האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנאות תכנות בשפת C ו־++C (67312 ו־67317)
++C – תרגיל 3

תאריך ההגשה של התרגיל והבוחן התיאורטי: 24 בינואר, 2020, בשעה 23:55;
הגשה מאוחרת (בהפחתת 10 נקודות): **לא קיימת לתרגיל זה.**

נושאי התרגיל: templates, C++ standard library, exceptions.

1 רקע

מימוש של תיבת דואר אלקטרוני הוא למעשה מימוש של פרוטוקול, המאפשר לשני מחשבים (דהיינו, שרתים) "המדברים" באותו הפרוטוקול לשלוח ולקבל הודעות.[1] עובדה זו מאפשרת לשלוח בקלות הודעות המכילות מסרים "זדוניים", שמטרתם לפגוע במשתמש הקצה (בין אם על ידי שליחת וירוסים, קישורים שמאפשרים גניבת זהות, בקשות להעברת כסף וכיוצ"ב).
לכן, כדי למנוע חשיפת משתמשים לסיכונים אלו – נרצה ליצור מנגנון המאפשר "לקטלג" (to classify) הודעות שנדמה כי הן "זדוניות", ולהסב את תשומת לבו של המשתמש לכך. מנגנון זה מוכר בשם Spam Detector.
בפן הטכני, אחת השיטות הפשוטות למימוש מנגנון מסוג זה היא יצירת בנק, או מאגר, של "ערכים רעים" שנרצה לסנן. לכל ערך כאמור נצמיד ניקוד המצביע על חומרת השימוש בערך. אילו הודעה עברה סף מסויים, שמסמן את סכימת הניקוד של כל "הערכים הרעים" שהופיעו בהודעה – אזי מדובר בהודעת Spam.
בתרגיל זה נשתמש בידע שצברנו במהלך הקורס ונשלבו בנושאים כדוגמת גנריות ושימוש בספריה הסטנדרטית של ++C, לצורך מימוש מבנה הנתונים HashMap ושימוש בו. בחלק הראשון של התרגיל, נממש את מבנה הנתונים HashMap, בעוד בחלק השני – נממש אלגוריתם פשוט של Spam Detection המסתמך על העיקרון האמור.

2 הגדרות

להלן נזכיר מספר הגדרות בסיסיות הנוגעות לטבלאות גיבוב, שהוצגו בקורס מבני נתונים:

- **הגדרה**: פונקצית גיבוב (hash function) היא פונקציה שממפה מידע מגודל כלשהוא ("מפתחות") למידע אחר כלשהוא, בגודל סופי. לשם הפשטות, נוכל להניח שמדובר

[1] פרוטוקול מוכר, למשל, הוא POP - https://en.wikipedia.org/wiki/Post_Office_Protocol

בפונקציה מהצורה $h : U \to \{0, \ldots, m-1\}$ כאשר U היא קבוצת איברים כלשהיא, כמו למשל מחרוזות, מספרים וכדומה, ו־$\{0, \ldots, m-1\}$ היא קבוצה סופית של תוצאות אותן ניתן לקבל מהפונקציה (תוצאת פעולת ה־hash).

- **מסקנה מההגדרה הקודמת:** נרצה שכל פונקצית hash, שנסמנה h, תקיים:
  - h תהיה קלה לחישוב;
  - ש־h תמפה כמה שפחות איברים לאותו התא, כדי למנוע התנגשויות רבות.
- **הגדרה**: hash map הוא מבנה נתונים המכיל מיפוי של מפתחות לערכים. המפתחות יכולים להיות מספרים, מחרוזות או כל טיפוס נתונים נתמך אחר – וכך גם הערכים. ברמה האינטרנית, טיפוס נתונים זה עושה שימוש בטבלת גיבוב כדי **למפות** מפתחות לערכים. הייתרון של מבנה נתונים זה, הוא שבהינתן פונקצית גיבוב "טובה", פעולות ההוספה, החיפוש וההסרה שלו מבוצעות בסיבוכיות זמן ריצה <u>ממוצעת</u> של $\Theta(1)$.

3 מימוש HashMap

משהוצגו ההגדרות המקדימות הנדרשות לפתרון התרגיל, נציג להלן מספר נושאים נוספים הנוגעים לפונקציות, טבלאות ומפות גיבוב, להם אתם נדרשים במימוש התרגיל:

3.1 גורם העומס (Load Factor)

לבד מגודל הקלט בפועל, הביצועים של מפת הגיבוב מושפעים משני פרמטרים: גורם עומס העליון וגורם העומס התחתון (upper load factor & lower load factor). גורם העומס מוגדר כך:

$$Load\,Factor = \frac{M}{capacity}, \qquad capacity > 0 \wedge M \geq 0$$

כאשר $M \in \mathbb{N} \cup \{0\}$ מייצג את כמות האיברים שמבנה הנתונים **מכיל <u>כרגע – כלומר בפועל</u>, בעוד** $capacity \in \mathbb{N}$ מייצג את כמות הנתונים **המקסימלית** שניתן לשמור במבנה הנתונים (כלומר, הקיבולת). אם כך, מהם גורמי העומס התחתון והעליון? אלו הגורמים שמציינים עד־כמה נסכים שמבנה הנתונים יהיה ריק או ממלא. כלומר, נרצה שכאשר כאשר נחצה רף מסויים (threshold) – נגדיל או נקטין את הטבלה בהתאם, כך שמצד אחד לא תדרוש זיכרון רב מדי ומהצד השני תוכל להכיל כמה איברים שנרצה.
לשם כך, עת שנחצה (**ממש**) את אותו רף (תחתון או עליון) – נבצע הליך שנקרא re-hashing ובו נגדיל או נקטין את הטבלה, נחשב את ערכי ה־hash בשנית ונמקם שוב את כל האיברים שבטבלה. במילים אחרות, נעתיק את כל הערכים הישנים לטבלה "חדשה".
כברירת מחדל, נגדיר את גורם העומס התחתון להיות $\frac{1}{4}$ ואת גורם העומס העליון להיות $\frac{3}{4}$. בכל מקרה, גורם העומס העליון יהיה בהכרח גדול מהתחתון, ושניהם יהיו בקטע $(0, 1]$.

3.2 גודל הטבלה

ישנן שתי אפשרויות פופולריות לקביעת הגודל המקסימלי (capacity) של טבלאות גיבוב ברגע נתון: שימוש במספרים ראשוניים או בחזקות של 2. האופציה הראשונה, דהיינו מספרים ראשונים, טובה מכיוון שפיזור האיברים נעשה בצורה הרבה יותר אחידה. מננגד, הקושי שבבחירה זו הוא שאין דרך קלה לבצע re-hash – כלומר לא נוכל, למשל, לבצע חישוב אריתמטי פשוט כמו העלאה בחזקה, כדי לקבל את גודל הטבלה החדש. מנגד, בעוד

שהאופציה השנייה – דהיינו שימוש בחזקות של 2 – אינה יוצרת פיזור אחיד טוב מספיק, מאוד קל לבצע בעניינה re-hashing וכן ניתן לחשב בה את המיקום של כל איבר בדרך מהירה יותר, על ידי bitwise operators (נראה זאת בהמשך). **לפיכך, בתרגיל זה, גודל הטבלה יהיה חזקה של 2, כאשר הגודל ההתחלתי הוא 16. שימו לב שבהכרח** $capacity \geq 1$**.**

3.3 פונקצית הגיבוב

כאמור לעיל, עלינו לבחור פונקצית גיבוב "טובה" כדי להגיע למבנה נתונים שפועל ביעילות טובה. פונקצית הגיבוב שנבחר מאוד בסיסית, והיא:

$$h(x) = x \, mod \, capacity, \qquad capacity \in \mathbb{N}$$

כאשר $capacity$ מייצג את גודל הטבלה ו־x הוא **ייצוג מספרי** של הערך שנרצה לשמור בטבלה. **כדי להמיר מחרוזת, מספרים וכיוצ"ב למספר שלם, תוכלו להשתמש בפונקציה** `std::hash`**. ניתן להניח שכל טיפוס שתדרשו להכיל במפה, יתמוך ב־**`std::hash`**.**
נשים לב שאופרטור ה־modulo ב־C++ אינו פועל באופן זהה לאופרטור ה־modulo המתמטי. למשל, בעוד שב־C++ $(-3) \, mod \, 7 = -3$, התוצאה המתמטית היא כידוע 4. כדי לפתור זאת, יהיה עלינו לחשב את הפונקציה ב־C++ כערך מוחלט. יתרה מכך, אנו עשויים להיתקל בקושי כאשר נחשב את המודולו של INT_MIN, כיוון שאין לו ערך מקסימלי תואם. נפתור זאת על ידי casting ל־long. כלומר (% הוא modulo ב־C++):

$$v \, mod \, size = |(long)v \, \% \, size|$$

לסיום, נציע חלופה אחרת לחישוב פונקצית הגיבוב: נבחין שפונקצית הגיבוב שהגדרנו עושה שימוש ב־$capacity$, שכאמור לעיל הוא חזקה של 2. לכן, נוכל להשתמש באופרטור הלוגי and (מיוצג על ידי &) כדי לחשב את אותו הערך, כלומר:

$$v \, mod \, size = v \, \& \, (size - 1)$$

פתרון זה עדיף, כיוון שאופרטורים לוגים מהירים יותר מאריתמטים – אז חישוב האינדקס יהיה מהיר יותר. בנוסף, גם לא ניזקק יותר לשימוש בערך מוחלט או ל־casting ל־long.

3.4 אלגוריתם המיפוי – שיטת מיפוי פתוח (Open Hashing).

בהמשך לאמור, קל לראות שפונקצית הגיבוב שנבחרה, $h(x)$, תיצור, במוקדם או במאוחר, התנגשויות. ראשית, אם כמות האיברים שנרצה לשמור במבנה הנתונים תהיה גדולה מ־$capacity$, זה ינבע ישירות מעיקרון שובך היונים. אחרת, גם כאשר $capacity$ גדול מכמות האיברים שנרצה לשמור, אנו עשויים להיתקל במקרים בהם לשני ערכים, $x, y \;\; S.t. \;\; x \neq y$, נקבל ש־$h(x) = h(y)$ ובמקרה זה ניתקל בהתנגשות. ישנן שתי שיטות לפתרון התנגשויות:

- **Open hashing**: שיטה המאפשרת לשמור יותר מערך אחד בכל תא. התאים, שנקראים "סלים" (buckets) ובנויים מטיפוס נתונים אחר – לדוגמה מרשימה מקושרת. כך, גם אם יש התנגשות, כל שקורה הוא שהאיבר המתנגש נוסף לרשימה המקושרת.
- **Close hashing**: שיטה לפיה כל תא יכול להכיל רק איבר אחד. במקרים אלו, עלינו למצוא דרך אחרת להתמודד עם התנגשויות ולמפות איברים. [2]

בתרגיל זה, נממש את ה־hash set באמצעות Open hashing.

[2] שיטה מוכרת למימוש closed hashing היא quadric probing. להרחבה ראו: https://en.wikipedia.org/wiki/Quadratic_probing.

4 חלק א' – המחלקה HashMap

בחלק הראשון נממש את המחלקה הגנרית HashMap, שתייצג מיפוי בין מפתחות לערכים ($key \mapsto value$), המבוסס על טבלת גיבוב. כלומר, מבנה הנתונים ימפה בין מפתחות, מסוג KeyT, לערכים, מסוג ValueT. יש לתמוך ב־API הבא:

	התיאור	הערות
	פעולות מחזור החיים של האוביקט	
בנאי ברירת מחדל	בנאי שמאתחל HashMap ריק.	
בנאי 1	בנאי המקבל שני וקטורים, אחד שמכיל ערכי KeyT ואחד שמכיל ערכי ValueT ושומר את הערכים במפה לפי הסדר.	יש לוודא שהוקטורים באותו הגודל. המיפוי יהיה $\forall 0 \leq i < n \;\; keys[i] \mapsto values[i]$
בנאי העתקה	מימוש של בנאי העתקה.	
destructor	מימוש destructor.	
	פעולות	
`size`	הפעולה מחזירה את כמות איברי המפה.	המספר שמחוזר מטיפוס `int`.
`capacity`	פעולה המחזירה את קיבולת המפה.	המספר שמחוזר מטיפוס `int`.
`empty`	פעולה הבודקת האם המפה ריקה.	הפעולה תחזיר `bool`.
`insert`	פעולה המקבלת מפתח וערך, ושומרת את המיפוי שהתקבל.	הפעולה תחזיר אמת אם הערך נוסף בהצלחה.
`containsKey`	הפעולה מקבלת מפתח ובודקת האם הוא קיים במפה.	הפעולה מחזירה `bool`.
`at`	פעולה מקבלת key ומחזירה את ה־$value$ המשוייך אליו.	הפעולה תזרוק חריגה במקרה שה־key לא נמצא.
`erase`	הפעולה מקבלת מפתח ומסירה את הערך המשוייך לו מהמפה.	הפעולה תחזיר אמת אם הערך הוסר בהצלחה.
`getLoadFactor`	פעולה המחזירה את גורם העומס.	המספר שמחוזר מטיפוס `double`.
`bucketSize`	תקבל מפתח ותחזיר את גודל הסל.	הפעולה תחזיר `int`. אם המפתח לא נמצא – יש לזרוק חריגה.
`bucketIndex`	תקבל מפתח ותחזיר אינדקס הסל.	הפעולה תחזיר `int`. אם המפתח לא נמצא – יש לזרוק חריגה.
`clear`	פעולה המסירה את כל איברי המפה.	ה־$capacity$ לא משתנה.
`iterator`	מימוש מחלקת iterator ובנוסף כל הפעולות הנדרשות ל־iterator (לרבות typedefs), בהתאם לשמות הסטנדרטים של C++.	עליכם לממש const forward iterator בלבד. ה־iterator יחזיר `std::pair<KeyT, ValueT>`.
	אופרטורים	
השמה	תמיכה באופרטור ההשמה (=).	השמה לכל ערכי האוביקט.
subscript	תמיכה באופרטור [].	האופרטור יקבל מפתח ויחזיר את הערך המשוייך לו. **אין לזרוק חריגה במקרה זה.**
השוואה	תמיכה באופרטורים == , ! =	בדיקה האם שני סטים מכילים איברים זהים.

דגשים, הבהרות, הנחיות והנחות כלליות:

- את המחלקה עליכם להגדיר בקובץ `HashMap.hpp` (למה לא נוכל להגדירה בקובץ

cpp? כי לא ניתן לכלול בקובץ שעובר קומפילציה ישירה מימוש של templates. תוכלו לקרוא הסבר מפורט יותר על נושא זה ב־C++ Exercise 2).

- כאמור, על המחלקה להיות גנרית. הערך הגנרי הראשון שהמחלקה תקבל הוא טיפוס הנתונים שמייצג את המפתחות, אליו התייחסנו בשם KeyT. הפרמטר השני הגנרי שהמחלקה תקבל הוא סוג הנתונים המייצג את הערכים אליהם המפתחות ממפים, נסמנו כ־ValueT. **ניתן להניח** כי KeyT נתמך על ידי std::hash, כי KeyT ו־ValueT תומכים ב־operator==, operator=, וכן כי יש לשניהם בנאי דיפולטיבי. שימו לב שניתן להיעזר ב־default(T) כדי לקבל את הערך הדיפולטיבי של T.

- **הנכם מחוייבים לשמור את הסלים בתור מערך שמוקצה דינמית.**[3]

- **דרישות זמן ריצה:** על הפעולות insert, at, contains, erase (וכמובן $operator[]$) לפעול בסיבוכיות לינארית ביחס לסל שבו האיבר נמצא – כלומר ב־$O(n)$ כאשר n הוא גודל הסל.

- **למותר לציין, אבל יצויין בכל זאת, שאין להשתמש במחלקות של STL באופן שייתר את פתרון התרגיל. למשל, אין לעשות שימוש ב־std::map במקום לממש מפת גיבוב באופן עצמאי. מנגד, אתם בהחלט רשאים (ומצופה מכם) לעשות שימוש ב־STL.**

- **שימו לב:** ה־API הנ"ל מציג לכם את שמות הפונקציות המחייבות, הפרמטרים, ערכי החזרה וטיפוסיהם. בעת מימוש ה־API, עליכם ליישם את העקרונות שנלמדו בקורס באשר לערכים קבועים (constants) ומשתני ייחוס (references). **שימוש בקונבנציות אלו הוא חלק אינטגרלי מהתרגיל, עליו אתם מקבלים ניקוד.** עיקרון זה נכון בפרט גם לגבי מימוש ה־iterator.

דגשים לגבי מתודות ספציפיות:

- **בנאי 1:** לא ניתן להניח שהוקטורים שיתקבלו יהיו בגודל זהה. אם $keys.size() \neq values.size()$, אזי יש לזרוק חריגה. כמו כן, אם יש ערכי מפתחות כפולים – אזי עליכם לדרוס את הערכים הישנים עם החדשים.

- bucketSize **ו־**bucketIndex **:** יש לזרוק חריגה אם המפתח לא קיים.

- at ו־operator[]: שימו לב להבדלים שבין at ובין operator[], כפי שכבר נידונו בהרצאות ובתרגולים:

  – **קריאה:** בעוד שב־at תיזרק חריגה כאשר ניגש לאיבר שאינו קיים, כשמדובר על operator[] ההתנהגות אינה מוגדרת ותלויה בכם (no-throw guarantee).[4]

  – **כתיבה:** במקרה שבו פונים ב־HashMap::operator[] **לאיבר שלא קיים, עליכם ליצור איבר חדש בטבלה.** גישה זו תאפשר לנו להשתמש בביטויים כמו map["foo"] = "bar".

[3] ניתן לחשוב שאפשר להשתמש בטיפוס נתונים, כמו למשל vector, כדי "להחזיק" את הסלים עצמם. גישה זו אינה מדוייקת שכן אין לנו שליטה על כמות הזיכרון שמערכת ההפעלה תקצה ל־vector. למשל, נניח שנשתמש ב־constructor של vector שמקבל capacity. שימוש זה יבטיח כי הוקטור יוכל להכיל כמות **מינימלית** של איברים בטרם יעבור resize. אלא – ופה העיקר – **שימוש זה אינו כופה על STL שלא להקצות כמות זיכרון גדולה יותר**, אם ההספריה "סבורה" שכך נכול לעשות. מכאן באה הדרישה האמורה.

[4] התנהגות זו תואמת להתנהגות של std::map. ראו למשל: http://www.cplusplus.com/reference/map/map/operator[]/. למעשה, כך עובדים גם טיפוסי נתונים אחרים ב־STL. למשל ב־std::vector מוחזר מקום שלא מוגדר בזיכרון.
כדי להבין מה המשמעות של "התנהגות לא מוגדרת" מומלץ לנסות לבחון את התנהגות std::map.

- **מימוש iterator:** שימו לב כי מימוש ה־iterator יחייב אתכם לממש מחלקת it-erator, ולא ניתן להפנות לפעולות iterator של STL. עליכם לממש את מחלקת ה־iterator כמחלקה פנימית (nested class) של HashMap. זכרו לממש גם את begin ו־end ב־HashMap. בפרט – חשבו איזה וריאציות צריך? האם נצטרך את cbegin ו־cend? מצד שני האם נצטרך את begin ו־end?[5]

5 מימוש SpamDetector

ניגש כעת לחלק השני של התרגיל, והוא – כאמור – מימוש מנגון לזיהוי הודעות Spam. את התוכנית נחבר בקובץ SpamDetector.cpp ופעולתה תהא זו: התוכנית תקבל שני קבצים דרך ה־CLI: קובץ "database" של מילים רעות וקובץ המכיל את התוכן של ההודעה שנרצה "לנתח". כמו כן, התוכנית תקבל ב־CLI גם ארגומנט שלישי, והוא רף הניקוד (ה־threshold) שחצייתו (**באופן חלש**) מסמנת כי מדובר בהודעת Spam. לאחר בניית אוביקט שמייצג את ה־database, התוכנית תעזר ב־database כדי לקטלג האם ההודעה חשודה כספאם אם לאו. להלן נגדיר בפירוט כיצד יתבצע כל שלב:

5.1 קלט

5.1.1 קובץ ה־database

קובץ ה־database יהיה בפורמט CSV (Comma-separated values)[6]. בקובץ, כל שורה תתאר צירוף רע ואת הניקוד שלו:

תא 1	תא 0
הניקוד הניתן לצירוף	הצירוף הרע

הערות והנחיות:

- על אף שפורמט הקובץ הוא CSV, הסיומת של הקובץ לא חייבת להיות ".csv" ויכולה להיות כל סיומת אחרת, למשל ".txt".
- **לא ניתן** להניח שהקובץ קיים או שהקובץ אינו ריק. אם הקובץ לא קיים – מדובר בשגיאה. אם הקובץ ריק – ניתן להתקדם הלאה (פשוט לא קיימים ביטויים "רעים").
- **ניתן** להניח שמדובר בקובץ CSV תקין. **<u>מנגד</u>, לא ניתן** להניח שיהיו רק שתי עמודות – עליכם לוודא זאת. קובץ שאינו עומד בפורמט המתואר **נחשב קובץ פגום** ועליכם לפעול בהתאם להוראות שיוצגו בהמשך (בפרק של טיפול בשגיאות).
- **לא ניתן** להניח שערכי העמודות תקינים. אם אחד מהערכים אינו תקין – מדובר בקובץ פגום. בפרט:
  - אם לא נשלח אף ערך לתא ה־0 או לתא ה־1 (למשל הוזן ",5") – מדובר בקובץ פגום.
  - אם אחת השורות ריקה (לרבות אם יש בה רק רווחים) – מדובר בקובץ פגום.

[5] רמז (למי שהגדיל וקרא את הערת השוליים ☺): בנוסף לסוג האיטרטור שביקשנו שתממשו, שימו לב איזה פעולות C++ מחייבת כדי לבצע "איטרצית for-each" (כלומר `for (auto it : map)`).

[6] ניתן להכיר את הפורמט ולראות דוגמה כאן: https://en.wikipedia.org/wiki/Comma-separated_values

- ביטוי יכול להכיל יותר ממילה אחת. **ניתן** להניח שבצירוף לא יופיע פסיק (כדי לא לפגוע בתקינות הפורמט). עם זאת, **לא ניתן להניח** שלא יופיעו תווים מיוחדים אחרים, למשל < או >.
- עליכם לוודא שערך הניקוד יהיה חיובי או 0. אם הערך שלילי, מדובר בשגיאה.
- **<u>שימו לב:</u> ניתן להניח** שלא יהיו ביטויים שמוכלים אחד בשני.
  לדוגמה: אם מופיע הביטוי "your bank account details", אזי לא יופיעו בקובץ ביטויים כדוגמת "bank account" או "account details" – כדי למנוע חפיפה. מנגד, כן יכול להופיע הביטוי "forward me your bank account" כיוון שהוא אינו מוכל במלואו בביטוי הראשון (אין ביניהם חפיפה חלקית).

- ניתן להניח שקובץ יסתיים ב־'\n'.
- הנכם **רשאים** להשתמש בספרית filesystem[7] ובספרית tokenizer[8] של boost[9]
- הנכם **רשאים** לעשות שימוש ב־STL (C++ Standard Template Library).

5.1.2 קובץ הקלט (תוכן הודעת הדואר האלקטרוני)

הערך השני שתקבלו דרך ה־CLI יהא נתיב לקובץ הקלט, שמייצג את התוכן של הודעת הדואר האלקטרוני. תוכן זה יופיע כטקסט פשוט (Plain Text) ואותו עליכם לנתח. שימו לב:

- **לא ניתן** להניח שהקובץ קיים או שהקובץ אינו ריק. אם הקובץ לא קיים – מדובר בשגיאה; מנגד, אם הקובץ ריק – וההודעה תקינה "באופן ריק".
- ניתן להניח שקובץ יסתיים ב־'\n'.
- הנכם **רשאים** להשתמש בספרית filesystem[10] ובספרית tokenizer[11] של boost[12]
- הנכם **רשאים** לעשות שימוש ב־STL (C++ Standard Template Library).

5.1.3 רף הניקוד המקסימלי לזיהוי Spam – ה־threshold

לבסוף, הנכם נדרשים לקלוט מה־CLI ערך מספרי **חיובי ממש** שמסמן את הרף לזיהוי הודעת Spam. לא ניתן להניח שקלט זה יהיה תקין באף מובן.

5.2 מימוש ופלט

עתה, על תוכניתכם לנתח את ההודעה באמצעות קריאתה ולבחון האם היא מסווגת כ־Spam אם לאו. הפלט שעל תוכניתכם להדפיס יהיה "SPAM" אם מדובר בספאם, ו־"NOT_SPAM" אם לא מדובר ב־Spam בלבד. בסוף ההדפסה, כמובן, תבוא ירידת שורה. שימו לב:

[7] ראו: https://www.boost.org/doc/libs/1_70_0/libs/filesystem/doc/index.htm
[8] הדוגמה נלקחה מהקישור הבא: https://theboostcpplibraries.com/boost.tokenizer.
[9] תוכלו לקרוא על הספריה המופלאה הזו בתרגיל 2 וכן כאן: https://en.wikipedia.org/wiki/Boost_(C%2B%2B_libraries)
[10] ראו: https://www.boost.org/doc/libs/1_70_0/libs/filesystem/doc/index.htm
[11] הדוגמה נלקחה מהקישור הבא: https://theboostcpplibraries.com/boost.tokenizer.
[12] תוכלו לקרוא על הספריה המופלאה הזו בתרגיל 2 וכן כאן: https://en.wikipedia.org/wiki/Boost_(C%2B%2B_libraries)

- עליכם לבדוק האם הניקוד הכולל של הקובץ חוצה או שווה ל־threshold (כלומר $threshold \leq score$). אם התשובה לכך חיובית – זהו ספאם.
- עליכם לעשות שימוש ב־HashMap שיצרתם לשם מימוש חלק זה של התרגיל. בפרט, כדאי, אך לא חובה, להיעזר ב־iterator שכתבתם.
- כפועל יוצא מהסעיף הקודם, אין לעשות שימוש ב־STL באופן שמדמה מפה.
- **<u>חשוב:</u>** עליכם לבצע את כל ההשוואות בין המילים שבקובץ ה־database ובין הטקסט **באותיות קטנות**. לכן, אם המילה "Facebook" מופיעה (כך ממש) ב־database, היא תילכד בכל המקרים "facebook", "FaceBook", "FACEBOOK", "Facebook".

5.3 רהתמודדות עם שגיאות

עליכם לטפל במקרים בהם לא התקבל קלט תקין. אם מספר הארגומנטים שנשלחו לתוכנה אינו תקין, עליכם להדפיס ל־stderr את הפלט:

```
Usage:  SpamDetector <database path> <message path> <threshold>\n
```

מנגד, אם נתקלתם בקלט שגוי (קובץ שאינו קיים, $threshold \leq 0$ וכו') – עליכם להדפיס:

```
Invalid input\n
```

בשני המקרים "\n" מסמן ירידת שורה (כפי שנהוג להדפיסה ב־C++). לאחר הדפסת הפלט, בשני המקרים עליכם לסגור באופן מיידי את התוכנית עם קוד סיום EXIT_FAILURE.
<u>שימו לב:</u> עליכם להתמודד עם כל חריגה שתוכניתכם עלולה לייצר, בין אם בעקבות חריגה שאתם זרקתם באופן מפורש, ובין אם בעקבות חריגה ש־C++ עלולה לזרוק. ההודעה שיש להדפיס תהיה תלויה בסיבה לחריגה.

6 דוגמה

נניח ש־database.csv מכיל את התוכן הבא:

```
financial help,5
billionaires,10
millionaires,10
lucky,10
random,5
your back account details,20
Facebook username and password,50
```

וכן נניח ש־message.txt מכיל את התוכן הבא:

```
Winner found!
Hello there, I'm Mark Zuckerberg, the founder and CEO of the social-networking
website Facebook, as well as one of the world's youngest billionaires.
I decided to secretly give $1,500,000.00 selected individuals worldwide
at random.  You should count yourself as the lucky individual.  Your
email address was chosen online while searching at random.  Kindly
email me back with your bank account details and Facebook username
and password so that I can verify your identity and send you the money.
```

```
Kind regards,
Mark Zuckerberg
```

אם כן, נשים לב שסריקת הקובץ תוביל לתוצאה הבאה:

ניקוד כולל (ניקוד כפול הופעות)	כמות הופעות	צירוף רע
0	0	financial help
10	1	billionaires
0	0	millionaires
10	2	random
10	1	lucky
20	1	your bank account details
50	1	Facebook username and password

בסך הכול, קיבלנו שניקוד ההודעה הוא 100. לכן נקבל בהרצת SpamDetector:

```
$ SpamDetector database.csv message.txt 50
SPAM
$ SpamDetector database.csv message.txt 150
NOT_SPAM
```

כאשר שורה שנפתחת ב־$ מסמנת את הפקודה שהוקלדה.

7 נהלי הגשה

- קראו בקפידה את הוראות תרגיל זה ואת ההנחיות להגשת תרגילים שבאתר הקורס.
- זכרו שהחל מתרגיל זה עליכם לקמפל את התוכנית כנגד מהדר לשפת C++ בתקן שנקבע בקורס. כמו כן, זכרו שעליכם **לתעדף** פונקציות ותכונות של C++ על פני אלו של C. למשל, נעדיף להשתמש ב־new ו־delete על פני malloc ו־free, וכן נעדיף להשתמש ב־std::string מאשר ב־*char.
- **נזכיר:** כאמור בהנחיות הכלליות להגשת תרגילים ־ הקצאת זיכרון דינמית מחייבת את שחרור הזיכרון, למעט במקרים בהם ישנה שגיאה המחייבת סגירת התוכנית באופן מיידי עם קוד שגיאה (כלומר קוד יציאה השונה מ־0). תוכלו להיעזר בתוכנה valgrind כדי לחפש דליפות זיכרון בתוכנית שכתבתם.
- פתרון בית הספר זמין בנתיב

  ```
  ~labcc/www/cpp_ex3/SpamDetector
  ```

- עליכם ליצור קובץ tar הכולל את הקבצים HashMap.hpp, SpamDetector.c בלבד. ניתן ליצור קובץ tar כדרוש על ידי הפקודה:

  ```
  $ tar -cvf cpp_ex3.tar HashMap.hpp SpamDetector.cpp
  ```

שימו לב: קבצי קוד המקור שתכתבו נדרשים להתקמפל כהלכה עם std++14, כנדרש בהוראות להגשת תרגילים שפורסמו באתר הקורס.

- אנא וודאו כי התרגיל שלכם עובר את ה־Pre-submission Script **ללא שגיאות או אזהרות**. קובץ ה־Pre-submission Script זמין בנתיב.

```
~labcc/www/cpp_ex3/presubmission
```

בהצלחה!!