האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות בשפת C ו־++C (67320)
C (קורס קיץ) – תרגיל 6

תאריך הגשה: 16 לספטמבר, 2020.

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

1 רקע

בתרגיל זה תדרשו לעשות שימוש בכלים שרכשתם במהלך הקורס בכדי לממש ספריה, המורכבת מקובץ header יחיד, שתכיל מבנה נתונים **גנרי** מסוג מפת גיבוב (HashMap) (יש לכם רעיון מה תהיה סיומת הקובץ?). מבנה זה יתנהג באופן זהה למבני נתונים אחרים הזמינים ב־STL (כמובן, בפרט, הוא יהיה תואם ל־`std::unordered_map`), ויעשה שימוש מאחורי הקלעים בנושאים רבים שהתעסקתם בהם, כגון אופרטורים, הקצאות זיכרון דינמיות, מערכים, templates ועבודה עם STL.
קראו היטב את ההוראות המופיעות לאורך המסמך, בפרט לאלו הנוגעות לטיפוסים מ־STL בהם מותר (או, אסור), להשתמש בתרגיל זה, כמו גם לנושאי יעילות.

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 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 עליו) – 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 $\frac{3}{4}$.[1]
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 $capacity \geq 1$.**

[1] בחירה זו מקובלת ונעשה בה מקום במימושים סטדרטים. ראו למשל, ב־Java: https://stackoverflow.com/a/435000.

3.3 פונקצית הגיבוב

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

$$h(x) = x \, mod \, capacity, \qquad capacity \in \mathbb{N} \;\; S.t. \;\; 2 \leq capacity$$

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++):

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

לאור החסרונות שבגישה זו, ולאור העובדה שנרצה להשתמש בערכי size_t, נציע את החלופה הבאה: 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:

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

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 המחלקה הגנרית HashMap

בתרגיל זה 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	זמן ריצה
פעולות מחזור החיים של המופע			
בנאי ברירת מחדל	בנאי שמאתחל HashMap ריק עם capacity ברירת מחדל של 16.		
בנאי 1	בנאי המקבל שני איטרטורים, אחד שמכיל ערכי KeyT ואחד שמכיל ערכי ValueT ושומר את הערכים במפה לפי הסדר. החתימה המלאה תופיע בהמשך.	וודאו שהוקטורים באותו הגודל. המיפוי יהיה $\forall 0 \leq i < n \;\; keys[i] \mapsto values[i]$. חריגה תיזרק אם האיטרטורים אינם שווי גודל.	$O(n)$
מתודות			
size	הפעולה מחזירה את כמות איברי המפה.	המספר שמחוזר מטיפוס size_t.	$O(1)$
capacity	פעולה המחזירה את קיבולת המפה.	המספר שמחוזר מטיפוס size_t.	$O(1)$
empty	פעולה הבודקת האם המפה ריקה.	הפעולה תחזיר bool.	$O(1)$
insert	פעולה המקבלת מפתח וערך, ושומרת את המיפוי שהתקבל.	הפעולה מחזירה bool.	$O(n')$ *
contains_key	הפעולה מקבלת מפתח ובודקת האם הוא קיים במפה.	הפעולה מחזירה bool.	$O(n')$ *
at	פעולה מקבלת key ומחזירה את ה־$value$ המשוייך אליו.	הפעולה תזרוק חריגה במקרה שה־key לא נמצא.	$O(n')$ *
erase	הפעולה מקבלת מפתח ומסירה את הערך המשוייך לו מהמפה.	הפעולה תחזיר אמת אם הערך הוסר בהצלחה.	$O(n')$ *
load_factor	פעולה המחזירה את גורם העומס.	המספר שמחוזר מטיפוס double.	$O(1)$
bucket_size	תקבל מפתח ותחזיר את גודל הסל.	הפעולה תחזיר size_t. אם המפתח לא נמצא – יש לזרוק חריגה.	$O(1)$
bucket_index	תקבל מפתח ותחזיר אינדקס הסל.	הפעולה תחזיר size_t. אם המפתח לא נמצא – יש לזרוק חריגה.	$O(n')$ *
clear	פעולה המסירה את כל איברי המפה.	ה־$capacity$ לא משתנה.	$O(n')$ *
iterator	**מימוש** מחלקת iterator וכל הפעולות הנדרשות ל־iterator (**לרבות typedefs**), בהתאם לשמות הסטנדרטים של C++.	עליכם לממש const forward iterator (כלומר, אין צורך בחלופה שאינה const). 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.**	$O(n')$ *
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?).
- החתימה לבנאי 1 היא:

```
template<typename KeysInputIterator,
         typename ValuesInputIterator>
HashMap(const KeysInputIterator keysBegin,
const KeysInputIterator keysEnd,
const ValuesInputIterator valuesBegin,
const ValuesInputIterator valuesEnd);
```

- 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.
- כאמור לעיל, ערך ברירת המחדל של קיבולת המפה יהיה 16. נזכיר שערך זה יכול רק לגדול (ולכן קיבלנו שבכל מצב, במימוש שלנו, $16 \leq capacity$).
- **הנכם מחוייבים לשמור את הסלים בתור מערך שמוקצה דינמית.**[3] **מימוש באמצעות כל שיטה אחרת ינוקד באופן אוטומטי בציון 0.**
  במילים אחרות, נבקש שתבחינו בין מבנה הנתונים שמחזיק את כל הסלים, לבין מבנה הנתונים שמייצג סל אחד ספציפי. כל סל ספציפי תוכלו לייצג באמצעות מבנה נתונים שבחרתם, כראות עיניכם (ובשימת לב לדרישות זמן הריצה), מ־STL (או לחלופין, מימשתם, אם יש סיבה מיוחדת). מנגד, מבנה הנתונים שמחזיק את **כל** הסלים, חייב להיות מערך דינמי.
- **<u>דרישות זמן ריצה:</u>** בסימוני הסיבוכיות שלעיל, $n \in \mathbb{N}$ מסמן את כמות האיברים הכוללת, שבכל הסלים, שנשמרים במפה, בעוד ש־$n' \in \mathbb{N}$ מסמן את כמות האיברים שנשמרו **בסל אחד ספציפי.** נקבל, אפוא, כי $0 \leq n' \leq n$. כך, למעשה, בכל פעולה שבה מופיעה כוכבית ליד הסיבוכיות, הכוונה היא שהפעולה תפעל **בסיבוכיות לינארית ביחס לגודל הסל.** לדוגמה, הפעולות insert, at, contains, erase (וכמובן $operator[]$) נדרשות לפעול בסיבוכיות לינארית ביחס לסל שבו האיבר נמצא – כלומר ב־$O(n')$ כאשר n' הוא גודל הסל . מנגד, בנאי 1, שם אין כוכבית, צריך לפעול בסיבוכיות לינארית סטנדרטית (כך ש־n מייצג את כל האיברים, ולא רק את האיברים שבסל אחד ספציפי).
- **למותר לציין, אבל יצויין בכל זאת, שאין להשתמש במחלקות של STL באופן שייתר את פתרון התרגיל. למשל, אין לעשות שימוש ב־**std::unordered_map **במקום לממש מפת גיבוב באופן עצמאי. מנגד, אתם בהחלט רשאים (ומצופה מכם) לעשות שימוש ב־STL** (כל עוד השימוש תואם להוראות של התרגיל).

[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 <u>שלא</u> להקצות כמות זיכרון גדולה יותר**, אם ההספריה "סבורה" שכך נכול לעשות. מכאן באה הדרישה האמורה.

- **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.

בפרט, אם מצאתם לנכון שמתודה מסויימת דורשת מימוש const ומימוש non-const נפרדים, אתם בהחלט רשאים לעשות זאת. הטבלה שלעיל מציינת רק מהם שמות המתודות שבהם עליכם לתמוך. השאלה איזה וריאציות const/reference יש לממש נתונה לשיקול דעתכם בהתאם לנלמד בהרצאות ובתרגולים.

נוסיף ונזכיר כי בתרגיל זה עליכם לתת את הדעת גם לנושא של חריגות, ולכן שימו לב לשימוש נכון ב־noexcept כשיש צורך (גם סימון זה לא מופיע בטבלה, ועליכם להחיל את הכללים שנלמדו באשר לשימוש בו).

- בהמשך לאמור בסעיף הקודם, חשבו האם יש צורך בבנאי העתקה, Destructor ו/או =operator. בעוד לא ציינו אותם בטבלה, אם אתם מוצאים לנכון, **אתם בהחלט רשאים לממש אותם.**

- השמות שבחרנו ל־API זה אינם עולים בקנה אחד עם ה־coding style של הקורס, אך הם נבחרו כך כדי שיתאמו ל־`std::unordered_map`. יש להתעלם מאזהרות ב־presubmission בנושא זה. עם זאת, כל משתנה פרטי, מתודה פרטית וכו' **מחוייבים** לעמוד ב־coding style. **לא נקבל ערעורים על נושא coding style** (וחבל על הזמן של כולנו...).

- ה־API לו אתם נדרשים זהה מבחינת הפרמטרים וערכי החזרה לזה של `std::unordered_map`. לכן המלצתנו החמה היא כי תעיינו ותכירו היטב את ממשק זה, שכן הוא ישרת אתכם בכל התלבטות הנוגעת למימוש.[4] כפועל יוצא, במקרה שאינכם בטוחים איך המחלקה צריכה להתנהג – תוכלו לעיין ב־API כאמור. כך, תוכלו להיעזר ב־`std::unordered_map` גם כדי לבדוק כי המימוש שלכם עובד כראוי.

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.

- `bucket_size` **ו־**`bucket_index` **:** 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).[5]

[4] ראו: https://en.cppreference.com/w/cpp/container/unordered_map

[5] 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`.

– **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'`.

- **מימוש iterator:** שימו לב כי מימוש ה־iterator יחייב אתכם לממש מחלקת iterator, 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 דוגמה – Characters Encoder

לתרגל זה לא יפורסם פתרון בית ספר. עם זאת, כדי לסייע לכם, יצרנו עבורכם תוכנית לדוגמה, העושה שימוש בכמה מהתכונות הבסיסיות של מפת הגיבוב. כך, אם זו מומשה נכון, תוכלו לקמפל ולהריץ בהצלחה את התוכנית. תוכנית זו, השמורה תחת הקובץ Encoder.cpp, מצורפת כחלק מקובצי התרגיל. תוכלו לעשות בה שינויים כרצונכם, ואין להגישה עם המבחן.
מטרתה של התוכנית Encoder הוא לקבל מ־stdin מחרוזת כקלט, ולקודד אותה בהתאם ל־"מפת הקידוד" (encoding map). במילים אחרות, בהינתן מיפוי $src[i] \mapsto dst[i]$ מטרתה של התוכנית להמיר כל תו, $src[i]$, שהופיע במחרוזת הקלט, לתו המתאים לו – $dst[i]$.
נראה דוגמה: נניח שיש לנו את טבלת המיפוי הבאה:

src	dst
a	x
b	y
x	b
y	x

עתה, נניח שהתבקשנו לקודד את המחרוזת "aabbxyxb". אז, Encoder יעבור תו תו ויבדוק האם הוא קיים במפה. אם אינו קיים במפה, התוכנית תצא בקוד שגיאה. אחרת, התוכנית תמיר את $src[i] \mapsto dst[i]$. כך, נקבל את המחרוזת "$xxyybxby$".
להלן מספר דגשים:

- התוכנית מבקשת לקבל כארגומנט CLI ערך יחיד, שהוא הנתיב לקובץ ה־encoding. כל שורה בקובץ מכילה 2 תווים (char) המופרדים ברווח. התו השמאלי הוא תו "המקור" והתו הימני הוא תו "היעד".

- התוכנית מבצעת בדיקות קלט בסיסיות בלבד. תוכנית זו אינה מתיימרת להיות פתרון מלא ומקיף, אלא להציג שימוש בסיסי ב־HashMap שיצרתם.

- תוכלו ליצור קידודים נוספים שונים ולראות האם התוכנית מוציאה פלט זהה לזה שציפיתם לו או לא.
אנו ממליצים כי תעיינו בקפידה בתוכנית, הכוללת הערות המסבירות את הנעשה שלב שלב. תוכנית זו תוכל לסייע לכם בהבנת המשימה.

- הנכם רשאים לערוך את קובץ זה כראות עיניכם ולעשות בו שימוש על מנת לבחון את תרגילכם.

6 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.
- עליכם ליצור קובץ tar הכולל את הקובץ HashMap.hpp בלבד. ניתן ליצור קובץ tar כדרוש על ידי הפקודה:

```
$ tar -cvf ex6.tar HashMap.hpp
```

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/ex6/presubmission
```

בהצלחה!!