האוניברסיטה העברית בירושלים
בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות C ו-++C - תרגיל מסכם בשפת C (חלק א')

מועד הגשה (חלק א'): יום ד' 17 לאוגוסט ב-22:00

חשוב: חלק א' של תרגיל 3 (מסמך זה מהווה את חלק א', חלק ב' יפורסם בשבוע הבא) צריך לעבור פרה-סאבמיט בלבד והוא מהווה 25% מהציון הסופי של תרגיל 3

נושאי התרגיל: #מצביעים #מערכים_דינמיים #קריאה_מקבצים #מחולל_טקסט #ניהול_זיכרון

1 רקע

עיבוד שפה טבעית - NLP - Natural Language Processing

אחד מהתחומים החשובים ביותר כיום במדעי המחשב הוא התחום של עיבוד שפה טבעיתNLP . הוא תחום רחב המנסה לגשר בין שפה טבעית (שבה אנו משתמשים ביום יום) לבין המחשב. דוגמא לאחת מבין המשימות הרבות של NLP הוא לגרום למחשב "להבין" דיבור של אדם (לדוגמת Alexa, Siri, Google assistant וכד').

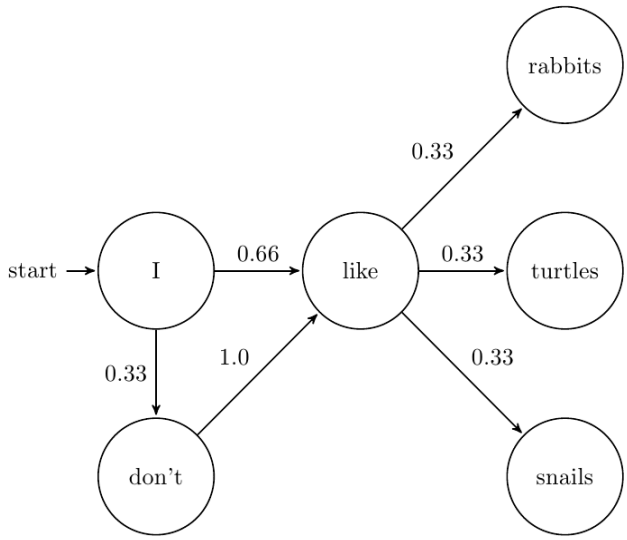
בתרגיל זה נתמקד בנושא נוסף של NLP, שימוש בתוכנת מחשב על מנת לייצר ציוצים (tweets) חדשים בהסתמך על מאגר נתונים קיים. נבצע זאת בעזרת שרשראות מרקוב (Markov chains).

שרשרת מרקוב

שרשרת מרקוב היא מודל הסתברותי המשמש לתיאור התפתחות של תהליכים כמעבר בין סדרה של מצבים. הצמתים (עיגולים) מייצגים מצבים, והקשתות המכוונות (חיצים) מייצגות את ההסתברויות") סיכויים(" לעבור ממצב למצב.

בתרגיל זה, הצמתים מייצגים את המילים השונות, והחיצים הם ההסתברות לעבור ממילה למילה.
למשל בדוגמא להלן:

- מהמילה I ישנו סיכוי של שליש לעבור למילה don't, ושני שליש לעבור למילה like.
- מהמילה don't יש סיכוי של 1 (כלומר מתקיים תמיד) לעבור למילה like.
- מהמילה like יש סיכוי של שליש לעבור לכל אחת מהמילים rabbits, turtles ו-snails.



סכום הסיכויים (החיצים) היוצאים מאותה מילה, תמיד יהיה 1. למשל מהמילה I יוצאים שליש ושני שליש שסכומם 1.

הפונקציות rand() ו-srand()

לרוב, המחשב לא יכול לייצר ערך אקראי אמיתי[1], ולכן מתכנתים נעזרים בפונקציות מחוללות מספרים פסידו-אקראיים **pseudo-random generators**. פונקציות אלו מאותחלות באמצעות ערך התחלתי (seed). לאחר מכן, הן מבצעות סדרת פעולות מתמטיות על ערך ה-seed, וכך מחשבות סדרה של מספרים פסידו-אקראיים, כלומר אקראיים למראית עין בלבד.

ממשק:

- אנו נשתמש ב-**srand(seed)** שמקבלת כפרמטר את הseed ומגדירה אותו עבור הריצה הנוכחית.
- אנו נשתמש ב-**rand()** על מנת לייצר את סדרת הערכים הפסידו-אקראיים. בכל קריאה לפונקציה rand(), יוחזר מספר פסידו-אקראי שלם בין 0 (כולל) ל-RAND_MAX (לא כולל). RAND_MAX הוא קבוע המוגדר מראש בספרייה stdlib.h.

בחירת seed:

1. במקרה בו לא נבחר seed, הקומפיילר מאתחל ערך זה ל-1, ולכן אם נריץ את התוכנית מספר פעמים ברצף, בכל ריצה נקבל תמיד את אותם ערכים פסידו-אקראיים.
2. דרך נפוצה לעקוף בעיה זו היא לבחור seed בעזרת השעה בא החלה התוכנית לפעול.

[1] ניתן לייצר מספרים אקראיים אמיתיים המבוססים על רעש סביבתי של התקנים שונים של המחשב. הנושא הוא מעבר לחומר הקורס, להעשרה: https://en.wikipedia.org/wiki/Hardware_random_number_generator

3. במקרה שלנו נקבל את ערך ה-seed בתור פרמטר שהתוכנית תקבל בתחילת הריצה. ניתן להניח שערך ה-**seed** שתקבלו הוא חיובי ושלם (unsigned int).

2 אופן פעולת התוכנית

התוכנית שנכתוב תעבוד באופן הבא:

1. קלט: נקבל Text corpus, מאגר טקסט גדול המכיל משפטים (ציוצים) רבים.
2. למידה: נקרא את ה-corpus ונשמור את המילים הנתונות בו למבנה נתונים. עבור כל מילה, נשמור את התדירות (frequency) בה מילים אחרות מופיעות לאחריה ב-corpus (למשל **are** מופיעה 100 פעם אחרי **you**).
3. פלט: נשתמש במבנה הנתונים על מנת לייצר ציוצים באופן הסתברותי:
   a. נגריל מילה ראשונה לציוץ מתוך מבנה הנתונים (למשל **You**).
   b. נשתמש ברשומה (entry) של אותה מילה במבנה נתונים על מנת להגריל את המילה הבאה בציוץ. הסיכוי של מילה להיבחר בהגרלה פרופורציונאלי לתדירות בה היא מופיעה אחרי המילה הקודמת ב-corpus (למשל, סביר שהמילה **are** תיבחר בסיכוי גבוה יותר מהמילה **am** לאחר המילה **You**).
   c. נמשיך לייצר את המילים הבאות באותה דרך, עד שנגיע למילה המסתיימת בנקודה (סוף משפט). (למשל **You->are->absolutely->awesome.**).
   d. כל ציוץ שייצרנו באופן הנ"ל מהווה סדרת מרקוב, המילים בציוץ הן המצבים שנבחרו מתוך השרשרת.

3 קבצים

סיפקנו לכם 3 קבצי קוד וקובץ קלט:

- markov_chain.h שמכיל את השלד של הפונקציות שצריכים להגיש.
- linked_list.c ו- linked_list.h - רשימה מקושרת לשימושכם.
- justdoit_tweets.txt - דוגמא לקובץ קלט. הקובץ מכיל ~4400 ציוצים ותוכלו לבדוק את תוכנתכם באמצעותו. השתדלנו לנקות את המאגר מתוכן פוגעני. אם החמצנו משהו, אנו מתנצלים על כך מראש.

אתם צריכים להגיש:

- markov_chain.h מעודכן עם הסטראקטים שתכתבו.

- markov_chain.c עם מימוש של הפונקציות המוכרזות ב-markov_chain.h.
- tweets_generator.c שמכיל את ה-main ועוד פונקציות שנפרט עליהם בהמשך.

4 קלט

פרמטרים

ממשו את פונקציית ה-main בקובץ tweets_generator.c שמקבלת את הארגומנטים הבאים (בסדר הבא):

1. ערך **seed** - מספר שיינתן לפונקציית ה-srand() פעם אחת בתחילת הריצה. ניתן להניח כי הוא מספר שלם חיובי. (unsigned int).
2. כמות הציוצים שנייצר- ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0 (int).
3. נתיב (path) לקובץ ה- **Text corpus** - אין להניח כי הנתיב שניתן תקין. במקרה בו הקובץ לא קיים או שלתוכנית אין הרשאות גישה אליו, יש להדפיס הודעת שגיאה מתאימה ל- stdout המתחילה ב-
"Error:"
ולצאת מהתוכנית עם **EXIT_FAILURE**.
4. כמות המילים שיש לקרוא מהקובץ - במקרה בו לא התקבל פרמטר רביעי יש לקרוא את הקובץ כולו. ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0 (int). אין להניח כי כמות המילים בקובץ הטקסט גדולה או שווה לפרמטר הנתון, במקרה זה יש לקרוא את הקובץ כולו ולהמשיך בתוכנית ללא כל הודעה.

באם כמות הפרמטרים שהתקבלו לא תואמת את הדרישות (3 או 4 פרמטרים), יש להדפיס הודעה ל-stdout המפרטת בקצרה את הפרמטרים הנדרשים ומתחילה ב-
"Usage:"
ולצאת מהתוכנית עם **EXIT_FAILURE**.

דוגמא לקריאה תקנית לתוכנה בלינוקס:

```
tweets_generator 454545 30 "path-to-file\exmaple.txt" 100
```

קובץ הקלט

לקריאה מהקובץ אנו ממליצים להשתמש ב- fgets() ו- sscanf(). אפשר להניח ש-fgets יחזיר null רק בסוף הקובץ (ז"א שלא תהיה שגיאה ואיך צורך לבדוק זאת).

על הקובץ ממנו תקראו את הציוצים ניתן להניח את ההנחות הבאות:

- כל ציוץ (משפט) ייכתב בשורה נפרדת.
  הבהרה: הכוונה שאותו ציוץ לא יפוצל בין מספר שורות בקובץ הקלט. באותה שורת קלט יכולים להופיע מספר ציוצים, ומובטח שכל אחד מהם יופיע בשלמותו.
- אורך ציוץ לא יעלה על 1000 תווים.
- הציוצים יכילו אותיות לועזיות ב-lower-case, מספרים, רווחים וסימני פיסוק סטנדרטים של ASCII בלבד.
- כל שתי מילים יופרדו ע"י רווח אחד או יותר. כלומר, מילה השמורה אצלכם במבנה נתונים לא אמורה להכיל את התווים רווח ' ' ושורה חדשה '\n'.
- בכל שורה יופיע סימן הנקודה '.' לפחות פעם אחת, ובכל מקרה התו האחרון במשפט בקובץ הקלט תמיד יהיה נקודה.
- מילה יכולה להכיל את התו נקודה בתוכה (למשל "a.a"), מילה זו לא נחשבת למילה המסיימת משפט, כיוון שהתו האחרון בה איננו נקודה.
- מילה יכולה להיות רק מספר. למשל, בציוץ "אחותי בת 3 ואוהבת תותים" - "3" נחשבת למילה בפני עצמה.
- אפשר להניח שכל מילה לא תהיה ארוכה מ-100 תווים.
- בקובץ יהיה לכל הפחות ציוץ אחד עם לפחות 2 מילים.
- אנו נתייחס לסימני פיסוק כאל תווים במילה, הווה אומר המילים הבאות הן 4 מילים שונות מבחינתנו:

hello #hello hello, hello.

5 שלב הלמידה

הוספת מילה למבנה נתונים

נוסיף כל מילה מקובץ הקלט למבנה נתונים מסוג רשימה מקושרת פעם אחת בדיוק. ההוספה תהיה לפי סדר ההופעה בקובץ הקלט. אם לא מוסיפים כל מילה פעם אחת בלבד ולפי סדר הופעתה, עלולה להתקבל תוצאה שונה מפתרון בית הספר וכתוצאה לגרום לכישלון בטסטים.

למשל, עבור הקובץ:

שרה שרה שיר שמח, שיר שמח שרה שרה.
שמח שמח, שמח שמח.
שמח שמח, רק שמחה בלב.

ניצור את הרשימה המקושרת:

שרה ← שיר ← שמח, ← שמח ← שרה. ← שמח. ← רק ← שמחה ← בלב.

שימו לב שכל מילה מופיעה ברשימה פעם אחת – המילים שרה ו- שרה. הן מילים שונות, וכך גם, שמח ו- שמח, . שמח.

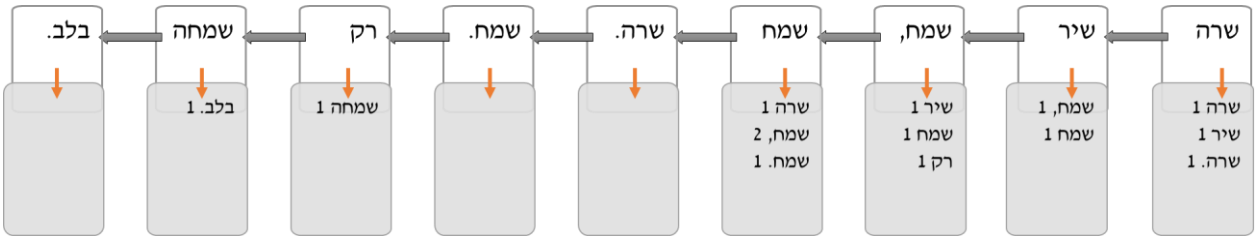
עדכון רשימת תדירויות

לכל מילה ברשימה המקושרת הנ"ל ניצור מערך דינמי של מילים עוקבות (מילים המופיעות לאחריה) ואת התדירות בה הן מופיעות. גם למערך הדינמי צריך להוסיף כל מילה עוקבת פעם אחת בלבד ועל פי סדר הופעתה בקובץ (אחרת עלולים להיכשל בטסטים). אם נתקלים באותה מילה עוקבת יותר מפעם אחת, צריך לעדכן את התדירות (מספר המופעים) שלה ככל שנתקדם בקריאת הקובץ.

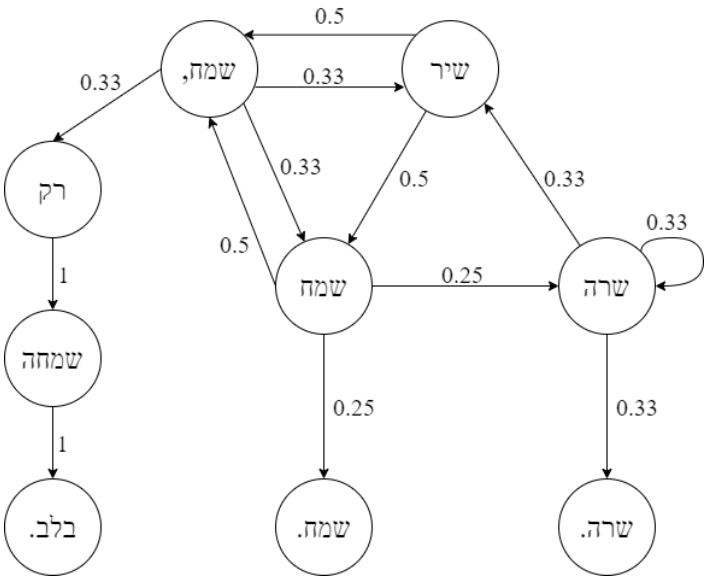
למשל, עבור הקובץ לעיל, המילה "שמח" מופיעה 4 פעמים, ורשימת התדירויות של המילים העוקבות אחריה:

אינקדס	מילה	מספר מופעים אחרי המילה "שמח"
0	שרה	1
1	שמח,	2
2	שמח.	1

בסיום שלב הלמידה, כך יראה מבנה הנתונים:



הערה: ניתן להסתכל על הנתונים כאוסף הסתברויות ("סיכויים") במקום תדירויות ולהציגם בתרשים כך[2]:



אך אנו נשתמש במבנה הראשון שפירטנו לעיל.

6 יצירת ציוץ

בחירת המילה הראשונה - פונקציית get_first_random_node

פונקציה זו תחזיר את אחת מהמילים בטקסט המקורי שאינה מילה אחרונה (אינה מסתיימת בנקודה). הפונקציה תבחר מילה באופן רנדומלי ובהתפלגות אחידה, כלומר לכל מילה יש הסתברות ("סיכוי") שווה להיבחר מבין כל המילים במבנה נתונים שאינן אחרונות:

נגריל מספר *i* באמצעות פונקציית get_random_number (הממומשת עבורכם) ובהתאם נבחר את המילה ה-i ברשימה המקושרת (בהנחה שהרשימה ממוינת לפי סדר הופעת המילים בקובץ, ר' סעיף 5). אם המילה ה-i היא מילה אחרונה, נחזור על ההגרלה ככל שצריך.

למשל, עבור הקובץ לעיל, נניח שהגרלנו וקיבלנו את המספר 4, אזי נבחרה המילה "שרה." מילה זו היא מילה אחרונה, ולכן נאלץ להגריל

[2] העשרה: מודל מסוג זה נקרא "שרשרת מרקוב". https://en.wikipedia.org/wiki/Markov_chain

מספר חדש. נניח שהגרלנו את המספר 1, אזי נבחרה המילה "שיר" ואותה נחזיר.

בחירת המילים הבאות - פונקציית get_next_random_node

פונקציה זו תקבל מילה, ותחזיר את אחת מהמילים העוקבות לה (בטקסט המקורי), באופן רנדומלי, כך שהסיכוי של כל מילה עוקבת להיבחר פרופורציונלי לתדירות שבה היא מופיעה.

נגריל מספר *i* באמצעות get_random_number ונחזיר את המילה המתאימה ברשימת התדירויות, בהתחשב בכמות המופעים של כל המילים ברשימה.
באופן זה, כל עוד ה-seed זהה, תקבלו תמיד את אותה תוצאה.

למשל, עבור המילה "שמח" כקלט, אלה התוצאות שיוחזרו בהתאם לערך שיוחזר מ- get_random_number:

ערך ההחזרה של get_random_number	ערך ההחזרה של get_next_random_node
0	"שרה"
1	"שמח,"
2	"שמח,"
3	"שמח."

המילה "שמח," הופיעה פעמיים אחרי המילה "שמח", ולכן יש לה הסתברות ("סיכוי") של 50% להיבחר.

תהליך יצירת משפט (ציוץ) שלם

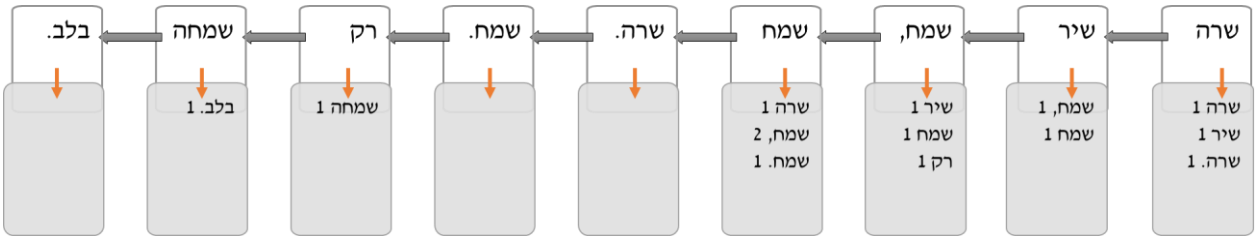
1. נבחר את המילה הראשונה בציוץ בהתפלגות אחידה, כלומר לכל מילה יש הסתברות ("סיכוי") שווה להיבחר) מבין כל המילים במבנה נתונים שאינן מילה אחרונה (= אינן מסתיימות בנקודה).
2. בעזרת פונקציית get_next_random_node נבחר את המילים הבאות בציוץ.
3. נסיים כשנגיע למילה המסיימת משפט (מסתיימת בנקודה), או כשאורך הציוץ הינו מקסימלי.

הערות:
- מספר המילים המקסימלי לציוץ שמייצרים הינו 20.
- במקרה של הגעה לאורך ציוץ מקסימלי <u>אין</u> להוסיף נקודה בסוף הציוץ.

- אם תקבלו כמות מילים הקטנה מכמות הציוצים שתתבקשו ליצור, מותר שיהיו כפילויות.

דוגמה ליצירת ציוץ

תזכורת: לאחר שלב הלמידה, קיבלנו את מבנה הנתונים הבא:



1. מבין המילים במבנה נתונים, מגרילים מילה ראשונה: "שיר".
2. מגרילים את המילה הבאה בציוץ, מבין המילים שמופיעות אחריה:
   - המילה "שמח," מופיעה פעם אחת,
   - המילה "שמח" מופיעה פעם אחת.

   הגרלנו "שמח".
3. מילה זו הופיעה 4 פעמים בטקסט (לאו דווקא מיד אחרי המילה "שיר"), ואחריה הופיעו:
   - המילה "שרה" פעם אחת,
   - המילה "שמח," פעמיים,
   - המילה "שמח." פעם אחת.

   ולכן באופן דומה נבחר מבין מילים אלו מילה אחת שתהיה בציוץ.
4. נמשיך באותו אופן ונסיים כשנגיע למילה שהיא סוף משפט (לדוגמת המילה "שרה.", בה יש תו אחרון נקודה). דוגמה לציוץ אפשרי: שיר שמח שמח, שמח שרה שרה שרה.

7 מבני הנתונים

רשימה מקושרת

יש להשתמש ברשימה המקושרת הנתונה עבורכם בקבצי linked_list.h ו- linked_list.c, אין להגיש או לשנות קבצים אלה .

שימו לב לטיפוס של הדאטא ב- Node.

שימו לב שהפונקציה add() של linkedList מכילה פעולת malloc, עליכם לשחרר בסיום התוכנית גם את זיכרון זה.

את מבנה הנתונים תממשו בעזרת רשימה מקושרת של MarkovNodes.

עליכם לממש את המבנים (structs) הבאים:

Struct – MarkovNode המכיל את השדות:

- data – מצביע אל תוכן המילה.
- counter_list – מצביע למערך של NextNodeCounter המכיל את כל המילים העוקבות האפשריות ע"פ הטקסט הנתון.
  - את המערך תקצו דינאמית, אתם אחראים להקצות את הזיכרון שלו ולשחרר אותו בסיום השימוש.
  - מערך זה בד"כ קטן ולכן נשתמש באסטרטגיה הבאה: בכל פעם שנרצה להכניס מילה חדשה למערך, נבצע realloc(), ונגדיל את גודלו ב-1.
  - עבור מילה שמסתיימת בנקודה (סוף משפט), המערך NextNodeCounter יצביע ל-NULL.
- כל שדה נוסף שתרצו לשימושכם.

Struct – NextNodeCounter המכיל את השדות:

- markov_node – מצביע אל המילה הבאה האפשרית בציוץ.
- frequency – שסופר את כמות הפעמים שהמילה $word_2$ מופיעה מיד אחרי $word_1$ בטקסט.
- כל שדה נוסף שתרצו לשימושכם.

Struct - MarkovChain המכיל את השדה:

- database – מצביע לרשימה מקושרת המכילה את כל המילים הייחודיות בטקסט.

8 פונקציות למימוש

1. עליכם לממש את הפונקציות הבאות בקובץ **markov_chain.c**:

- `MarkovNode* get_first_random_node (MarkovChain *markov_chain);`
- `MarkovNode* get_next_random_node (MarkovNode *state_struct_ptr);`
- `void generate_random_sequence (MarkovChain *markov_chain, MarkovNode *first_node, int max_length);`
- `bool add_node_to_counter_list (MarkovNode *first_node, MarkovNode *second_node);`
- `Node* get_node_from_database (MarkovChain *markov_chain, char *data_ptr);`
- `Node* add_to_database (MarkovChain *markov_chain, char *data_ptr);`
- `void free_markov_chain (MarkovChain **markov_chain);`

את התיעוד וההסבר המלא לכל פונקציה ניתן למצוא בקובץ **markov_chain.h**.
אין לשנות את חתימות הפונקציות הנ"ל, אך מותר להוסיף פונקציות נוספות לשימושכם.

כמו כן, העתיקו את הפונקציה הבאה (כמו שהיא) לקובץ markov_chain.c שלכם:

```
/**
 * Get random number between 0 and max_number [0, max_number).
 * @param max_number maximal number to return (not including)
 * @return Random number
 */
int get_random_number(int max_number)
{
    return rand() % max_number;
}
```

2. עליכם לממש את הפונקציה הבאה בקובץ **tweets_generator.c**:

- `void fill_database (FILE *fp, int words_to_read, MarkovChain *markov_chain);`

אשר מקבלת קובץ ומספר מילים לקריאה ומצביע למבנה נתונים של מרקוב, קוראת מהקובץ את מספר המילים לקריאה וממלאת את מבנה הנתונים.

9 דגשים והנחיות לתרגיל

- בסיום הריצה עליכם לשחרר את כלל המשאבים בהם השתמשתם, התוכנית שלכם תיבדק ע"י valgrind ויורדו נקודות במקרה של דליפות זיכרון.
- במקרה של שגיאת הקצאת זיכרון הנגרמה עקב **malloc/realloc/calloc**, יש לשחרר את כל הזיכרון שהוקצה עד כה בתוכנית, וכן להדפיס הודעת שגיאה מתאימה ל-stdout המתחילה ב-
  "Allocation failure:"
  ולצאת מהתוכנית עם **EXIT_FAILURE**. אין להשתמש ב-exit().
- אם לתוכנית שלכם יוצאת תוצאה זהה לזו של פתרון ביי"ס (כאשר משתמשים באותו ה-seed), זה אומר שככל הנראה הקוד שלכם תקין. קבלה של תוצאות שונות אומרת שיש לכם טעות בקוד. אם התוצאות שלכם שונות- שימו לב שבחירת המילים שלכם לציוץ מתבצעת באותו אופן שמוגדר בתרגיל.
- אם אפשרי, תעדיפו תמיד לעבוד עם int/long מאשר float/double. ניתן לפתור את התרגיל כולו בעזרת שימוש במספרים שלמים בלבד.
- כל ציוץ יודפס בשורה נפרדת ל-stdout. בתחילת כל שורה יש לכתוב: Tweet <number>:
  לדוגמה:

  Tweet 6: hello, nice to meet you.
- אין להשתמש בקורס ב-VLA (variable length array), כלומר מערך במחסנית שגודלו נקבע ע"י משתנה. שימוש שכזה יגרור הורדת ציון משמעותי מתרגיל.

10 נהלי הגשה

- תרגיל זה הינו התרגיל המסכם של שפת C. יש לתרגיל שני חלקים, החלק הראשון מהווה הכנה לחלק השני. החלק השני יתפרסם רק לאחר ההגשה של החלק הראשון (בשבוע הבא).
- קראו בקפידה את הוראות חלק זה של התרגיל. התרגיל מורכב ואנו ממליצים להתחיל לעבוד עליו כמה שיותר מוקדם. זכרו כי התרגיל מוגש ביחידים, ואנו רואים העתקות בחומרה רבה!
- יש להגיש את התרגיל באמצעות ה-git האוניברסיטאי ע"פ הנהלים במודל.
- יש להגיש את markov_chain.c markov_chain.h tweets_generator.c בלבד.

- התרגיל נבדק על מחשבי האוניברסיטה, ולכן עליכם לבדוק כי הפתרון שלכם רץ ועובד גם במחשבים אלו.
- כחלק מהבדיקות תיבדקו על סגנון כתיבה. חוסר שימוש בקבועים עלול לגרור הורדת נקודות.
- כשלון בקומפילציה או ב-presubmit יגרור ציון 0 בתרגיל.
- כדי לקמפל את התוכנית תוכלו להיעזר בפקודה הבאה:

```
gcc -Wall -Wextra -Wvla -std=c99 tweets_generator.c
markov_chain.c linked_list.c  -o tweets_generator
```

- תוכלו להריץ את פתרון ביי"ס במחשבי האוניברסיטה, או בגישה מרחוק בעזרת הפקודה הבאה ב-CLI:

```
~proglab/school_solution/ex3a/schoolSolution
```

- את בדיקת ה-presubmit תוכלו להריץ באמצעות הפקודה הבאה ב-CLI:

```
~proglab/presubmit/ex3a/run
```

- הציון הסופי של חלק א' הוא הציון שמקבלים ב-presubmit, טסטים נוספים ירוצו על חלק ב' בלבד.