתימצא בקובץ stop של המסמך.

תאור מפורט של המשימות שעליך לבצע בשלב מס׳ 5 של המיני-פרויקט

משימתכם בשלב זה של המיני-פרויקט להגדיר ולממש את הממשק הגראפי למערכת לניהול וחיפוש מסמכים שבניתם במשך ארבעת השלבים הקודמים. אתם חופשיים לתכנן את הממשק בצורה שתמצאו לנכון בתנאי שבאמצעותו יהיה אפשרי לבצע את כל הפעולות שמימשתם במשך ארבעת השלבים הקודמים, ובהתאם למצב המערכת (מצב "שאילתות" או מצב "תחזוקה").
חשוב מאוד שהשימושיות שלו תהיה כגורם מרכזי בתכנונו.

בממשק של הצגת תוצאות נרצה לקבל שני סוגים שונים של מידע, אחד פתוח לכולם בשני מצבי ההפעלה של המערכת ("תחזוקה" ו-"שאילתות"), והשני פתוח רק במצב "תחזוקה" :
(א) רשימת שמות של מסמכים שעונים לשאילתא (ב-"תחזוקה" וגם ב-"שאילתות") :

- השורה הראשונה תהיה שורת סיכום של השאילתא, בה יופיעו שני מספרים: מספר המסמכים שעונים לשאילתא וכמה אחוזים הם מתוך סה"כ המסמכים במאגר.
- מייד אחרי השורה הראשונה תופיע רשימת המסמכים שעונים לשאילתא, כולל קטע טקסט מתוך המסמך בו המלה (או ביטוי) שהמערכת חיפשה מופיעה. ליד השם של כל מסמך ברשימת המסמכים האלה יופיע נתון נוסף לפיו רשימה זו תהיה ממוינת. נתון זה יוכל להיות אחד משני סוגים שונים:
  (1) מספר הופעות של המלה (או ביטוי) במסמך.
  או
  (2) אחוזי הופעה מתוך סה"כ המופעים של אותה מלה (או ביטוי) במסמכים האלה יחד.

(ב) רשימת שמות של מסמכים ממוינים לפי הביקוש עבורם (ב-"תחזוקה" בלבד).

- השורה הראשונה תהיה שורת סיכום בה יופיעו המספרים הבאים: סה"כ המסמכים שענו לשאילתות למיניהן, כמה אחוזים הם מתוך סה"כ המסמכים במאגר.
- מייד אחרי השורה הראשונה תופיע רשימת המסמכים שענו לשאילתות למיניהן. ליד השם של כל מסמך ברשימת המסמכים האלה יופיע נתון נוסף לפיו רשימה זו תהיה ממוינת. נתון זה יוכל להיות אחד משני סוגים שונים:
  (1) מספר הפעמים שהמסמך היה חלק מתשובה לשאילתות כלשהן, על המאגר כולו.
  או
  (2) כשנתון מספר הפעמים שמסמך מאונדקס היה חלק מתשובות לשאילתות כלשהן על המאגר כולו, איזה אחוז הוא מספר זה מחיבור יחד של סה"כ הפעמים שכל מסמך מאונדקס בנפרד היה חלק מתשובות לשאילתות כלשהן על המאגר כולו; כלומר, קודם כל מחברים יחד את סה"כ מספר הפעמים שכל אחד מהמסמכים המאונקסים היה חלק מתשובות לשאילתות למיניהן (למאגר כולו) ולאחר מכן מחשבים כמה אחוזים מהסה"כ הזה הוא מספר הפעמים שמסמך מסוים היה חלק מתשובות לשאילתות כלשהן.

על פי נתונים אלו, יהיה אפשר לדעת מה רמת הביקוש של כל מסמך במאגר.

בנוסף לנ"ל, זה הרגע לקבוע ***מנגנון כניסה למערכת***.
(א) mount יהיה משתמש אחד ויחיד שיוכל לבצע mount של המאגר, והוא המשתמש "admin". לצורך ביצוע ה-mount המערכת תבקש שם משתמש וסיסמה. לאחר אישור הסיסמה, "admin" יוכל לבצע את mount תוך בחירת מצב המאגר ("שאילתות" או "תחזוקה"). גם המשתמש היחיד שיוכל לבצע unmount הוא "admin", ורק אחרי אישור שם משתמש וסיסמה ע"י המערכת.
(ב) במצב "שאילתות" הכניסה תהיה חופשית ללא צורך הכנסת שם משתמש וסיסמה; כלומר, כאשר הפעולות היחידות שניתן לבצע על המערכת הן ביצוע שאילתות והצגת תוצאותיהן בצורה שהוסבר לעיל, המערכת לא תדרוש הכנסת שם משתמש וסיסמה.

(ג) במצב "תחזוקה" המערכת תדרוש הכנסת שם משתמש וסיסמה; לאחר אישור הכניסה, המערכת תאפשר לבצע כל פעולה על המאגר.
(ד) admin אותו משתמש "" יהיה היחיד שיוכל לבצע כל פעולת "תחזוקה" (בנוסף לשאילתות, הוא יוכל לבצע upload, index, וכו').

<u>הערות והארות חשובות בקשר לשלב מס' 5</u>

1. user centered design עקרון מרכזי בתכנון מערכות אינטראקטיביות חלונאיות הוא " " או "usage centered design"; כלומר, מערכת שהמשתמש (או אופן השימוש בה) במוקד התכנון שלה. חייבים לתכנן את ה-GUI *לשימושיות* (usability) מרבית – *שימוש פשוט ומובן מאלין*, בלי שזה יבוא על חשבון הפונקציונאליות שהמערכת מספקת למשתמש באמצעות ה-GUI.
היבט שגם כדאי לקחת בחשבון הוא ה*אסתטיקה* – כלומר, *איך ה-GUI נראה*. בכל זאת, ה-GUI שלכם לא חייב להיות "מצוין" או מושלם – מספיק שהוא יאפשר לבצע את הפעולות מהשלבים הקודמים, ומה שדרוש להוסיף בשלב 5, בצורה פשוטה ומובנת מאליה.
כל מי שביצע את השלבים הקודמים תוך פיתוח GUI עבורם, לא צריך לגעת אפילו בשורה אחת של קוד אלא להוסיף את הדברים המעטים שדרושים בשלב 5.
<u>ה-GUI שכבר נעשה שריר וקיים ואין שום סיבה לשנות אפילו שורה אחת של קוד.</u>

2. GUI מהתחלת הדרך, הרעיון היה שלכל אחד משני המצבים של מאגר ("תחזוקה" ו-"שאילתות") יהיו מרכיבים מתאימים ב- (תפריטים למיניהם, כפתורים למיניהם, חלונות שגיאה למיניהם, חלונות להצגת תוצאות, כו').
- <u>במצב "שאילתות":</u>
ה-GUI אמור לאפשר לבצע פעולות איחזור מידע (information retrieval) בלבד, כגון ביצוע שאילתות והצגת תוצאותיהן, הורדת מסמכים מהמאגר, הצגת רשימת המסמכים שבמאגר, וכו'.
עיקר העניין מבחינת ה-GUI לשאילתות הוא חיפוש על המאגר כולו; ברור שה-GUI יכול לאפשר למשתמש גם לבצע חיפוש על מסמך בודד (על סמך בחירה מתוך רשימת שמות של מסמכים, למשל), דבר שהתשתית כבר פותחה בשלב 3. המשתמש יוכל לבטא שאילתא (ביטוי חיפוש) על כל המאגר והמערכת תציג לו את תוצאת השאילתא כרשימת שמות המסמכים העונים לשאילתא ביחד עם קטע טקסט המכיל חלק או כל המילים שבביטוי החיפוש, דבר שהתשתית למטרה זו כבר פותחה בשלב 3.
הדבר שהוספנו בשלב 5 בקשר להצגת תוצאות של שאילתא, הוא כדלקמן:
- שורה אחת של מידע סטטיסטי בקשר למספר המסמכים שעונים לשאילתא וכמה אחוזים הם מסה"כ המסמכים במאגר.
- על סמך מה שכבר כתבתם בשלב 3, ליד השם של כל מסמך שעונה לשאילתא, יוצג מספר הופעות של ביטוי החיפוש במסמך או אותו המספר יופיע כאחוזים של סה"כ ההופעות בכל המסמכים שעונים לשאילתא. מספר זה ישמש למיון המסמכים בהצגת התשובה לשאילתא.
- <u>במצב "תחזוקה":</u>
ה-GUI אמור לאפשר לבצע פעולות ניהול, כגון יצירת מאגר, mount, unmount, הוספת מסמך למאגר, אינדוקס, מחיקה, ארגון מחדש, עדכון מסמך קיים בגרסה חדשה וכו'.
בנוסף לזה, ה-GUI יציג מידע סטטיסטי כדלקמן:
- שורה אחת בקשר למסמכים מאונדקסים שענו לשאילתות בכלל – יופיעו שני מספרים: מספר הפעמים שכל אחד מהם ענה לשאילתות למיניהן במונחים מוחלטים וגם במונחים יחסיים (אחוזים מסה"כ המסמכים שענו לשאילתות למיניהן).
- ליד השם של כל מסמך מאונדקס יוצג מספר הפעמים שאותו מסמך היה תשובה לשאילתות למיניהן, או אותו מספר יוצג כאחוזים של סה"כ הפעמים שכל המסמכים המאונדקסים שבמאגר היו חלק מתשובות לשאילתות למיניהן שהתבצעו על המאגר כולו.
<u>שימו לב</u> שכל מה שצריך לעשות הוא לשמור מונה לכל מסמך מאונדקס, שיעיד על מספר הפעמים שאותו מסמך ענה לשאילתא כלשהי. לאחר מכן, צריך לעבור על המונים של כל הקבצים המאונדקסים, ואז מחשבים את סה"כ ואחוזים של הסה"כ הזה.
כל מונה מסוג זה יאוחסן בקובץ המיועד למטרה זו בלבד. קובץ זה חייב להימחק בזמן מחיקה לוגית של מסמך, ביחד עם קובץ ה-trie.
<u>קבענו שיהיה משתמש מיוחד, אחד ויחיד (ששמו "admin" וסיסמתו "system") שיהיה היחיד שיוכל לבצע פעולות במצב "תחזוקה".</u>

<u>הצעה למימוש הממשק הגראפי – GUI</u>

כמה נקודות למחשבה בקשר לדו"ח הסופי ולהגנת המיני-פרויקט

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 וחיפוש מאגר מסמכים שבנית:
  - תיאור מימושם של כל השלבים, במיוחד אלגוריתמים ומבני נתונים שהשתמשת (למשל, איזה שיטה החלטת להשתמש למחיקת trieמילים מה-). 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')
- סיכום כללי בו תכתוב מה להערכתך למדת במשך המיני-פרויקט, דעותיך בקשר לקורס, צוות ההוראה, האם השו"תים בפורום של הקורס עזרו, הצעות לשיפור, וכו').
- נספח שיכלול מפרטים של כל הפונקציות שכתבת – בפועל, תיעוד המחלקות והפונקציות שכתבת בהתאם לפורמט שביקשנו להשתמש: שם הפונקציה, סוג הערך המוחזר, פרמטרים, ותיאור קצר בקשר למה שהפונקציה עושה.
- נספח שיכלול הצעה לשלב נוסף למיני-פרויקט על סמך השלבים הנוכחיים, ונימוק להצעתך.
- נספח שיכלול הערות, הארות, ביקורות וכו'.
- 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.

הרצה והגנה על המיני-פרויקט:

- הדגמת המיני-פרויקט כולו והסברים על ה- מבחינה תכנונית ותכנותית. GUI
- 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-פרויקט
- שאלות שונות על המיני-פרויקט. להלן רשימה של סוגי השאלות שיכולות להישאל:

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