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

סדנת תכנות C ו-++C - תרגיל מסכם בשפת C (חלק ב׳)

מועד הגשה (חלק ב׳): יום ד' 24 לאוגוסט ב-22:00

נושאי התרגיל: #מצביעים #מצביעים_לפונקציות #תכנות_גנרי #ניהול_זיכרון

חלק ב - תכנות גנרי

בחלק זה של התרגיל נהפוך את הקוד שכתבנו בחלק א' לקוד גנרי. מומלץ להשתמש בקוד שכתבתם לחלק א׳ ולעדכן אותו בהתאם לשינויים ולתוספות. **מי שלא בטוח מה זה קוד גנרי ואיך מממשים קוד כזה- נמליץ לו שיחזור על השיעורים והתרגולים לפני שהוא צולל לעומק התרגיל.**

בחלק זה של התרגיל נעדכן את מבנה הנתונים markov_chain כך שתוכל ליצור שרשראות של טיפוסים שונים (ולא רק tweets). בכדי שתוכלו לבחון את מבנה הנתונים הגנרי, תכתבו שני קבצים שישתמשו במבנה הנתונים: tweets_generator (בדומה למה שכתבתם בחלק א') ו-snakes_and_ladders (קובץ חדש).

שני הקבצים tweets_generator.c ו-snakes_and_ladders.c יכילו פונקציית main ונריץ כל פעם רק אחת מהם.

בכדי לוודא שמבנה הנתונים markov_chain ממומשת באופן גנרי לחלוטין, הטסטים האוטומטיים של בית הספר ירוצו גם על טיפוסים שאתם לא מכירים.

1 קבצים

כמו בחלק א׳, סיפקנו עבורכם קבצי קוד וקובץ קלט:

- justdoit_tweets.txt - קובץ הקלט שהתכנית tweets_generator מקבלת.
- markov_chain.h שמכיל את השלד של הסטראקטים המעודכנים.
- linkedList.c linkedList.h - רשימה מקושרת לשימושכם.
- snakes_and_ladders.c - קובץ (ממומש חלקית) המשתמש במבנה הנתונים מרקוב שכתבתם.

אתם צריכים להגיש:

- markov_chain.h מעודכן עם הסטראקטים שתכתבו.
- markov_chain.c מימוש של הפונקציות שנמצאות ב- markov_chain.h.
- tweets_generator.c דומה לקובץ שהגשתם בחלק א׳ אך עם שינויים מותאמים למבנה הנתונים החדש.
- snakes_and_ladders.c הקובץ משתמש במבנה הנתונים מרקוב שכתבתם, תצטרכו לממש אותו בהתאם להוראות שיפורטו בהמשך.
- makefile עם פקודות מתאימות לקימפול והרצה של שתי תכניות שונות, אחת שמריצה את tweets generator עם מבנה הנתונים של מרקוב ואחת שמריצה את snakes and ladders עם מבנה הנתונים של מרקוב. פירוט בהמשך.

2 מבני נתונים

עליכם לשנות את ה-structs הבאים בקבצים markov_chain.h ו-markov_chain.c:

MarkovNode

- המצביע למילה יהפוך להיות מצביע לדאטא גנרי.

NextNodeCounter

- אין שינויים.

MarkovChain

יש להוסיף מצביעים לפונקציות גנריות:

- print_func מצביע לפונקציה שמקבלת מצביע מטיפוס גנרי, לא מחזירה כלום, ומדפיסה את הדאטא.
- comp_func מצביע לפונקציה שמקבלת שני מצביעים לדאטא גנרי מאותו טיפוס, ומחזירה:
  - ערך חיובי אם הראשון יותר גדול מהשני;
  - ערך שלילי אם השני יותר גדול;
  - 0 אם שניהם שווים.
- free_data מצביע לפונקציה שמקבלת מצביע מטיפוס גנרי, לא מחזירה כלום, ומשחררת את הזכרון של המשתנה אותו קיבלה.
- copy_func מצביע לפונקציה שמקבלת מצביע מטיפוס גנרי, ומחזירה עותק שלו המוקצה דינאמית.
- is_last מצביע לפונקציה שמקבלת מצביע מטיפוס גנרי ומחזירה ערך בוליאני true אם הדאטא הוא אחרון בשרשרת, ו-false אחרת.

<u>הנחיות</u>:

1. מומלץ להשתמש ב-typedef כדי ליצור טיפוס חדש של מצביע לפונקציה, למשל:

```
typedef bool (*IsEven)(int);
```

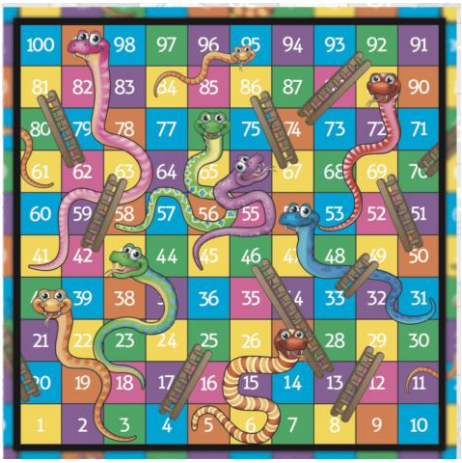
   מייצר טיפוס חדש בשם IsEven של מצביע לפונקציה אשר מקבלת int ומחזירה bool.
2. בקובץ markov_chain.h יש את השלד של הסטראקט, אסור לשנות את השמות של המשתנים ואסור להוסיף עוד משתנים ל-MarkovChain. (כדי שנוכל להריץ טסטים)
3. בקבצים של מבנה הנתונים מרקוב <u>לא ממשים</u> את הפונקציות הנ"ל, אלא רק מוסיפים מצביעים לפונקציות.
4. צריך להתאים את הקוד ב-markov_chain.c להשתמש במצביעים של הפונקציות לעיל במקום להשתמש בפונקציות של מחרוזות (למשל להשתמש ב-comp_func במקום ב-strcmp).

3 שימוש במבנה הנתונים markov_chain על ידי tweets_generator

- **הקלט והפלט** זהים לקלט ופלט מחלק א' של התרגיל.
- **שלב הלמידה ושלב יצירת המשפטים** זהים <u>מבחינה לוגית</u> לשלבים אלו בחלק א', כלומר הלוגיקה נשארת זהה אך הקוד צריך להתאים לעדכונים במבנה הנתונים markov_chain.
- בכדי להתאים את הקוד ב- tweets_generator.c כך שיעבוד עם מבנה הנתונים המעודכן, צריך לחשוב אילו פונקציות לספק ל-MarkovChain בקובץ tweets_generator.c כדי שזה יעבוד על מחרוזות (טיפ: חלק מהפונקציות קיימות ב- <string.h>).

- מומלץ להשתמש בקובץ justdoit_tweets.txt כדי להריץ את התוכנית לוודא נכונות.

4 שימוש ב markov_chain על ידי snakes_and_ladders



כדי לבדוק גנריות, נרצה לבדוק את מבנה הנתונים מרקוב על טיפוס נוסף, נשתמש במשחק סולמות ונחשים לבדיקת ה-MakrovChain.

הנחיות והנחות:

- הלוח בגודל 100 (10*10), מתחיל מתא מספר 1, ומסתיים בתא מספר 100.
- לצורך פשטות, נניח שמשחקים את המשחק עם שחקן יחיד.
- אנו נתייחס לכל משחק של שחקן יחיד כ"מסלול" (כמו "משפט" בחלק א'). כל מסלול יכלול רצף חוקי של תאים.
- התוכנית תייצר מסלולים אפשריים של שחקן במשחק, מהתא הראשון בלוח (תא 1) לתא האחרון (100).
- ניתן להניח שאף תא לא מכיל נחש וסולם בו זמנית.

מסופק עבורכם קובץ snakes_and_ladders.c שמכיל את הסטראקט הבא המייצג תא במשחק:

```
typedef struct Cell {
  int number; // cell number (1-100)
  int ladder_to;  // ladder_to represents the jump of the ladder in case there is one from this cell
  int snake_to;   // snake_to represents the jump of the snake in case there is one from this cell
  //both ladder_to and snake_to should be -1 if the cell doesn't have them
} Cell;
```

התכנית:

- **קלט**
  - ערך **seed** – מספר שיינתן לפונקציית ה-srand() פעם אחת בתחילת הריצה. ניתן להניח כי הוא מספר שלם חיובי (unsigned int).
  - כמות המסלולים שנייצר – ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0 (int).
  - בניגוד למייצר ציוצים, התוכנית לא מקבלת נתיב לקובץ קלט אלא מקבלת רק seed ומספר מסלולים, למשל:

```
>> snakes_and_ladders 3 2
```

יריץ את התכנית עם הערך 3 ל-seed ויודפסו שני מסלולים אפשריים של משחק.

- **שלב הלמידה**
  - השלב כולו מומש עבורכם. ודאו שאתם מבינים אותו וקוראים לפונקציות עם פרמטרים נכונים.
  - מימשנו עבורכם בקובץ snakes_and_ladders.c את הפונקציות create_board ו-fill_database. יש לעיין בקוד ולהבין אותו.

- הפונקציה create_board משתמשת במערך הדו-מימדי transitions כדי לייצר את הלוח. השימוש בה מחליף את הקריאה מקובץ שהיתה בחלק א'.
- מימשו עבורכם את fill_database כך שמעבר בין שני תאים מוגדר עם הלוגיקה הבאה:
  - אם נמצאים בתא שמכיל סולם אז תמיד "עולים" בסולם לתא שנמצא ב-ladder_to.
  - אם נמצאים בתא שמכיל נחש אז תמיד "גולשים" לתא בקצה הנחש שנמצא ב-snake_to.
  - אחרת, נרצה לדמות זריקת קובייה, לכן מכל תא יש אפשרות לקפוץ לאחד משש התאים העוקבים באותה הסתברות. למשל: אם נמצאים כרגע בריבוע 50, ניתן לקפוץ לאחד התאים מ-51 עד 56 באותה הסתברות.

- **<u>יצירת מסלול</u>**
  - <u>בחירת התא הראשון במסלול</u>: התא הראשון במסלול תמיד יהיה התא הראשון בלוח (ואין צורך לבחור את אחד מהתאים רנדומלית).
  - <u>בחירת התא הבא</u>: כמו בחלק א', נשתמש ב-database שיצרנו, ונבחר תא באופן רנדומלי מהתאים העוקבים של התא האחרון שבחרנו, כך שהסיכוי של כל תא עוקב להיבחר פרופורציונלי לתדירות שבה הוא מופיע.
  - מסלול מסתיים כשמגיעים לתא מספר 100 או לאחר ששיחקנו 60 סיבובים. כלומר:
    - תא 100 הוא "סוף משפט".
    - ו-60 הוא max_length.

- **<u>פלט</u>** התכנית תדפיס את המסלולים בפורמט הבא (הצבעים לא מודפסים, זה לנוחות קריאה):

Random Walk 1: [1] -> [5] -> [9] -> [11] -> [17] -> [23]-ladder to 76 -> [76] -> [77] -> [82] -> [84] -> [86] -> [92] -> [95]-snake to 67 -> [67] -> [68] -> [72] -> [75] -> [77] -> [81]-snake to 43 -> [43] -> [44] -> [47] -> [49] -> [52] -> [54] -> [59] -> [62] -> [66]-ladder to 89 -> [89] -> [95]-snake to 67 -> [67] -> [69]-snake to 32 -> [32] -> [33]-ladder to 70 -> [70] -> [75] -> [79]-ladder to 99 -> [99] -> [100]

Random Walk 2: [1] -> [3] -> [6] -> [10] -> [15]-ladder to 47 -> [47] -> [52] -> [55] -> [59] -> [65] -> [67] -> [68] -> [71] -> [73] -> [76] -> [77] -> [78] -> [84] -> [85]-snake to 17 -> [17] -> [22] -> [27] -> [31] -> [36] -> [42] -> [45] -> [46] -> [48] -> [51] -> [55] -> [56] -> [61]-snake to 14 -> [14] -> [20]-ladder to 39 -> [39] -> [43] -> [45] -> [48] -> [54] -> [56] -> [57]-ladder to 83 -> [83] -> [85]-snake to 17 -> [17] -> [18] -> [22] -> [24] -> [26] -> [29] -> [31] -> [36] -> [38] -> [43] -> [44] -> [48] -> [51] -> [57]-ladder to 83 -> [83] -> [88] -> [94] ->

<u>פירוט פורמט ההדפסה</u>:

1. כל שורה מהווה מסלול חוקי וכל מסלול מסתיים בירידת שורה.
2. כל מסלול מתחיל בטקסט "Random Walk", אחריו מספר המסלול (מ-1 ועד כמות המסלולים) ונקודתיים.
   Random Walk i:
3. כל תא שעוברים בו ייכתב בתוך סוגריים מרובעים.
4. מעברים:
   a. כל מעבר בין תאים ייכתב עם חץ בין התאים (ורווח יחיד בין שני צידי החץ).
   b. כאשר יש סולם בין תא x לתא y נדפיס: [x]-ladder to y-> [y]
   c. כאשר יש נחש בין תא x לתא y נדפיס: [x]-snake to y-> [y]

5. אם הגענו לריבוע ה-100 (ניצחון) המסלול מסתיים ב- [100].
6. אם הסתיימו 60 שלבים ולא הגענו ל-100 (כישלון) משאירים את החץ אחרי הריבוע האחרון.

קובץ snakes_and_ladders.c:

- עליכם לכתוב פונקציית main שמקבלת את הארגומנטים מה-CLI ומשתמשת בפונקציות הממומשות כדי ליצור ולמלא את ה-MarkovChain ואז ליצור ולהדפיס מסלולים אפשריים לפלט.
- שימו לב שעליכם לכתוב ולספק ל-MarkovChain מצביעים לפונקציות המתאימות לטיפוס Cell החדש, ממשו אותן בהתאם לצורך ובהתאם להוראות.

5 התאמות נוספות וחשובות

- על מנת שהטסטים יעבדו וירוצו בצורה טובה על שתי התכניות, כל פונקציה שאתם ממשים ב- tweets_generator.c ו- snakes_and_ladders.c צריכה להיות סטטית. (יש להוסיף את המילה static לפני השם של הפונקציה)
- הפונקציה fill_database אמורה להחזיר int ולא void, נא תקנו את החתימה אצלכם בקוד:

```
static int fill_database(MarkovChain *markov_chain)
```

- החתימה של הפונקציה add_node_to_database בקובץ markov_chain.h השתנתה ל-

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

6 קובץ makefile

הסבר כללי:

בחלק זה של התרגיל נתרגל שימוש בסיסי ב-Make. Make היא תוכנה לניהול אוטומטי של קומפילציית קוד, והיא חלק מפרויקט התוכנה החופשית GNU. כדי להשתמש ב-Make ניצור קובץ טקסט בשם makefile בו יכתבו ההוראות לקומפילציה, כאשר הפורמט הבסיסי להוראה הינו:

```
target_name: dependencies
	commands
```

כאשר name_target הוא שם כלשהו (לבחירתכם), במקום dependencies נשים את קובץ הקוד שנרצה לקמפל, או שם של target נוסף שעבורו גם מוגדרות הוראות קומפילציה, ואת commands נחליף בפקודת הקומפילציה (אותה פקודה שהיינו כותבים בטרמינל).

לאחר שהגדרנו את הוראות הקומפילציה עבור target_name, נוכל להריץ בטרמינל את הפקודה `make target_name` , ו-make יריץ את commands והקומפילציה תתבצע.

בתרגיל זה:

בתרגיל זה עליכם להגיש makefile שיכיל שני target שונים, אחד לציוצים ואחד לסולמות ונחשים:

1. בהרצת הפקודה make tweets בטרמינל (בתיקיה עם קבצי הקוד), יווצר קובץ מקומפל אותו נוכל להריץ, למשל כך: ./tweets_generator 123 2 "justdoit_tweets.txt".
2. הפקודה make snake תייצר קובץ מקומפל שניתן להריצו: ./snakes_and_ladders 3 2.

מומלץ ליצור את הקובץ make עם ה-targets הנ"ל בתחילת העבודה על התרגיל, כך תוכלו להשתמש בפקודות הנ"ל בשביל לקמפל בקלות את הקבצים השונים בהתאם לתוכנית אותה אתם מעוניינים להריץ.

7 פתרון בית-ספר ו-presubmit

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

```
~proglab/presubmit/ex3b/run
```

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

```
~proglab/school_solution/ex3b/schoolSolution <prog> <arguments>
```

שימו לב! בגלל שבחלק הזה יש שתי תוכניות צריך להגדיר ל-school_solution איזו מהן להריץ, כך ש- <prog> יכיל tweets אם נרצה להריץ את tweets_generator או יכיל את snake אם נרצה להריץ את הסולמות והנחשים. את שאר הארגומטים מוסיפים אחרי כרגיל, למשל:

```
~proglab/school_solution/ex3b/schoolSolution snakes 3 2
```

הערה: 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.

8 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 להדפיס הודעת שגיאה מתאימה ל-stdout המתחילה ב- "Allocation failure:", לשחרר את כל הזיכרון שהוקצה עד כה בתכנית, ולצאת מהתוכנית עם EXIT_FAILURE. כרגיל, אין להשתמש ב-exit().
- אם אפשרי, תעדיפו תמיד לעבוד עם int/long מאשר float/double. ניתן לפתור את התרגיל כולו בעזרת שימוש במספרים שלמים בלבד.
- אין להשתמש ב-vla, כלומר מערך במחסנית שגודלו נקבע ע"י משתנה. שימוש שכזה יגרור הורדת ציון.

9 נהלי הגשה

- תרגיל זה הינו התרגיל המסכם של שפת C. יש לתרגיל שני חלקים, החלק הראשון מהווה הכנה לחלק השני. קובץ זה מהווה הוראות לחלק השני של התרגיל. אנו לא ממליצים להתחיל לממש את החלק השני לפני שאתם עוברים את ה-presubmit של החלק הראשון.
- קראו בקפידה את הוראות חלק זה של התרגיל. זהו תרגיל מורכב ולכן אנו 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.

- בחלק זה של התרגיל תהיה בדיקה אוטומטית. כחלק מהבדיקה האוטומטית תיבדקו על סגנון כתיבה.
- התרגיל נבדק על מחשבי האוניברסיטה, ולכן עליכם לבדוק כי הפתרון שלכם רץ ועובד גם במחשבים אלו.
- כשלון בקומפילציה או ב-presubmit יגרור ציון 0 בתרגיל.
- נזכיר כי חלק זה של התרגיל מהווה 75% מהציון הסופי של התרגיל.