האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

Programming Workshop in C & C++ - 67312

תרגיל בית 4 – שפת C
סמסטר חורף, 2020/21

מועד פרסום התרגיל: תאריך – 03/12/20202
מועד הגשת התרגיל: תאריך – 17/12/2020 בשעה - 23:25

1 רקע

בתרגיל זה תדרשו לעשות שימוש בכלים שרכשתם במהלך הקורס בכדי לממש ספריה שתכיל מבנה נתונים גנרי מסוג מפת גיבוב. מבנה זה יעשה שימוש מאחורי הקלעים במבנה נתונים אחר אותו תממשו (וקטור בגודל דינמי). קראו היטב את ההוראות המופיעות לאורך המסמך ואת ההנחיות. בנוסף, ודאו כי אתם מבינים היטב את הAPIs אותם אתם נדרשים לממש (יופיעו בקבצים שיסופקו לכם עם התרגיל).

2 הגדרות

נגדיר מספר הגדרות הנוגעות לטבלאות גיבוב (לקחו מקורס מבני נתונים):

1. <u>פונקציית גיבוב (hash function)</u> – פונקציה הממפה מידע מגודל כלשהו ("מפתחות") למידע אחר כלשהו, בגודל סופי. לשם פשטות, ניתן להניח שמדובר בפונקציה מהצורה: $h: U \to \{0, \dots, m-1\}$ כאשר U היא קבוצת איברים כלשהי (למשל: מחרוזות, מספרים וכו') ו- $\{0, \dots, m-1\}$ היא קבוצה סופית של תוצאות אותן ניתן לקבל מהפונקציה h.

בנוסף, נרצה שכל פונקציית גיבוב h תקיים:

- h תהיה קלה לחישוב.
- ש-h תמפה כמה שפחות איברים לאותו התא (כדי למנוע התנגשויות).

2. <u>מפת גיבוב (hashmap)</u> – מבנה נתונים המכיל מיפוי של מפתחות לערכים. המפתחות והערכים יכולים להיות מספרים, מחרוזות או כל טיפוס נתונים נתמך אחר. היתרון של מבנה נתונים זה הוא שבהינתן פונקציית גיבוב "טובה" (כזו שעונה על התנאים לעיל), פעולות ההוספה, החיפוש וההסרה שלו מבוצעות בסיבוכיות זמן ריצה ממוצעת של $\theta(1)$.

3 מימוש HashMap

לאחר הצגת ההגדרות הקודמות הנדרשות לפתרון התרגיל, נציג מספר נושאים נוספים הנוגעים לפונקציות, טבלאות ומפות גיבוב, להם תדרשו במימוש התרגיל:

3.1 גורם עומס (Load Factor)

לבד מגודל הקלט בפועל, הביצועים של מפת הגיבוב מושפעים משני פרמטרים: גורם עומס עליון וגורם עומס תחתון (upper load factor & lower load factor). גורם העומס מוגדר באופן הבא:

$$\text{LoadFactor} = \frac{M}{\text{capacity}} \quad , \quad \text{capacity} > 0 \ \wedge M \geq 0$$

כאשר M מייצג את כמות האיברים שמבנה הנתונים מכיל ברגע נתון (בפועל), ו-capacity מייצג את כמות הנתונים המקסימלית שניתן לשמור במבנה הנתונים (הקיבולת). גורם העומס העליון והתחתון מגדירים כמה ריק או מלא

נסכים שמבנה הנתונים יהיה. כלומר, אם נחצה רף מסוים, נגדיל או נקטין את מבנה הנתונים בהתאם, כך שמחד לא נדרוש הרבה זיכרון מיותר ומאידך נוכל להכיל את כמות האיברים שנרצה.

3.2 גודל הטבלה

בתרגיל זה, גודל מבנה הנתונים (ה-capacity) יהיה חזקה של 2 כאשר הגודל ההתחלתי של הטבלה יהיה 16 (מוגדר כקבוע). שימו לב – בהכרח מתקיים $\text{capacity} \geq 1$.

3.3 פונקציית הגיבוב

כאמור לעיל, עלינו לבחור פונקציית גיבוב "טובה" כדי להגיע למבנה נתונים יעיל. אנו נשתמש בפונקציית גיבוב פשוטה מאוד:

$$\mathrm{h(x) = x \bmod capacity} \ , \qquad \mathrm{capacity} \in \mathbb{N} \ \mathrm{s.t\ capacity} \geq 2$$

כאשר הקיבולת זהה לאופן בו הגדרנו אותה קודם לכן. x הוא ייצוג מספרי של הערך שנרצה לשמור בטבלה. כדי להמיר מחרוזות, מספרים, וכיוצא בזה למספר שלם, נספק לכם פונקציות מתאימות (ראו API למטה). נשים לב שאופרטור מודולו (%) לא עובד כמו האחד המתמטי. למשל, $(-3)\%7 = -3$ אם נחשב בעזרת קוד, אך המודולו המתמטי נותן ערך 4. מסיבה זו ומסיבות נוספות, נחשב את המודולו באופן הבא: $\mathbf{v \ mod \ capacity} = \mathbf{v} \ \& \ (\mathbf{capacity} - \mathbf{1})$ (שימו לב שזה אופרטור and לוגי!).
פתרון זה עדיף, כיוון שאופרטורים לוגיים מהירים יותר מאריתמטיים.

3.4 אלגוריתמי מיפוי

בהמשך לאמור, ניתן לשער שהפונקציה שהגדרנו קודם לכן תגיע במהרה להתנגשויות. כלומר, אנו עלולים להיתקל בשני ערכים $\mathrm{x, y}$ שונים זה מזה אך יתקיים $\mathrm{h(x) = h(y)}$, וזה שקול כמובן להתנגשות. ישנן שתי שיטות נפוצות לפתרון התנגשויות:

1. <u>מיפוי פתוח (Open Hashing)</u> – שיטה המאפשרת לשמור יותר מערך אחד בכל תא. התאים, הנקראים גם "סלים" (או buckets) ובנויים מטיפוס נתונים אחר (למשל מערך דינמי או רשימה מקושרת). כך, גם במקרה של התנגשות, כל שקורה הוא שהאיבר המתנגש נוסף לרשימה המקושרת.
2. <u>מיפוי סגור (Close Hashing)</u> – שיטה לפיה כל תא יכול להכיל רק איבר אחד. במקרים אלו, עלינו למצוא דרך אחרת להתמודד עם התנגשויות ולמפות איברים.

בתרגיל זה, נשתמש בשיטת מיפוי פתוח.

4 הספרייה HashMap

כאמור, בתרגיל זה נממש את הספרייה HashMap (מפת גיבוב). הספרייה תממש מפת גיבוב המבוססת שיטת מיפוי פתוח. על מנת לממש שיטה זו, כל ערך במפת הגיבוב צריך להיות מסוגל להחזיק מספר ערכים שונים שמופו לאותו מפתח. לכן, בחלק הראשון של המימוש, נממש מבנה נתונים נוסף שיקרא Vector (סוג של מערך דינמי) ובו נשתמש על מנת לאכסן מספר ערכים שונים עבור מפתח כלשהו. בצורה סכמתית ניתן לחשוב על זה באופן הבא: כל מפתח במפת הגיבוב ממופה לווקטור של ערכים
$key \rightarrow Vector(\{val1, val2, ..\})$.

4.1 הספרייה Vector

לחלק זה מצורף הקובץ Vector.h הכולל את חתימות הפונקציות אותן תדרשו לממש בחלק זה ואת הקבועים שיסייעו לכם כמו גודל התחלתי ופרמטר הגדילה.
שימו לב – מימוש חלק זה קריטי להצלחת מימוש התרגיל כולו. בנוסף, הAPI של חלק זה נבדק בנפרד וגם יחד עם API של הHashMap.
בנוסף, הAPI הנ"ל הינו דרישה אך כמובן שניתן להוסיף פונקציות עזר בראות עיניכם ולפי צרככם (בקובץ הc).

4.2 הספרייה HashMap

לחלק זה כמו לקודם מצורף קובץ HashMap.h הכולל את חתימות הפונקציות אותן תדרשו לממש בחלק זה.
חלק זה הינו החלק העיקרי ועליו יעשו מרבית הבדיקות.
שימו לב:

1. הספרייה חייבת להיות **גנרית** (ומכאן, גם הוקטור חייב להיות גנרי). ספרייה שתיכתב באופן שאינו גנרי תפסיד נקודות רבות ללא אפשרות לערעור, שכן לא תעברו טסטים (אין ביכולתכם לדעת מה נבחר להכניס לPairs).
2. הנכם חייבים להשתמש בספריית הVector מסעיף 4.1. בנוסף, בכל סיטואציה בה תוכלו להשתמש באחת

הפונקציות מהAPI של הווקטור ותבחרו שלא לעשות כך, תוכלו להפסיד נקודות. אחת ממטרות הAPI היא לאפשר כתיבת קוד נקי וברור.

4.3 הממשק Pair

חלק זה נתון עבורכם במודל ועליכם להשתמש בו בתרגיל. האובייקט התכנותי אותו מייצג חלק זה הינו זוג מהצורה { key: value }.
זוג זה ייצג את האלמנט הגנרי שתכניסו ל-HashMap. הממשק כולל מפתח וערך גנריים. כלומר, ניתן לייצר באמצעות ממשק זה כל זוג אפשרי.
דוגמא: בהינתן struct Person ניתן יהיה לייצר זוג מהצורה { id (size_t *) : person (Person *) }.
בנוסף לקבצים Pair.c, Pair.h בהם תשתמשו בקוד שלכם, נתון לכם קובץ עזר הכולל דוגמא לממשק עבור Pair מסוג ספציפי הכולל מימוש של זוג ואת כל הפונקציות הנלוות הנדרשות.

4.4 הקובץ Hash

בקובץ הנ"ל (Hash.h) תקבלו מימוש של מספר פונקציות גיבוב (פשוטות מאוד) לדוגמא בהן תוכלו להשתמש לצורך בדיקות.

5 הרצת התוכנית

את התוכנית שלכם נקמפל יחד עם קובץ main.c חיצוני שאינכם ראיתם. הקובץ ישתמש בפונקציות שונות מתוך הAPI אותו כתבתם.
קובץ לדוגמא מופיע בנספח ב' של מסמך זה.

6 הגשת התרגיל

עליכם להגיש קובץ tar בשם ex4.tar והוא יכלול את הקבצים הבאים והם בלבד:
Vector.c
HashMap.c

לא להגיש את קבצי הh (לא נהיה סלחניים לגבי זה).

8 הערות ודגשים

- אנא וודאו כי התרגיל שלכם עובר את סקריפט ה-pre-submission ללא שגיאות או אזהרות. הסקריפט זמין בנתיב:

```
~labcc/presubmit/ex4/run <path_to_submission>
```

- עליכם לבדוק כי התוכנית מתקמפלת ללא הערות על מחשבי בית ספר לפי הפקודה הבאה:

```
gcc -Wall -Wextra -Wvla -Werror -g -lm -std=c99 <code_files> <main_file> -o hash_map_prog
```

- כחלק מהבדיקה האוטומטית תבדקו על סגנון כתיבת קוד. אנא ודאו כי הנכם מבינים את דרישות ה coding style של הקורס במלואן.

- הקפידו על בדיקות של דליפות זיכרון ושאר השגיאות אותן בודק ה-**valgrind**. המשקל אשר יינתן לבדיקות אלו יהיה רב ולכן וודאו זאת היטב.
למען הסר ספק, הvalgrind אמור לרוץ באופן תקין מלא-מלא, כלומר שום שגיאה או הערה מכל סוג שהוא.
אם מופיעה הערה כלשהי, גם אם אינה קשורה לדליפת זיכרון באופן ישיר, היא נחשבת כשגיאה ותהיה על כך הורדת נקודות.

- תיעוד – הקפידו על תיעוד הולם של הקוד אותו אתם כותבים.

- נראות הקוד – הקפידו על כל הדגשים התכנותיים – אורך פונקציות, שמות משתנים ושמות פונקציות אינפורמטיביים וכו'. תהיה בדיקה ידנית ואנחנו נתייחס לדגשים אלה בקפדנות.

בהצלחה!

“I have yet to see a language that comes even close to C”

(Linus Torvalds)

נספח א' – קובץ Main לדוגמא

```
// Includes //

void *ElemCpy(const void *elem) {
  int *a = malloc(sizeof(int));
  *a = *((int *) elem);
  return a;
}

int ElemCmp(const void *elem_1, const void *elem_2) {
  return *((const int *) elem_1) == *((const int *) elem_2);
}

void ElemFree(void **elem) {
  free(*elem);
}

/////

int main() {
  // Insert elements to vector.
  Vector *vector = VectorAlloc(ElemCpy, ElemCmp, ElemFree);
  for (int k_i = 0; k_i < 8; ++k_i) {
    VectorPushBack(vector, &k_i);
  }
  VectorFree(&vector);

  // Create Pairs.
  Pair *pairs[8];
  for (int k_i = 0; k_i < 8; ++k_i) {
    char key = (char) (((k_i) % 75) + 48);
    int value = k_i;
    pairs[k_i] = PairAlloc(&key, &value, CharKeyCpy, IntValueCpy,
CharKeyCmp,
                           IntValueCmp, CharKeyFree, IntValueFree);
  }

  // Create hash-map and inserts elements into it.
  // Uses the Char-Int pairs you received.
  HashMap *hash_map = HashMapAlloc(HashChar, PairCharIntCpy,
PairCharIntCmp, PairCharIntFree);
  for (int kI = 0; kI < 8; ++kI) {
    HashMapInsert(hash_map, pairs[kI]);
  }

  // Free the pairs.
  for (int k_i = 0; k_i < 8; ++k_i) {
    PairFree(&pairs[k_i]);
  }

  // Free the hash-map.
  HashMapFree(&hash_map);
  return 0;
}
```