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 א' 11 פברואר 2024

חשוב: חלק א' של תרגיל 3 צריך לעבור פרה-סבמיט בלבד והוא מהווה 25% מהציון הסופי של תרגיל 3

נושאי התרגיל: ●מצביעים ●מערכים דינמיים ●קריאה מקבצים ●מחולל טקסט ●ניהול זיכרון

הקדמה

1 רקע

עיבוד שפה טבעית (NLP – Natural Language Processing)

אחד מהתחומים החשובים ביותר כיום במדעי המחשב הוא התחום של עיבוד שפה טבעית (NLP). זהו תחום רחב המנסה לגשר בין שפת היום-יום לבין המחשב. השימושים הפופולריים ביותר של NLP כיום הוא "לגרום" למחשב להבין דיבור של אדם (כמו Google Assistant, Siri, Alexa וכו'), וכמובן יצירת אינטרקציה עם מודלי שפה גדולים (LLM) כדוגמאת ChatGPT.

בתרגיל זה נתמקד בנושא של NLP, שימוש בתוכנת מחשב על מנת לייצר משפטים/ציוצים (כינוי להודעות המתפרסמות ברשת החברתית טוויטר, כיום X) חדשים, בהתבסס על מאגר קיים של ציוצים.

נבצע זאת באמצעות שרשראות מרקוב (Markov chains) .

2 אופן פעולת התוכנית

נציג כעת בקצרה את שלושת השלבים שלפיהם תפעל התוכנית, ובהמשך המסמך נפרט לעומק על כל שלב.
חשוב לנו שתהיה לכם האופציה לבדוק את התרגיל בחלקים תוך כדי הכתיבה, ולכן המסמך כתוב בסדר המאפשר כתיבה נכונה ואינקרמנטלית לפתרון התרגיל, אך לא בהכרח בסדר בו התוכנית תפעל.
אחרי שתסיימו לממש כל אחד מהשלבים תוכלו לבדוק את הקוד שלכם לפני שתעברו לחלק הבא.

בנוסף, ריכזנו עבורכם את ההנחות עבור כל שלב, אנא הקפידו על קריאתם לפני המימוש שלכם.

שלב הקלט (חלק ב' של המימוש)
התוכנית תקבל כקלט קובץ טקסט Text corpus המכיל מאגר גדול של ציוצים קיימים, שלפיהם היא תבנה את הציוצים החדשים. על מנת שלא נבלבל, החלטנו לקרוא בינתיים לציוצים שבקובץ הקלט text corpus משפטים כדי להבדיל אותם מהפלט של התוכנית.

שלב הלמידה (חלק א' של המימוש)
התוכנית תיקרא את ה-corpus. היא תשמור ותעבד את המשפטים והמילים בו לתוך מבנה נתונים. עבור כל מילה, היא תחשב בנוסף את תדירות המופעים (frequency) של המילים המופיעות אחריה בקובץ.

שלב הפלט – יצירת ציוצים (חלק ג' של המימוש)
התוכנית תשתמש במבני הנתונים שיצרנו ובתדירויות שחושבו מקובץ הקלט כדי לייצר ציוצים באופן הסתברותי.

שימו ♥ בכל חלק בתרגיל, אתם תממשו את השלבים בתוכנית לפי הטבלה הבאה:

חלק של מימוש	חלק א'	חלק ב'	חלק ג'
שלב התוכנית	למידה	קלט	פלט

3 שרשראות מרקוב

שרשרת מרקוב היא מודל המתאר תהליך נתון כשרשרת (סידרה) של מצבים עוקבים. המצב הראשון בשרשרת נבחר באקראי. המצב הבא אחריו נבחר גם הוא באקראי, כאשר הסיכוי שלו להיבחר תלוי אך ורק בזהות המצב הנוכחי. המצב הבא נבחר באותה דרך, וחוזר חלילה, עד להגעה למצב סיום.

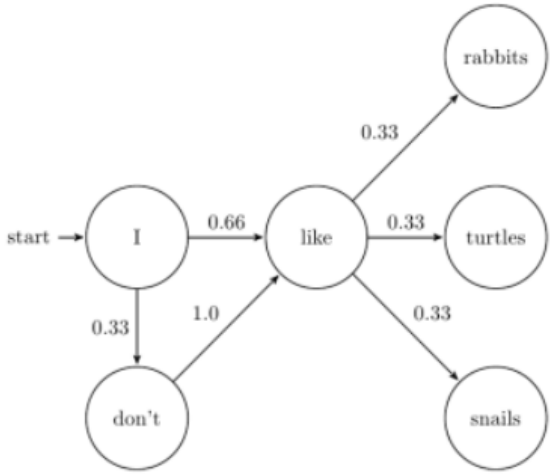
במקרה שלנו, נשתמש בשרשראות מרקוב על מנת לתאר את תהליך היצירה של ציוץ. כלומר המצבים בשרשרת שלנו הן מילים - המילה הראשונה נבחרת באקראי. כל מילה שבאה אחרי נבחרת גם היא באקראי, כאשר הסיכוי שלה להיבחר תלוי אך ורק בזהות המילה שלפניה. תהליך זה, ממש כמו בהגדרה של שרשרת מרקוב בפסקה הקודמת, חוזר חלילה עד להגעה למילה המסיימת משפט (נסביר בהמשך מה זה אומר בדיוק).

ניתן להסתכל על תהליך היצירה של שרשרת מרקוב כטיול בגרף. הצמתים מייצגים את המצבים (מילים) והקשתות המכוונות מייצגות את ההסתברות (סיכוי) לעבור מהמצב הנוכחי (מילה נוכחית) למצב הבא (המילה הבאה). למשל בדוגמא להלן:

- מהמילה I ישנו סיכוי של שליש לעבור למילה don't, וסיכוי של שני שליש לעבור למילה like.
- מהמילה don't נעבור תמיד אל המילה like.
- מהמילה like נעבור בהסתברות שווה בין המילים rabbits, turtles, snails (כלומר ההסתברות לבחור בין כל אחת מהמילים זהה, ושווה לשליש).

דוגמא מגניבה שממחישה את שרשאות מרקוב (מומלץ לעיין!).

נשים ♥ סכום ההסתברויות (הסיכויים) הכתובות על החיצים היוצאים ממילה מסוימת תמיד יהיה 1.



4 כלים לכתיבת התרגיל

פונקציות לבחירת מספר רנדומלי

המחשב לא יכול ליצר ערך אקראי אמיתי, לכן מתכנתים נעזרים בפונקציות "Pseudo-random generator". פונקציות אלו מחשבות, בעזרת סדרה של פעולות מתמטיות, ערך שלמראית עין נראה אקראי ובכל קריאה יוחזר ערך שונה. אנחנו נוהגים להגדיר לפונקציות אלו גרעין (**seed**), ערך כלשהו אשר עליו יופעלו הפעולות המתמטיות.
למה פונקציות אלו הן רק פסאודו? אם נשתמש באותו ה-seed בריצות שונות, נקבל את אותו רצף מספרים "אקראיים". (מוזמנים לנסות!)
בתרגיל תצטרכו להשתמש בפונקציות הבאות:

- **void srand**(seed) – פונקציה זו מקבלת כפרמטר מספר שלם אי-שלילי ומגדירה אותו כגרעין עבור הריצה הנוכחית.
- **int rand**() – פונקציה זו מחזירה מספר שלם פסאודו-אקראי בין 0 (כולל) ל-**RAND_MAX** (לא כולל).

בהמשך נסביר איך נשתמש ב-seed שנקבל בתור פרמטר בתחילת ריצת התוכנית.

פונקציות לקריאת קבצים

אנחנו ממליצים להשתמש בפונקציות **fgets()** ו-**sscanf()**.

כדאי לדעת

1. **RAND_MAX** הוא קבוע המוגדר מראש בספרייה stdlib.h.
2. בתחילת התוכנית הקומפיילר מאתחל את ערך ה-seed להיות 1, על כן אם לא נגדיר בעזרת srand(seed) את הגרעין, בכל הריצות נקבל את אותם ערכים "אקראיים".
3. דרך אחת לאפשר לתוכנית שלכם לבחור ערך seed ייחודי בכל ריצה הוא על ידי הגדרת ערך ה- seed כתלות בזמן הנוכחי, תוכלו לראות את זה גם בלינק שצירפנו לכם על srand(seed) למעלה.

```
/* Intializes random number generator */
srand((unsigned) time(&t));
```

5 קבצים

סיפקנו לכם 3 קבצי קוד וקובץ Text Corpus לדוגמא:

1. **markov_chain.h** – קובץ המכיל את השלד של חלק מהפונקציות שעליכם לכתוב.
2. **linked_list.h, linked_list.c** – קבצים ובהם מימוש של רשימה מקושרת לשימושכם.
3. **justdoit_tweets.txt** – דוגמא לקובץ קלט. הקובץ מכיל ~4400 משפטים ותוכלו לבדוק את התוכנית שלכם באמצעותו. השתדלנו לנקות את המאגר מתוכן פוגעני. אם החמצנו משהו, אנו מתנצלים על כך מראש.

אתם צריכים להגיש את הקבצים:

1. **markov_chain.h** מעודכן עם הסטראקטים שתכתבו.
2. **markov_chain.c** עם מימוש של הפונקציות המוכרזות בקובץ ה-h המתאים.
3. **tweets_generator.c** שמכיל את ה-main ועוד פונקציות שנפרט עליהם בהמשך.

לפני שנצלול למימוש התרגיל נסתכל על הגדרות שישמשו אותנו לאורך התרגיל.

הגדרה – משפט וציוץ: כדי להבדיל מתי אנחנו מתכוונים לקובץ הקלט ומתי לפלט התוכנית, ניתן להם שמות שונים. קובץ הקלט מורכב מ-משפטים, התוכנית שלכם תייצר ציוצים ותדפיס אותם כפלט ל-**stdout**.

הגדרה – מילה המסיימת משפט (מילה אחרונה): משפט או ציוץ **תמיד יסתיימו** במילה אשר התו האחרון בה הוא נקודה (התו '.' קוד ascii: 46).

חלק א' – שלב הלמידה

בחלק זה אתם תממשו את אבני היסוד של התוכנית – את ה-structs שיחזיקו את הנתונים על המשפטים ואת הפעולות עליהם אשר יעזרו לעבד את המידע בתהליך הלמידה.

מימשנו עבורכם - רשימה מקושרת

לצורך פתירת התרגיל תצטרכו להשתמש ברשימה מקושרת. עליכם להשתמש במימוש שסיפקנו עבורכם בקבצים **linked_list.h** ו- **linked_list.c**. קבצים אלו אינם להגשה (ראו חלק הקדמה פרק 5), ולכן **לא תוכלו** לשנות, להוסיף או להרחיב קבצים אלו. עיינו במבני הנתונים לפני שתתחילו את האימפלמנטציה שלכם, תוודאו שאתם יודעים לאיזה מטרות הם. שימו לב שהפונקציה *add()* מכילה פעולת **malloc**. עליכם לשחרר בסיום התוכנית גם את הזיכרון הזה. את מבנה הנתונים הכללי תממשו בעזרת רשימה מקושרת של MarkovNode (ר' למטה).

1 מבני הנתונים (structs)

בתת-חלק זה עליכם להכריז על ה-structs הבאים:

MarkovChain

Struct זה הוא מבנה נתונים המתאר מודל מסוג שרשרת מרקוב, אשר ישמש אתכם ליצירת ציוצים. את תוכנו של מבנה נתונים זה תיצרו בהסתמך על תוכנו של קובץ הקלט. ההכרזה שלכם צריכה לכלול את השדה:

- **database** – מצביע לרשימה מקושרת המכילה את כל המילים הייחודיות בטקסט, כמצביעים למשתנים מטיפוס MarkovNode.

MarkovNode

Struct זה מתאר את הקודקודים בשרשרת מרקוב שבעזרתה תייצרו את הציוצים בהמשך. ההכרזה שלכם צריכה לכלול את השדות:

- **data** – מצביע אל תוכן המילה.
- **frequencies_list** – מצביע המשמש כמערך דינאמי של איברים מטיפוס MarkovNodeFrequency (ר' למטה). מערך זה יכיל את כל המילים העוקבות האפשריות אחרי המילה הנוכחית, על פי הטקסט הנתון.
  - אתם אחראים להקצאת המערך ושחרורו במהלך התוכנית.
  - עבור מילים המסיימות משפט, שדה זה יצביע ל-**NULL**.
- כל שדה נוסף שתרצו לשימושכם.

MarkovNodeFrequency

Struct זה מתאר את תדירות המופעים של מילה מסוימת ($word_2$) אחרי מילה אחרת ($word_1$) בקובץ הקלט, כפי שיובהר בתת-החלק הבא. קודקוד כזה רלוונטי רק אם $word_2$ אכן הופיעה אחרי $word_1$ ב-text corpus. ההכרזה שלכם צריכה לכלול את השדות:

- **markov_node** – מצביע אל קודקוד המרקוב של המילה העוקבת במשפט ($word_2$).
- **frequency** – סופר את מספר הפעמים ש-$word_2$ מופיעה מיד אחרי $word_1$ בטקסט.
- כל שדה נוסף שתרצו לשימושכם.

2 פונקציות הלמידה

בתת-חלק זה אתם תממשו פונקציות על מבני הנתונים שמימשתם בתת-החלק הקודם.
לצורך הדגמת שלב הלמידה של התוכנית שלנו, נשתמש בקובץ הבא:

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

הוספת מילה למבנה נתונים

כל מילה מקובץ הקלט תופיע ברשימה המקושרת (השדה database של ה-Struct MarkovChain) פעם אחת בדיוק. המילים בשרשרת יהיו מסודרות לפי סדר המופע הראשון של כל אחת בקובץ הקלט. נציין שאם מילה תישמר יותר מפעם אחת או שסדרן לא יהיה לפי סדר ההופעה בקובץ, עלול להתקבל פתרון שונה מפתרון בית הספר מה שיגרור כישלון בטסטים.

דוגמא לרשימה מקושרת של שרשרת מרקוב

עבור קובץ הקלט לעיל ניצור את הרשימה המקושרת:

שימו לב שכל מילה מופיעה ברשימה פעם אחת – המילים '***שרה***' ו'***שרה.***' הן מילים שונות. וכנ"ל גם '***שמח***' '***שמח.***' '***שמח,***' .

עבשיו תממשו בקובץ **markov_chain.c** את הפונקציות האחראיות להכנסה ושליפה של קודקודים:

- `Node* add_to_database(MarkovChain *markov_chain, char *data_ptr);`
- `Node* get_node_from_database(MarkovChain *markov_chain, char *data_ptr);`

אתם לא יכולים להניח ש:

- ✗ ה-database של שרשת המרקוב המתקבלת אינו ריק. יש להתמודד גם עם המקרה של מאגר ריק והכנסת המילה הראשונה אליו.

עדכון רשימת תדירויות

עבור כל מילה ברשימה המקושרת הנ"ל, עליכם לעדכן את המערך הדינמי frequencies_list ב-Struct של אותה מילה (להלן, "המילה הנוכחית"). לאחר העדכון, המערך יכיל את כל המילים המופיעות אחרי מילה זו בקובץ הקלט (להלן "המילים העוקבות"), ולכל מילה עוקבת תישמר התדירות שלה, כלומר מספר המופעים שלה לאחר המילה הנוכחית. בנוסף, על כל מילה עוקבת להופיע פעם אחת בלבד ברשימה, ועל פי סדר הופעתה בקובץ לאחר המילה הנוכחית (אחרת עלולים להיכשל בטסטים). אם נתקלים באותה מילה עוקבת יותר מפעם אחת, יש לעדכן את התדירות שלה ככל שנתקדם בקריאת הקובץ.

אם המערך מלא בעת ניסיון להוספת מילה חדשה, עליכם להגדילו באיבר אחד בדיוק תוך שימוש בפונקציית *realloc()* במידת הצורך.
כל עוד לא נצפתה מילה עוקבת למילה הנוכחית, אין צורך להקצות את רשימת התדירויות, ואורכה יהיה 0.

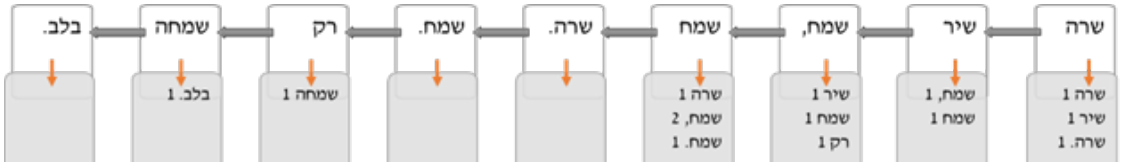
דוגמא לרשימת תדירויות לכל מילה בשרשרת מרקוב

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

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

לכן הקודקוד של המילה "שמח" יראה כך:

בסיום שלב הלמידה, כך יראה מבנה הנתונים:



עכשיו תממשו בקובץ **markov_chain.c** את הפונקציה המעדכנת את רשימת התדירויות:

```
bool add_node_to_counter_list(MarkovNode *first_node,
                              MarkovNode *second_node);
```

ואת הפונקציה שמשחררת את כלל הזיכרון של שרשרת המרקוב:

```
void free_markov_chain(MarkovChain ** ptr_chain);
```

אתם יכולים להניח ש:

✓ בכל הפונקציות, אלא אם צוין אחרת, כל המצביעים המתקבלים כפרמטרים יצביעו לכתובת אמיתית, כלומר לא יצביעו ל-**NULL**.

עכשיו אחרי שמימשתם את החלק הזה, זה זמן מצוין לקחת הפסקה ולשתות קפה.

חלק ב' – שלב הקלט

בחלק זה תממשו את פונקציית ה-main, את קריאת הקלט והכנסתו למאגר הנתונים.

1 פונקציית ה-main ומילוי מאגר הנתונים

פונקציית ה-**main** שלנו תקבל את הארגומנטים הבאים (ובסדר הבא):

1. ערך **seed**: מספר שיינתן לפונקציית ה-srand(seed) בתחילת הריצה.
2. כמות הציוצים שנייצר.
3. נתיב (path) לקובץ ה-**Text corpus**: קובץ טקסט המכיל מאגר גדול של משפטים, שלפיהם נבנה את הציוצים החדשים.
4. הכמות הכוללת של המילים שיש לקרוא מן הקובץ: הפרמטר הזה הוא אופציונלי, כלומר התוכנית צריכה לתמוך בהרצה עם ארבעה פרמטרים ובהרצה עם שלושה פרמטרים. במקרה בו לא התקבל פרמטר זה, יש לקרוא את הקובץ כולו.

אתם יכולים להניח ש:

- ✓ ערך ה-seed שתקבלו הוא תקין כפרמטר לפונקציה srand. כלומר, ניתן להניח כי הפרמטר יהיה מספר שלם ואי שלילי (unsigned int).
- ✓ הפרמטר של כמות הציוצים הוא מספר שלם וחיובי (int).
- ✓ הפרמטר של כמות המילים שיש לקרוא מן הקובץ הוא מספר שלם וחיובי (int).

אתם לא יכולים להניח ש:

- ✗ הנתיב שניתן לקובץ ה-text corpus תקין.
  - במקרה בו הנתיב לא תקין - כלומר הקובץ לא קיים, או שלתוכנית אין הרשאות גישה אליו, יש להדפיס הודעת שגיאה מתאימה ל-**stdout** המתחילה ב-"*Error:* ".לאחר מכן יש לצאת מהתוכנית עם קוד **EXIT_FAILURE**.
- ✗ הפרמטר של כמות המילים שיש לקרוא יהיה קטן או שווה לכמות המילים שיש בקובץ בפועל.
  - במקרה שהתבקשתם לקרוא יותר מילים ממה שאפשר, יש לקרוא את הקובץ כולו ולהמשיך בתוכנית ללא כל הודעה.

אם כמות הפרמטרים שהתקבלו אינה 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**.

דוגמה להרצה תקנית לתוכנה ב*לינוקס*:

tweets_generator	454545	30	"path-to-file\text_corpus.txt"	100
שם התוכנית	**seed**	**מס' הציוצים שיש לייצר**	**הנתיב למאגר המשפטים**	**כמות המילים שיש לקרוא מהקובץ**

הנחות לקובץ הקלט:

- הפונקציה fgets() תחזיר **NULL** רק בסוף הקובץ (ז"א שלא תהיה שגיאה ואיך צורך לבדוק זאת).
- כל משפט ייכתב בשורה נפרדת. כלומר, אותו משפט לא יפוצל בין מספר שורות בקובץ, אך באותה שורה יכולים להופיע מספר משפטים ומובטח שכל אחד מהם יופיע בשלמותו.
- אורך משפט לא ייעלה על 1000 תווים, ואורך מילה לא יעלה על 100 תווים.
- המשפטים יכילו אותיות לועזיות ב-lower-case, מספרים, רווחים וסימני פיסוק סטנדרטיים של acii בלבד.

- כל שתי מילים יופרדו על ידי רווח אחד או יותר. כלומר, המילה שאתם שומרים במבנה הנתונים לא אמורה להכיל את התווים רווח (**' '**) ושורה חדשה (**'\n'**).
- בכל שורה יופיע סימן הנקודה (**'.'**) לפחות פעם אחת, ובכל מקרה התו האחרון במשפט תמיד יהיה נקודה.
- מילה יכולה להכיל את התו נקודה בתוכה (למשל המילה a.a היא מילה חוקית). מילים כאלו **לא** נחשבות למילים המסיימות משפט, כיוון שהתו האחרון בה איננו נקודה.
- מילה יכולה להיות גם רק מספר. למשל במשפט *אחותי בת 3 ואוהבת תותים* המילה *3* נחשבת מילה בפני עצמה.
- בקובץ יהיה לכל הפחות משפט אחד עם לפחות 2 מילים.
- מספר המילים לקריאה, אשר מתקבל כפרמטר לפונקציה main, לא יחתוך משפט באמצע. כלומר, המילה במיקום הרלוונטי בקובץ תהיה מילה המסיימת משפט.
- כמו שהסובר בחלק השלישי של המסמך (ההסבר על למידה) אנו נתייחס לסימני פיסוק כאל תווים במילה. כלומר, המילים הבאות הן 4 מילים שונות מבחינתנו:

hello #hello hello, hello.

לכו לממש בקובץ **tweets_generator.c** את פונקציית ה-***main*** המתוארת לעיל, ואת הפונקציה הבאה:

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

פונקציה זו מקבלת כפרמטרים קובץ, מספר מילים לקריאה ומצביע למבנה נתונים של שרשרת מרקוב, קוראת מהקובץ את מספר המילים לקריאה וממלאת את מבנה הנתונים.

ערך ההחזרה של fill_database צריך להיות 0 אם מילוי מבנה הנתונים הצליח, ו-1 במקרה של כישלון. במקרה של הקצאת זיכרון שלא הצליחה, בו הפונקציה מחזירה 1 כאמור, עליכם לדאוג לשחרור כל הזיכרון שהוקצה לתוכנית (לאחר היציאה מהפונקציה fill_database) ולסיים את הריצה.

שימו לב לדגשים וההנחיות לעיל עבור שתי הפונקציות!

חלק ג' – שלב הפלט ויצירת ציוץ

בחלק זה תממשו את הפונקציות שעוזרות לנו לייצר את הציוצים משרשרת המרקוב. בין היתר יהיו פונקציות האחראיות להגרלה ובחירת מילים באופן אקראי, ופונקציות האחראיות על חיבור המילים לכדי ציוץ.

אופן יצירת הציוצים

נשתמש במבני הנתונים שייצרנו, ובתדיריות (frequencies) שחישבנו מהקובץ כדי לייצר ציוצים באופן הסתברותי:

a. נגריל מילה ראשונה מאוסף המילים מתוך מבני הנתונים (למשל '***שמח***').
b. נשתמש ברשומה (entry) של אותה מילה במבנה הנתונים על מנת להגריל את המילה הבאה בציוץ. הסיכוי של מילה להיבחר בהגרלה פרופורציונאלי לתדירות בה היא מופיעה אחרי המילה הקודמת ב-corpus (למשל, הסיכוי שהמילה '***שמח***' תיבחר שוב גבוה יותר מאשר המילים '***שרה***', '***שמח.***' כי בעוד שהאחרונות מופיעות בקובץ פעם אחת בלבד אחרי המילה '***שמח***', הראשונה מופיעה פעמיים).
c. נמשיך לייצר את המילים הבאות באותה דרך, עד שנגיע למילה המסיימת משפט, או כשאורך הציוץ הינו מקסימלי. (למשל, '***שמח***'←'***שרה***'←'***שרה.***')
d. כל ציוץ שייצרנו באופן הנ"ל מהווה סדרת מרקוב, המילים בציוץ הן המצבים שנבחרו מתוך השרשרת.

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

```c
/**
 * 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;
}
```

בחירת המילה הראשונה – הפונקציה get_first_random_node

פונקציה זו תחזיר אחת מהמילים בטקסט **שאינה** מילה המסיימת משפט. הפונקציה תבחר מילה באופן רנדומלי ובהתפלגות אחידה, כלומר לכל מילה יש הסתברות (סיכוי) שווה להיבחר מבין כל המילים במבנה הנתונים שהן לא מסיימות משפט:

נגריל מספר i באמצעות הפונקציה get_random_number (הממומשת עבורכם) ובהתאם נבחר את המילה ה-i ברשימה המקושרת (בהנחה שהרשימה מסודרת לפי סדר הופעת המילים בקובץ). אם המילה ה-i היא מילה המסיימת משפט, נחזור על ההגרלה ככל שצריך.

למשל, עבור הקובץ לעיל, נניח שהגרלנו את המספר 4, אזי נבחרה המילה '***שרה.***' מילה זו היא מילה אחרונה, ולכן ניאלץ להגריל מספר חדש. נניח שהגרלנו את המספר 1, אזי נבחרה המילה '***שיר***' אותה נחזיר אחר והיא אינה מילה אחרונה.

ממשו את הפונקציה get_first_random_node בקובץ **markov_chain.c**:

- `MarkovNode* get_first_random_node(MarkovChain *markov_chain)`

בחירת המילים הבאות – הפונקציה get_next_random_node

פונקציה זו תקבל מילה ותחזיר את אחת המילים העוקבות לה (בטקסט המקורי), באופן רנדומלי, כך שהסיכוי שכל מילה עוקבת להיבחר פרופורציונלי לתדירות שבה היא מופיעה.
נגריל מספר i באמצעות הפונקציה get_random_number (הממומשת עבורכם) ובהתאם נבחר את המילה ה-i ברשימת התדירויות, בהתחשב בכמות המופעים של כל מילה ברשימה. באופן זה, כל עוד ה-seed זהה, תקבלו תמיד את אותה התוצאה.
למשל, עבור המילה '***שמח***' נקלט אלה התוצאות שיוחזרו בהתאם לערך שיוחזר מ-get_random_number:

ערך ההחזרה של get_random_number	המילה שתיבחר עבור האינדקס המוגרל
0	"שרה"
1	"שמח,"
2	"שמח,"
3	"שמח."

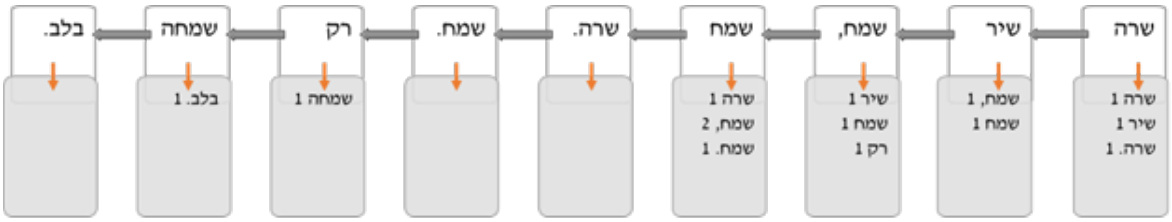
המילה '***שמח,***' הופיעה פעמיים אחרי המילה '***שמח***' ולכן יש לה הסתברות של 50% להיבחר.

ממשו את הפונקציה get_next_random_node בקובץ **markov_chain.c**:

- `MarkovNode* get_next_random_node(MarkovNode *state_struct_ptr)`

דוגמה ליצירת ציוץ

<u>תזכורת</u>: לאחר שלב הלמידה, קיבלנו את מבנה הנתונים הבא:



1. מבין המילים במבנה הנתונים, מגרילים מילה ראשונה: "שיר".
2. מגרילים את המילה הבאה בציוץ, ומבין המילים שמופיעות אחריה:
   a. המילה '***שמח,***' מופיעה פעם אחת.
   b. המילה '***שמח***' מופיעה פעם אחת.
3. נניח שהגרלנו את המילה '***שמח***', אשר מופיעה 4 פעמים בטקסט (לאו דווקא אחרי המילה '***שיר***'). אחריה הופיעו המילים: '***שרה***' – פעם אחת, '***שמח,***' – פעמיים, '***שמח.***' - פעם אחת. באופן דומה נבחר מבין מילים אלו מילה אחת שתהיה בציוץ.
4. נמשיך באותו אופן ונסיים כשנגיע למילה המסיימת משפט.

בקובץ generate_random_sequence ממשו את הפונקציה **markov_chain.c**:

```
• void generate_random_sequence( MarkovChain *markov_chain,
                                 *MarkovNode first_node, int max_length)
```

הערות ודגשים לחלק רביעי

- מספר המילים המקסימלי לציוץ שמייצרים הינו 20.
- במקרה של הגעה לאורך ציוץ מקסימלי, **אין** להוסיף נקודה בסוף הציוץ.

סיכום, דגשים ונהלי הגשה

1 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 בי"ס (כאשר משתמשים באותו ה-seed), זה אומר שבכל 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.

2 נהלי הגשה

- תרגיל זה הינו התרגיל המסכם של שפת C. יש לתרגיל שני חלקים, החלק הראשון מהווה הכנה לחלק השני. החלק השני יתפרסם רק מספר ימים לאחר פרסום חלק זה.

2 נהלי הגשה

- תרגיל זה הינו התרגיל המסכם של שפת C. יש לתרגיל שני חלקים, החלק הראשון מהווה הכנה לחלק השני.
- 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 הקבצים tweets_generator.c markov_chain.h markov_chain.c בלבד.
- התרגיל נבדק על מחשבי האוניברסיטה, ולכן עליכם לבדוק כי הפתרון שלכם רץ ועובד גם במחשבים אלו. שימו לב כי הרנדומליות שונה ממחשב למחשב אפילו אם מקבעים את ה-seed עם srand. כדי להשוות עם פתרון בית הספר צריך להריץ על מחשבי האוניברסיטה כדי לקבל רנדומליות זהה.
- בחלק 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 בי"ס במחשבי האוניברסיטה, או בגישה מרחוק בעזרת הפקודה הבאה ב-CLI:

```
~labcc/school_solution/ex3a/schoolSolution
```

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

```
~labcc/presubmit/ex3a/run
```

- נזכיר כי הציון הסופי של חלק א' הוא הציון שמקבלים ב-presubmit, טסטים נוספים ירוצו על חלק ב' בלבד. משמע אם עברתם את ה- presubmit של חלק זה ללא כל שגיאות או אזהרות ולא ימצאו שגיאות וולגרינד או קודינג סטייל אצלכם לאחר מכן תקבלו ציון מלא על חלק א'.
- מועד הגשה של חלק זה: יום א' 11 לפברואר ב-23:59
בונוס 5+:הגשה עד יום א' 4 לפברואר ב-23:59
- בונוס של 1+/2+/3+נקודות: הגשה יום/יומיים/שלושה ימים מראש, כרגיל.