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' 10 ליולי ב-22:00

נושאי התרגיל: #מצביעים #מצביעים_לפונקציות #תכנות_גנרי

#ניהול_זיכרון#רשימה מקושרת#סטראקטים

תרגיל 3 חלק ב - תכנות גנרי

<u>הקלות למשרתי מילואים</u>:
מדובר בתרגיל מסכם ולכן יעוגל הציון ל-100 **רק עבור סטודנטים בקטגוריות 2-3 שעברו פריסבמיט בסיסי בהצלחה.**
הטסטים אותם משרתי מילואים בקטגוריות שהוזכרו נדרשים לעבור מפורטים בנספח בסוף המסמך.

<u>תיאור התרגיל</u>:
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 הספרייה markov_chain כך שנוכל ליצור שרשראות של טיפוסים שונים (ולא רק tweets / שרשראות של מחרוזות). בנוסף לעדכון הספרייה והפיכתה לגנרית, תכתבו גם שני קבצים שהולכים להשתמש בספרייה ולהדגים את הגנריות שלה (כלומר כל קובץ יעשה שימוש בספרייה על מנת לייצר שרשראות של טיפוסים שונים).
הקבצים הינם: tweets_generator (מטרתו דומה לקובץ שכתבתם בחלק א' אולם הפעם תעשו שימוש בספרייה הגנרית שיצרתם) ו-snakes_and_ladders (קובץ חדש עליו יפורט בהמשך).

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

שימו לב, כדי לוודא שהספרייה markov_chain ממומשת באופן גנרי לחלוטין, הטסטים האוטומטיים של בית הספר ירוצו גם על **סטראקטים (סטראקט ברבים) שאתם לא מכירים**.

1. קבצים

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

- justdoit_tweets.txt – קובץ קלט לדוגמא עבור התכנית tweets_generator.

- markov_chain.h – מכיל את השלד של הסטראקטים **המעודכנים** (שימו לב שאינכם משתמשים בקובץ שסופק עבור חלק א')
- markov_chain.c – בו תממשו את הפונקציות המוגדרות ב-markov_chain.h
- linked_List.h linked_List.c – מימוש רשימה מקושרת לשימושכם. **קבצים אלה אינם להגשה ולכן אין לשנות אותם!**
- tweets_generator.c – בו תממשו תכנית דומה לחלק א של התרגיל העושה שימוש בספרייה הגנרית.
- snakes_and_ladders.c – קובץ (ממומש חלקית) המשתמש בספריית מרקוב שתכתבו.
- makefile – **קובץ הנועד לסייע לכם עם תהליך הקימפול של התרגיל, אין לעשות בו שינויים ואין להגיש אותו!** (מצורף הסבר על אופן השימוש בו בסוף המסמך)

אתם צריכים להגיש:

- markov_chain.h – מעודכן עם הסטראקטים שתכתבו (שימו לב שאינכם משנים את שמות הסטראקטים בקובץ שקיבלתם)
- markov_chain.c – מכיל את המימוש שלכם לפונקציות אשר נמצאות ב-markov_chain.h (כמובן שניתן לממש פונקציות עזר נוספות משלכם)
- tweets_generator.c – מטרתו זהה לקובץ שהוגש בחלק א', אך עם שינויים כך שיתאים לספרייה הגנרית החדשה.
- snakes_and_ladders.c – קובץ המשתמש בספריית מרקוב שכתבתם, הוראות המימוש של הקובץ מפורטות בהמשך.

2. מבני נתונים

בחלק זה של התרגיל תעדכנו את ספריית מרקוב שכתבתם בחלק א' (המכילה מבנה נתונים שמתאר מחרוזות) כך שמבנה הנתונים יתאר דאטא גנרי.

השינויים יתבטאו בהגדרת ה-structs בקובץ markov_chain.h **וכן באופן המימוש של פונקציות הספרייה** שהוגדרו בחלק א' של התרגיל.

בפרט, עליכם להגדיר מחדש את ה-structs הבאים בקבצים markov_chain.h כך שיתאימו למבנה נתונים גנרי:

- הסטראקטים החדשים יהיו זהים לסטראקים שהגדרתם בחלק א' של התרגיל מלבד השינויים הבאים:
- **<u>MarkovNode</u>**
  - השדה data יהפוך להיות מצביע לדאטא **גנרי**.
  - frequency_list – **ללא שינוי**.
  - ניתן להוסיף ל-struct זה משתנים כרצונכם.
- **<u>MarkovNodeFrequency</u>**
  - **ללא שינוי**.
- **<u>MarkovChain</u>**
  - כיוון שמבנה הנתונים מתעסק בדאטא גנרי שאין לו עליו ידע מוקדם, הוא נדרש להחזיק במצביעים לפונקציות שיעזרו לו לעבוד עם הדאטא מבלי להניח עליו הנחות (שימו לב, הספרייה עצמה אינה יודעת את סוג הדאטא, אולם המשתמש כן יודע ולכן **המשתמש הוא זה שמממש את הפונקציות** ומעביר מצביעים אליהן לספרייה).
  - יש להוסיף שדות המכילים **מצביעים לפונקציות גנריות (לא לממש אותם)**:

- **print_func** – מצביע לפונקציה המקבלת מצביע לטיפוס גנרי, לא מחזירה כלום ומדפיסה את הדאטא במצביע.
- **comp_func** – מצביע לפונקציה המקבלת שני מצביעים לדאטא גנרי ומחזירה:
  - ערך חיובי אם הדאטא במצביע הראשון גדול ממש מהדאטא במצביע השני.
  - ערך שלילי אם הדאטא במצביע השני גדול ממש מהראשון.
  - 0 אם הדאטא בשני המצביעים שווה.
- **free_data** – מצביע לפונקציה המקבלת מצביע מטיפוס גנרי, לא מחזירה כלום ומשחררת את הזיכרון של המשתנה שהיא קיבלה.
- **copy_func** – מצביע לפונקציה המקבלת מצביע מטיפוס גנרי ומחזירה מצביע שהוא העתק בלתי תלוי של המצביע שקיבלה. על ההעתק להיות מוקצה דינמית.
- **is_last** – מצביע לפונקציה שמקבלת מצביע מטיפוס גנרי ומחזירה ערך בוליאני לפי הפירוט הבא:
  - true – אם הדאטא אליו מצביע המצביע אמור להיות איבר אחרון בשרשרת שנייצר (למשל עבור ציוצים – האם המחרוזת נגמרת בנקודה). שימו לב **שמדובר באיבר שמופיע אחרון בשרשרת שנייצר ולא במבנה הנתונים שלנו.**
  - false - אם הדאטא אליו מצביע המצביע **לא** יכול להיות איבר אחרון בשרשרת שנייצר (למשל עבור ציוצים, אם המחרוזת לא מסתיימת בנקודה, ולכן לא מסיימת ציוץ).
- **אין להוסיף משתנים נוספים ל-struct זה.**

- **הערות:**
  - **מומלץ להשתמש ב-**typedef **כדי ליצור טיפוס חדש של מצביע לפונקציה.**

    למשל: typedef bool (*is_even_func_ptr)(int); המגדיר טיפוס בשם is_even_func_ptr של מצביע לפונקציה שמקבלת int ומחזירה bool.
  - בקובץ markov_chain.h יש את השלד של הסטראקט, **אסור** לשנות את השמות של המשתנים **ואסור** להוסיף עוד משתנים ל-**MarkovChain** (הנ"ל כדי שנוכל להריץ טסטים, שימו לב שלסטראקטים אחרים שאינם MarkovChain כן ניתן להוסיף משתנים נוספים).

שינויים במימוש פונקציות הספרייה ב-markov_chain.c:

בנוסף לשינויים בסטראקטים שהוזכרו לעיל, יש לשנות את המימוש של פונקציות הספרייה כך שיותאמו למבנה הנתונים הגנרי. יש לשם לב לנקודות הבאו:

1. בכל פעם שהמשתמש עושה שימוש בספריית מרקוב על מנת לייצר אובייקט מסוג MarkovChain הוא אחראי להעביר לה מצביעים לפונקציות שהוזכרו (ממומשות על ידו או פונקציות של הספרייה הסטנדרטית). הפונקציות שהמשתמש יעביר אמנם מקבלות מצביע גנרי אך הן ממומשות באופן שמותאם לטיפוס ספציפי (אותו הטיפוס של המצביע data ב-MarkovNode). **יש לעדכן את הקוד ב-markov_chain.c כך שהספרייה תעשה שימוש בפונקציות שהועברו** על מנת להתעסק עם הדאטא מבלי לדעת במפורש מאיזה טיפוס הוא ומבלי להניח עליו הנחות (למשל להדפיס אותו, להשוות בין מצביעים שונים, לשחרר את המצביעים וכו').
2. שימוש לדוגמא בפונקציות שהועברו הוא לעשות שימוש בפונקציה comp_func בשביל להשוות דאטא במקום להשתמש בפונקציה strcmp המניחה שמדובר במחרוזות.

3. בנוסף, **שימו לב שהחתימות של הפונקציות שמימשתם בקובץ markov_chain.c תואמות את אלו שמופיעות בקובץ markov_chain.h שקיבלתם**. בפרט (מלבד החלפת char* ב-void* בפונקציות בהן היה צורך), **<u>בהשוואה לחלק א' של התרגיל, שתי הפונקציות הבאות עודכנו:</u>**
   a. הפונקציה generate_tweet הוחלפה בפונקציה generate_random_sequence והתווסף לה ארגומנט נוסף מסוג MarkovChain*.
   b. לפונקציה add_node_to_frequency_list התווסף ארגומנט נוסף מסוג MarkovChain*.

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

```
int add_node_to_frequency_list(MarkovNode *first_node, MarkovNode
*second_node, MarkovChain *markov_chain);
```

3. שימוש בספריית markov_chain על ידי tweets_generator

בחלק זה של התרגיל תדגימו את השימוש בספרייה הגנרית על מנת לייצר מבנה נתונים השומר **מחרוזות** ובעזרתו ניתן לייצר ציוצים.

- מטרת התכנית tweets_generator שתממשו זהה במטרתה לתכנית שמימשת בחלק א' של התרגיל (ייצור ציוצים חדשים על פי למידה של קובץ טקסט).
- על הקלט והפלט של התכנית להיות זהה למפורט בהנחיות של חלק א' של התרגיל (כולל הפלט במקרה של העברת פרמטרים לא תקינים או שגיאות אחרות) אולם המימוש צריך להשתמש בספרייה הגנרית החדשה שכתבתם.
- שלב הלמידה ושלב יצירת הציוצים זהים <u>מבחינה לוגית</u> לשלבים אלו בחלק א', כלומר הלוגיקה נשארת זהה אך הקוד צריך להתאים לעדכונים בספריית markov_chain.
- כדי 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 שמבנה הנתונים יעבוד על מחרוזות (טיפ: חלק מהפונקציות קיימות ב- <string.h> ולא צריך לממש אותן מחדש).
- מומלץ להשתמש בקובץ justdoit_tweets.txt כדי להריץ את התוכנית לוודא נכונות. מימוש נכון של הספרייה והקובץ tweets_generator.c יוביל לתוצאות הזהות לאלו של חלק א'.

4. שימוש בספריית markov_chain על ידי snakes_and_ladders

כדי לבדוק את הגנריות של ספריית מרקוב שכתבתם, נרצה לבדוק אותה על טיפוס נוסף (כלומר לייצור מבנה נתונים ששומר משתנים שאינם מחרוזות). ספציפית, אנו נדמה משחק <u>סולמות ונחשים</u> ונייצר שרשראות המייצגות מסלולים אפשריים של שחקן על הלוח.

בדומה לתכנית tweets_generator, גם את התכנית הזאת ניתן לחלק לשלב למידה ולשלב השימוש.

מטרתו של שלב הלמידה הוא לבנות את מבנה הנתונים ובשלב השימוש נעזר במבנה הנתונים המלא על מנת לייצר מסלולים על הלוח.

שלב הלמידה

שלב הלמידה ממומש כולו עבורכם בקובץ snakes_and_ladders.c ואין צורך לשנות אותו. מומלץ מאוד לעבור על הקוד הממומש ולוודא שאתם מבינים אותו ויודעים להעביר לפונקציות את הפרמטרים המתאימים. להלן הסבר קצר על המימוש (הנ"ל אינו תחליף למעבר עצמאי על הקוד):

- מוגדר עבורכם הסטראקט cell המייצג משבצת (תא) בלוח המשחק.

```
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 square
    int snake_to;  // snake_to represents the jump of the snake in case there is one
from this square
    //both ladder_to and snake_to should be -1 if the Cell doesn't have them
} Cell;
```

- מוגדרת הפונקציה create_board המקבלת מערך של מצביעים ל-cell בגודל לוח המשחק ומאתחלת את הלוח (מקצה זיכרון לכל התאים וממלאת את הלוח בסולמות ונחשים ע"פ המערך הדו ממדי transitions). שימו לב, כל המידע המקדים על המשחק נקבע לפי גודל הלוח והמערך transitions ולכן בשונה מ-tweets_generator, כאן אין קריאה מקובץ.

```
int create_board(Cell *cells[BOARD_SIZE])
```

- מוגדת הפונקציה fill_database המקבלת מצביע ל-MarkovChain ובונה את מבנה הנתונים על פי הלוגיקה הבאה:
  - אם נמצאים ב-cell המכיל סולם אז תמיד (הסתברות 1) "עולים" בסולם (כלומר ה-cell הבא יהיה ה-cell המוגדר בשדה ladder_to). הנ"ל שקול ל-frequency_list בעלת שורה אחת בלבד.
  - באופן דומה, אם נמצאים ב-cell המכיל נחש אז תמיד (הסתברות 1) "יורדים" בנחש (כלומר ה-cell הבא יהיה ה-cell המוגדר בשדה snake_to).
  - אחרת, נרצה לדמות הטלת קובייה, כלומר מכל תא ניתן להתקדם לאחד מבין ששת התאים העוקבים לו. למשל מתא מספר 50 ניתן לעבור לאחד מבין התאים 51 עד 56 כולל בהתפלגות אחידה. הנ"ל שקול ל-frequency_list הבאה:

Cell	frequency
51	1
52	1
53	1
54	1

55	1
56	1

- במקרה שנמצאים בטווח התאים 95-99, רשימת התדירויות תכיל את התאים עד מספר 100 בלבד בהסתברויות המתפלגות אחיד (לא ניתן לחרוג מעבר לתא מספר 100)

```
int fill_database_snakes(MarkovChain *markov_chain)
```

<u>הנחות על שלב הלמידה</u>:

- הלוח בגודל 100, מתחיל בתא מספר 1 ומסתיים בתא מספר 100 (כולל).
- נניח שהמשחק מכיל שחקן יחיד.
- ניתן להניח שלא קיים תא המכיל גם סולם וגם נחש.
- ניתן להניח שלא קיים סולם וכן לא קיים נחש בתא מספר 100 (התא האחרון בלוח).

<u>שימוש במבנה הנתונים ליצירת מסלולים</u>

- אנו נגדיר "משחק" בתור מסלול חוקי על הלוח, כלומר רצף חוקי של תאים (מקביל ל"ציוץ" המוגדר כרצף מחרוזות).
- כל משחק מתחיל בתא מספר 1 ומסתיים בתא מספר 100 או לאחר 60 משבצות (max_length=60).
- יצירת מסלול מתבצעת ע"י:
  - בחירת התא הראשון (תמיד תא מספר 1, אין צורך בבחירה רנדומלית).
  - הגרלת התא הבא במסלול לפי מבנה הנתונים.
  - מסלול מסתיים כאשר מגיעים לתא מספר 100 (בדומה לכך שציוץ נגמר בנקודה) או לאחר 60 צעדים.

<u>התכנית עצמה:</u>

עליכם לכתוב פונקציית main שמקבלת ארגומנטים מה-CLI ומשתמשת בספריית מרקוב והפונקציות שמומשו עבורכם בקובץ snakes_and_ladders.c כדי ליצור ולמלא את ה-MarkovChain ואז ליצור בעזרתו ולהדפיס מסלולים אפשריים.

שימו לב, עליכם לכתוב ולספק ל-MarkovChain מצביעים לפונקציות המתאימות לטיפוס Cell החדש, ממשו אותן בהתאם לצורך ובהתאם להוראות.

<u>קלט:</u>

- ערך **seed** – מספר שיינתן לפונקציית ה-()srand פעם אחת בתחילת ריצת התוכנית. ניתן להניח כי הוא מספר שלם אי שלילי (unsigned int).
- כמות המסלולים שנרצה לייצר – ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0 (int).

בניגוד למייצר הציוצים, התוכנית לא מקבלת נתיב לקובץ קלט אלא מקבלת רק seed ומספר מסלולים.

למשל, הפקודה:

```
snakes_and_ladders 3 2
```

תריץ את התוכנית עם הערך $seed = 3$ ותדפיס שני מסלולים אפשריים של משחק.

<u>פלט</u>:

<u>פירוט פורמט ההדפסה</u>:

1. כל מסלול צריך להסתיים בירידת שורה.
2. כל מסלול מתחיל בטקסט "Random Walk", לאחר מכן יופיע רווח יחיד ומספר המסלול (מ-1 ועד כמות המסלולים שהתקבלה כפרמטר) ולאחר מכן נקודתיים ורווח נוסף. לדוגמא – " :Randon Walk i"
3. כל תא שהמסלול עובר בו יכתב בסוגריים מרובעים (ללא רווחים בתוך הסוגריים).
4. מעבר רגיל בין תאים יכתב ע"י חץ ורווח יחיד בכל צד של החץ: " -> "
5. במקרה של נחש נסמן: " -snake to-> " עם רווח יחיד בתחילת ובסיום החץ.
6. במקרה של סולם נסמן: " -ladder to-> " עם רווח יחיד בתחילת ובסיום החץ.
7. אם הגענו לתא 100, המסלול יסתיים ב-[100] (ולאחריו ירידת שורה כמו בסיום כל מסלול).
8. אם המסלול הגיע לאורך המקסימלי (max_length=60) מבלי להגיע לתא 100, יש להדפיס את החץ לאחר התא האחרון במסלול (יחד עם הרווח שמופיע תמיד אחרי כל חץ) ולאחר מכן לרדת שורה (כמו בסיום כל מסלול).

להלן דוגמא לפלט תקין של התכנית: (ירידות השורה המוצגות באמצע המסלול הן תוצאה של היעדר מקום במסך ולא של הדפסה של ירידת שורה)

```
Random Walk 1: [1] -> [2] -> [4] -> [7] -> [13] -snake to-> [4] -> [5] -> [6] -> [10] -> [12] -> [16] -> [21] -> [25] -> [29] -> [35]
-snake to-> [11] -> [15] -ladder to-> [47] -> [51] -> [55] -> [58] -> [61] -snake to-> [14] -> [20] -ladder to-> [39] -> [41] -ladder
to-> [62] -> [65] -> [68] -> [69] -snake to-> [32] -> [36] -> [37] -> [42] -> [43] -> [49] -> [50] -> [55] -> [59] -> [63] -> [69] -sn
ake to-> [32] -> [38] -> [39] -> [42] -> [47] -> [53] -> [54] -> [55] -> [56] -> [58] -> [61] -snake to-> [14] -> [16] -> [17] -> [18]
 -> [21] -> [27] -> [29] -> [32] ->
Random Walk 2: [1] -> [7] -> [12] -> [13] -snake to-> [4] -> [6] -> [11] -> [12] -> [14] -> [17] -> [20] -ladder to-> [39] -> [45] ->
[47] -> [50] -> [56] -> [60] -> [61] -snake to-> [14] -> [16] -> [22] -> [23] -ladder to-> [76] -> [82] -> [84] -> [87] -snake to-> [3
1] -> [32] -> [36] -> [39] -> [45] -> [51] -> [57] -ladder to-> [83] -> [87] -snake to-> [31] -> [34] -> [36] -> [38] -> [40] -> [42]
-> [46] -> [51] -> [52] -> [57] -ladder to-> [83] -> [87] -snake to-> [31] -> [32] -> [38] -> [44] -> [48] -> [49] -> [54] -> [58] ->
[64] -> [69] -snake to-> [32] -> [36] -> [38] ->
Random Walk 3: [1] -> [5] -> [8] -ladder to-> [30] -> [31] -> [36] -> [39] -> [41] -ladder to-> [62] -> [65] -> [68] -> [71] -> [73] -
> [77] -> [82] -> [85] -snake to-> [17] -> [22] -> [28] -ladder to-> [50] -> [53] -> [57] -ladder to-> [83] -> [84] -> [88] -> [94] ->
 [98] -> [99] -> [100]
```

במקרה שבו התקבל מספר פרמטרים שאינו תקין, יש להדפיס הודעה אינפורמטיבית ל-**stdout** המפרטת בקצרה את הפרמטרים הנדרשים לתכנית. על ההודעה להתחיל ב-" Usage:" (עם הקפדה על lower\upper case וכן רווח אחרי הנקודתיים, הודעת שגיאה בפורמט הנוכן מוגדרת עבורכם כ-macro בקובץ snakes_and_ladders.c).

5. התאמות נוספות

- על מנת שהטסטים ירוצו כמו שצריך על התרגיל **<u>אתם נדרשים לוודא שלא מוגדרות בתרגיל שתי פונקציות בעלות אותו שם</u>** (למשל אם הגדרתם פונקציה בשם comp_func בקובץ tweets_generator.c אסור לכם לייצור פונקציה בשם הנ"ל בקובץ snakes_and_ladders.c).

6. קובץ makefile

בתרגיל הנוכחי אתם מגישים שני קבצים נפרדים שבכל אחד מהם ממומשת פונקציית main. נזכיר כי לא ניתן לקמפל שתי פונקציות main יחד לכן בכל פעם שתריצו את התרגיל שלכם תידרשו לכלול לרשימת הקבצים לקמפול או את tweets_generator.c או את snakes_and_ladders.c.

על מנת להקל עליכם, סיפקנו עבורכם את הקובץ makefile שיעזור לנהל את תהליך הקמפול.

להלן אופן השימוש ב-makefile:

1. בטרמינל (בתיקייה עם שאר קבצי הקוד) – הפקודה "make tweets_generator" תייצר קובץ מקומפל בשם tweets_generator אותו ניתן להריץ למשל באופן הבא:

   "./tweets_generator 123 2 justdoit_tweets.txt"

   הפקודה make tweets_generator שקולה לקמפול דרך הטרמינל בעזרת הפקודה:

   gcc tweets_generator.c markov_chain.c linked_List.c -o tweets_generator

2. באופן דומה – הפקודה "make snakes_and_ladders" תייצר קובץ מקומפל בשם snakes_and_ladders אותו ניתן להריץ למשל באופן הבא:

   "./snakes_and_ladders 2 3"

3. על מנת למחוק את הקבצים המקומפלים שייצרתם (למשל לפני קמפול מחדש), ניתן להשתמש בפקודה "make clean".
4. **הקובץ נועד לסייע לכם בלבד ואין שום חובה להשתמש בו**. ניתן לבצע את הליך הקמפול גם ללא שימוש ב-makefile.

7. פתרון בית-ספר ו-presubmit

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

```
~labcc2/presubmit/ex3b/run
```

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

```
~labcc2/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_generator אם נרצה להריץ את tweets_generator או יכיל את snakes_and_ladders אם נרצה להריץ את הסולמות והנחשים. את שאר הארגומנטים מוסיפים אחרי כרגיל, למשל:

```
~labcc2/school_solution/ex3b/schoolSolution snakes_and_ladders 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 שנגרמה 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 השמור במחסנית שגודלו נקבע ע"י משתנה. שימוש שכזה יגרור הורדת ציון.
- **יש להקפיד על פורמט ההדפסה המפורט לעיל, הן בפלט התקין והן בהודעות השגיאה!** לא יתקבלו ערעורים על נפילה של טסטים עקב פורמט הדפסה שגוי.
- **נדגיש שמעבר של ה-presubmit אינו מבטיח קבלת ציון מושלם** (מלבד משרתי המילואים שהוזכרו), התרגיל יבדק גם על טסטים נוספים. בנוסף, בדיקות ה-presubmit אינן כוללות בדיקה באמצעות valgrind ולכן **תיתכן הורדה של נקודות בציון הסופי גם על טסטים שעברו בבדיקת ה-presubmit.**

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 ממליצים שלא 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.

- כחלק מהבדיקה תבדקו על סגנון כתיבה.
- מכיוון שהתרגיל נבדק על מחשבי האוניברסיטה, 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. וכן חובה לעבור את בדיקות ה-presubmit בהצלחה על מנת לעבור את הקורס!**
- **מועד הגשה של חלק זה:** יום ד' 10 ליולי ב-22:00 **בונוס 5+:** הגשה עד יום ד׳ 3 ליולי ב-22:00
  בונוס של 1+/2+/3+ נקודות: הגשה יום/יומיים/שלושה ימים מראש (כרגיל)

<u>נספח – טסטים למשרתי מילואים</u>:

להלן רשימת הטסטים אותם נדרשים לעבור משרתי מילואים בקטגוריות 2-3 על מנת שהציון יתעגל ל-100 (בנוסף לקמפול מוצלח של התרגיל).

השוואה של פלט הכתנית לפתרון בית ספר:

- TestComparison_2
- TestComparison_4
- TestComparison_12

מימוש הפונקציות המפורטות בתיאור התרגיל:

- TestAST_1
- TestAST_2
- TestAST_3
- TestAST_4
- TestAST_5
- TestAST_6
- TestAST_7