סדנת תכנות בשפת C ו-++C
תרגיל 4

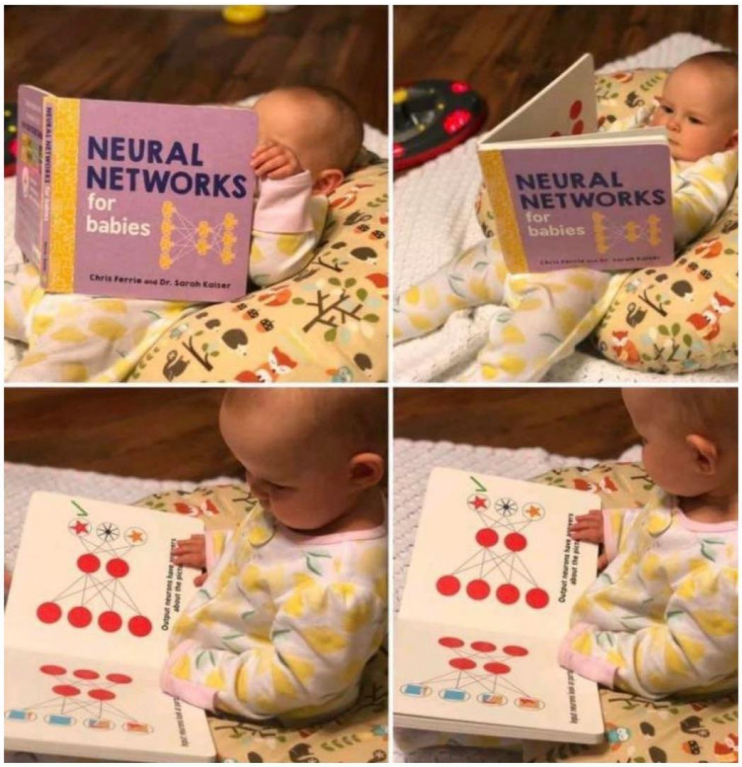
תאריך ההגשה של התרגיל: יום רביעי, ה-28 לדצמבר, 2022 – עד השעה 23:59

נושאי התרגיל: Classes, Operator Overloading, References, Rule of Three, Exceptions

אנא הקפידו לקרוא את כל התרגיל מתחילתו ועד סופו לפני שתגשו לממשו.

1 רקע

בתרגיל זה נכתוב תוכנה לזיהוי ספרות הנכתבות בכתב יד. התוכנה שלנו תקבל כקלט תמונה של ספרה בין 0 ל-9 ותחזיר כפלט את הספרה אשר זוהתה. נעשה זאת על ידי בניית מודל של רשת נוירונים. הרשת שנריץ תגיע לדיוק של כ- 96 אחוזים בזיהוי ספרות.



1.1 הקדמה

רשת נוירונים היא מודל בלמידת מכונה המבוסס על מבנה המוח האנושי: נוירון מקבל גירוי חשמלי מנוירונים אחרים - אם הגירוי הזה עובר סף מסוים הוא שולח בעצמו אות אל נוירונים אחרים. המוח מורכב ממספר רב של נוירונים המקושרים זה לזה ברשת מורכבת, ויחד הם מסוגלים לבצע את הפעולות הנדרשות ממנו. רשת נוירונים מלאכותית (Artificial neural network) פועלת באופן דומה. ברשתות אלו נעשה שימוש בזיהוי ומיקום של עצמים בתמונה, הבנת שפה אנושית וניתוחה, יצירת טקסט ועוד. מוצרים רבים בחיינו משתמשים ברשתות נוירונים: עוזרים קוליים (Amazon Alexa, Apple Siri), השלמה אוטומטית לתוכן המייל ב-Gmail, זיהוי מחלות

בתמונות סריקה רפואית ועוד.שימו לב: למידת מכונה בכלל, ורשתות נוירונים בפרט, הינם נושאים רחבים ומורכבים ולכן לא יכללו בתוכן תרגיל זה. לצורך מימוש התרגיל אין צורך להבין איך ולמה עובדת רשת הנוירונים. הרקע התיאורטי הנדרש יוצג בסעיף 1.2, פרטי הרשת שנבנה יובאו בסעיף 2.2, והמחלקות למימוש יובאו בסעיף 3. מומלץ לצפות בסרטון הבא המפרט על המבנה של רשת נוירונים וכיצד ניתן לממש אותה באמצעות אלגברה לינארית: https://www.youtube.com/watch?v=aircAruvnKkm

1.2 רקע תיאורטי

1.2.1 רשת נוירונים Fully Connected

- רשת בנויה משכבות (סעיף 1.2.2).
- הקלט של כל שכבה הוא וקטור, והפלט הוא וקטור אחר.
- הפלט של כל שכבה הוא הקלט של השכבה הבאה.
- קלט הרשת הוא וקטור המייצג את האובייקט שהרשת תעבד. ברשת שלנו, הוא מייצג תמונה של ספרה (סעיף 2.2.2).
- פלט הרשת הוא וקטור המייצג את המסקנה של הרשת. ברשת שלנו, הוא מייצג את הספרה שהרשת זיהתה (סעיף 2.2.3).

1.2.2 שכבה ברשת

כל שכבה ברשת מקבלת וקטור קלט $x \in \mathbb{R}^m$ ומחזירה וקטור פלט $y \in \mathbb{R}^n$ באמצעות הפעולה המתמטית $y = f\left(W \cdot x + b\right)$ כאשר:

- $y = f\left(W \cdot x + b\right)$ מטריצה שאיבריה נקראים המשקולות של השכבה (Weight).
- $b \in \mathbb{R}^n$ וקטור שנקרא ההיסט (Bias) של השכבה.
- $f : \mathbb{R}^n \to \mathbb{R}^n$ פונקציית האקטיבציה של השכבה (סעיף 1.2.3).

כלומר, בהינתן וקטור קלט $x \in \mathbb{R}^m$, הוקטור $y = f(W \cdot x + b) \in \mathbb{R}^n$ יהיה הפלט של השכבה.

1.2.3 פונקציית אקטיבציה

פונקציה $f : \mathbb{R}^n \to \mathbb{R}^n$ אשר מחזירה את התוצאה הסופית של שכבה ברשת הנוירונים. פונקציה זו אינה לינארית. בתרגיל נממש שתי פונקציות אקטיבציה שונות:

- פונקציית ReLU

$$\forall x \in \mathbb{R} \qquad \text{relu}(x) = \begin{cases} x & x \geq 0 \\ 0 & else \end{cases}$$

כאשר הפונקציה פועלת על וקטור $x \in \mathbb{R}^n$ היא מבצעת את הפעולה הזו על כל קואורדינטה בנפרד.

- פונקציית Softmax

$$\forall x \in \mathbb{R}^n \qquad \text{softmax}(x) = \frac{1}{\sum_{k=1}^{n} e^{x_k}} \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ e^{x_3} \\ \vdots \\ e^{x_n} \end{bmatrix} \in \mathbb{R}^n$$

x_k – הקוארדינטה ה-k של $x \in \mathbb{R}^n$. e^t – הפונקציה המעריכית exp(t). לצורך החישוב ניתן להשתמש בפונקציה std::exp המיובאת מ-cmath. הפונקציה מקבלת וקטור $x \in \mathbb{R}^n$ וממירה אותו לוקטור התפלגות (וקטור שאיבריו הם מספרים אי-שליליים שסכומם 1) באופן שתואם את הפלט הסופי של הרשת שלנו. בתרגיל זה וקטור הוא מטריצה עם טור אחד.

שימו לב שניתן להפעיל את פונקציות האקטיבציה גם על מטריצות. במקרה זה, הפעילו את האקטיבציה על כל איברי המטריצה. לדוגמה:

$$\text{softmax}\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\right) = \frac{1}{\sum_{i=1}^{4} e^i}\begin{bmatrix}e^1 & e^2\\ e^3 & e^4\end{bmatrix}$$

2 מימוש הרשת

2.1 שימוש ב-float ודיוק הרשת

איברי המטריצה שנממש יהיו מטיפוס float (32-bit). מאופן המימוש של float במעבד, פעולות האריתמטיקה אינן בהכרח אסוציאטיביות, כלומר לא בהכרח יתקיים: $(a+b)+c = a+(b+c)$. לכן, כדי להימנע משגיאות נומריות בעת ביצוע כפל מטריצות, אנא ממשו את סדר הפעולות לפי ההגדרה המתמטית שלמדתם בלינארית 1:

$$(A \cdot B)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

הרשת שנריץ מגיעה לכ-96 אחוזי דיוק. לכן, הרשת עלולה לטעות בחלק מהתמונות שתזינו לה – זוהי התנהגות תקינה של התוכנה. גם אם אחוזי ההצלחה של הרשת שלכם נמוכים במעט מ-96 אחוזים, ייתכן כי הדבר נובע משגיאות נומריות, ולא צפויה הורדה של נקודות במקרה זה. עם זאת, אם אחוזי ההצלחה של הרשת נמוכים משמעותית מרף זה, ייתכן שהדבר נובע מטעות במימוש.

2.2 תיאור הרשת

2.2.1 שכבות הרשת

- הרשת מורכבת מ- 4 שכבות:

שכבה	משקולות -Weights	היסט - Bias	פונקציית אקטיבציה
1	$W_1 \in M_{128\times784}$	$b_1 \in \mathbb{R}^{128}$	Relu
2	$W_2 \in M_{64\times128}$	$b_2 \in \mathbb{R}^{64}$	Relu
3	$W_3 \in M_{20\times64}$	$b_3 \in \mathbb{R}^{20}$	Relu
4 (מוצא)	$W_4 \in M_{10\times20}$	$b_4 \in \mathbb{R}^{10}$	Softmax

- כלומר, על מנת לממש את הרשת עלינו לשרשר את רצף הפעולות הבא:

$$\begin{aligned} r_1 &= Relu(W_1 \cdot x + b_1)\\ r_2 &= Relu(W_2 \cdot r_1 + b_2)\\ r_3 &= Relu(W_3 \cdot r_2 + b_3)\\ r_4 &= Softmax(W_4 \cdot r_3 + b_4) \end{aligned}$$

- x – וקטור הקלט לרשת (סעיף 2.2.2).

- r_i – הפלט של השכבה ה-i, שהוא גם הקלט לשכבה ה-$i+1$.
- r_4 – הפלט של השכבה הרביעית, שהוא וקטור הפלט של הרשת (סעיף 2.2.3).

2.2.2 וקטור הקלט

- כל תמונה מקודדת בתור מטריצה A בגודל 28×28 של פיקסלים בגווני אפור (Grayscale), וכל מספר במטריצה הוא ערך בין 0 ל-1, כלומר $A \in M_{28\times 28}\left([0,1]\right)$.
- לנוחיותכם, בקבצי העזר נמצא הקובץ plot_img.py אשר מקבל כקלט נתיב לתמונה ומציג אותה בחלון חדש.
- וקטור הקלט שיישלח לרשת יהיה וקטור (מטריצה עם עמודה אחת) עם $28 \cdot 28 = 748$ שורות.

2.2.3 וקטור הפלט

- וקטור הפלט מהשכבה האחרונה הוא וקטור התפלגות (וקטור שאיבריו הם מספרים אי-שליליים שסכומם 1 באורך 10) .
- כל אינדקס בווקטור מייצג ספרה בין 0 ל-9
- הערך של האינדקס מייצג את הסיכוי שזוהי הספרה בתמונה, לפי הרשת.
- התשובה שתיתן הרשת היא האינדקס עם הערך המקסימלי, כלומר הספרה הסבירה ביותר, וההסתברות של אותה ספרה
- במקרה של שיוויון, נחזיר את האינדקס הנמוך מבין השניים
- לדוגמא:

Value	0	0.003	0.08	0	0	0	0	0.9	0.007	0.01
Index	0	1	2	3	4	5	6	7	8	9

בהינתן וקטור הפלט הזה, תשובת הרשת תהיה שהספרה בתמונה היא 7 בהסתברות 90%.

2.3 מהלך ריצת התוכנית

שימו לב: חלק זה ממומש עבורכם בקובץ main.cpp המצורף לכם עם קבצי התרגיל ואין צורך להגיש אותו.

התוכנה תקבל בשורת הפקודה נתיבים לקבצי המשקולות וההיסטים כקבצים בינאריים.

נריץ את התוכנה עם המשקולות וההיסטים כך:

```
$./mlpnetwork w1 w2 w3 w4 b1 b2 b3 b4
```

- w1 – נתיב לקובץ המשקולות של השכבה ה-i
- w2 – נתיב לקובץ ההיסט של השכבה ה-i

כאשר התוכנה רצה היא ממתינה לקלט מהמשתמש. הקלט יהיה נתיב לקובץ של תמונה המכילה ספרה. התוכנה:

1. תפתח את הקובץ ותטען את התמונה למטריצה
2. תכניס את המטריצה אל הרשת כקלט

3. כאשר התקבלה תוצאה, התוכנה תדפיס את התמונה, את הספרה שהרשת זיהתה ובאיזו הסתברות.לדוגמא (מתוך פתרון בית הספר):

4. התוכנה תמתין לקלט חדש

כאשר נזין לתוכנה q – התוכנה תצא עם קוד 0.

3 המחלקות למימוש

הינכם נדרשים לממש את המחלקות הבאות בלבד. אין להגיש מחלקות נוספות.

- בטבלאות המובאות לפניכם רשומות כלל הפונקציות והאופרטורים שעליכם לממש.
- אין להרחיב את ה-API המפורט, כלומר **אין להוסיף פונקציות public, constructors ו-destructors שלא נתונים במסמך זה** (ניתן להוסיף פונקציות private).
- חישבו היטב על החתימה של כל פונקציה: מהו ערך ההחזרה שלה? האם היא משנה את האובייקט הנוכחי? כיצד נגדיר את הטיפוס של הארגומנטים שלה?
- שימו לב: עליכם להחליט איפה יש להשתמש ב-const, איפה המשתנים וערכי ההחזרה צריכים להיות by value או by reference, והאם לממש כל פונקציה כחלק מהמחלקה (member function) או כפונקציה העומדת בפני עצמה (standalone, non-member function).
- **כפילויות קוד**: שימו לב שיש בתרגיל זה **המון** הזדמנויות למחזר קוד. קיראו את כל הפונקציות שעליכם לממש וחישבו כיצד ניתן להשתמש בהן כדי לחסוך קוד. אל תעבדו קשה, תעבדו חכם!
- **איסור על שימוש ב-STL: בתרגיל זה אין להשתמש בספריית STL של C++, ובפרט לא במבני נתונים כמו std::vector.**

3.1 המחלקה Matrix

מחלקה זו מייצגת אובייקט של מטריצה (גם וקטור הינו מטריצה, בעלת עמודה אחת ו-n שורות).
נזכיר כי איברי המטריצה יהיו מטיפוס float.

Description	Name	Comments
	Constructors	

Description	Name	Comments
Constructor	`Matrix(int rows, int cols)`	Constructs Matrix of size rows $\times$ cols. Inits all elements to 0.
Default Constructor	`Matrix()`	Constructs Matrix of size 1×1. Inits the single element to 0.
Copy Constructor	`Matrix(matrix)`	Constructs matrix from another Matrix m.
Destructor	`~Matrix()`	
	Methods & Functions	
Getter	`get_rows()`	returns the amount of rows as int.
Getter	`get_cols()`	returns the amount of cols as int.

	`transpose()`	Transforms a matrix into its transpose matrix, i.e(A . transpose())$_{ij} = A_{ji}$. Supports function calling concatenation. e.g:
		`Matrix a(5,4), b(4,5);`
		`a.transpose();`
		`a.get_rows == 4`
		`a.get_cols == 5`
		`b.transpose().transpose(); // b is same as before`

	`vectorize()`	Transforms a matrix into a column vector(section 3.1.2). Supports function calling concatenation. e.g.:
		`Matrix m(5,4);`
		`m.vectorize();`
		`m.get_cols() == 1`
		`m.get_rows() == 20`

	`plain_print()`	Prints matrix elements, no return value. Prints space after each element (including last element in row). Prints newline after each row (including last row).
	`dot(matrix)`	Returns a new matrix which is the element-wise multiplication (Hadamard product) of this matrix and another matrix: $\forall i, j : \left(A\ .dot(B)\right)_{ij} = A_{ij} \cdot B_{ij}$
	`sum()`	Returns the sum of the matrix: A.sum() $= \sum_{i,j} A_{ij}$
	`norm()`	Returns the Frobenius norm of the given matrix: A.norm() $= \sqrt{\sum_{i,j} A_{ij}^2}$
	`argmax()`	Returns the brackets index of the largest number in the matrix s.t A[a.argmax()] is the largest element in matrix A (see operator[] below)
	Operators	
`+=`	Matrix addition accumulation	`Matrix a, b;` $\rightarrow$ `a += b`
`+`	Matrix addition	`Matrix a, b;` $\rightarrow$ `a + b`

Description	Name	Comments
`=`	Assignment	`Matrix a, b;` → `a = b`
`*`	Matrix multiplication	`Matrix a, b;` → `a * b`
`*`	Scalar multiplication on the right	`Matrix m; float c;` → `m * c`
`*`	Scalar multiplication on the left	`Matrix m; float c;` → `c * m`

`()`	Parenthesis indexing	For i, j indices, Matrix m:
		m(i, j) will return the i, j element in the matrix e.g.
		`Matrix m(5,4);`
		`m(1,3) = 10;`
		`float x = m(1,3); // x == 10`

`[]`	Brackets indexing	For i index, Matrix m:
		m[i] will return the i'th element (section 3.1.2) e.g:
		`Matrix m(5,4);`
		`m[3] = 10;`
		`float x = m[3]; // x == 10`

`<<`	Output stream	Pretty export of matrix as per section 3.1.1
`>>`	Input stream	Fills matrix elements: stream must be big enough to fill the entire matrix. We do not care if the stream is too big for the matrix. see section 3.1.3 for more details.

3.1.1 הדפסת תמונה ממטריצה

כדי להדפיס את התמונה שמיוצגת במטריצה A באמצעות אופרטור >> , נשתמש בפסאודו-קוד הבא:

```
for i = 1 to A.rows:
     for j = 1 to A.cols:
          if A(i, j) > 0.1:
               print "**" (double asterisk)
          else:
               print "  " (double space)
     print newline
```

3.1.2 אינדקס יחיד לזכרון דו-מימדי

מטריצה A הינה אובייקט דו-מימדי, ולכן אנו זקוקים לשני אינדקסים על מנת לגשת לאיבר בה. פעמים רבות, נוח יותר לגשת לכל איבר במטריצה באמצעות אינדקס יחיד. נבצע את המיפוי מזוג אינדקסים לאינדקס יחיד באופן הבא:

$$A(i, j) \iff A[i \cdot \text{rowsize} + j]$$

- i – אינדקס השורה
- j – אינדקס העמודה

- rowsize – אורך השורה (מספר העמודות)

לדוגמה, תהי $A \in M_{3\times4}$, כלומר בעלת 3 שורות ו- 4 עמודות. מתקיים:
$A(2,1) \iff A[2 \cdot 4 + 1] \iff A[9]$

וודאו שאתם מבינים מדוע מיפוי זה הינו חד-חד-ערכי ועל.

3.1.3 קריאת תמונה מ-istream בינארי למטריצה

כדי לקרוא מידע בינארי מתוך istream, עליכם להשתמש בפונקציה std::istream::read, שמקבלת char* ומספר בתים\תווים לקרוא. תוכלו למצוא תיעוד ומידע על הפונקציה בלינקים הבאים:

https://www.tutorialspoint.com/reading-and-writing-binary-file-in-c-cplusplus

/https://www.cplusplus.com/reference/istream/istream/read

שימו לב, במקרה הזה תוכלו לעשות casting מפורש ל -char*.

3.2 קבצי Activation

בקבצים אלה נגדיר את פעולת פונקציות האקטיבציה. בקבצי ה-Activation עליכם לממש את פונקציות האקטיבציה relu ו-softmax כפונקציות השייכות ל-namespace activation. כלומר, על מנת לקרוא לפונקציות נצטרך לרשום activation::relu או activation::softmax. על פונקציות אקטיבציה לעבוד על כל המטריצה, ומחזירה את אותם מימדים כמו המטריצה המקורית.

טיפ: שימו לב שנשתמש בתרגיל זה ב-function pointers על מנת לגשת לפונקציות אלו. מומלץ להגדיר typedef של פוינטר לפונקצית אקטיבציה ולהשתמש בו בתוכנית.

3.3 המחלקה Dense

מחלקה זו מייצגת שכבה ברשת הנוירונים.

Description	Name	Comments
	Constructors	
Constructor	`Dense(weights, bias, ActivationFunction)`	Inits a new layer with given parameters. C'tor accepts 2 matrices and activation function
	Methods	
Getter	`get_weights()`	Returns the weights of this layer.
Getter	`get_bias()`	Returns the bias of this layer.
Getter	`get_activation()`	Returns the activation function of this layer.
	Operators	
()	Parenthesis	Applies the layer on input and returns output matrix. Layers operate as per section 2.2.1 e.g: `Dense layer(w, b, act);` `Matrix output = layer(input);`

3.4 המחלקה MlpNetwork

מחלקה זו תשמש אותנו לסדר את השכבות השונות למבנה רשת ותאפשר הכנסה של קלט לרשת וקבלת הפלט המתאים. מחלקה זו מממשת ספציפית את הרשת המתוארת במסמך זה (סעיף

(2.2.1. שימו לב כי struct digit מומש עבורכם בקבצים שקיבלתם. נקודה למחשבה: מה היה נדרש לממש במחלקה זו על מנת לתמוך ברשת עם מספר שכבות וגודל שכבות הניתן בזמן ריצה?

Description	Name	Comments
	Constructors	
Constructor	`MlpNetwork(weights[], biases[])`	Accepts 2 arrays of matrices, size 4 each. one for weights and one for biases. constructs the network described (sec. 2.2)
	Operators	
()	`Parenthesis`	Applies the entire network on input. returns digit struct. `MlpNetwork m( ... );` `digit output = m(img);`

4 טיפול בשגיאות

בתרגיל זה נדרוש מכם להשתמש ב-exceptions.

- חשבו בעצמכם היכן יכולות להיות שגיאות (בדיקת הקלטים של הפונקציות, בדיקת ערכי החזרה של פעולות שונות, ועוד).
- במקרה של שגיאה, זרקו חריגה (exception) מתאימה בהתאם להוראות הבאות:
  - אם התבצעה שגיאה הנוגעת למימדים בעייתיים, זרקו std::domain_error.
  - אם התבצעה שגיאה שקשורה לגישה למיקום לא חוקי, זרקו std::out_of_range.
  - אם התבצעה שגיאה עקב קלט לא נכון מהמשתמש זרקו std::runtime_error.
  - במקרה שהקצאת זיכרון נכשלה אינכם נדרשים לזרוק חריגה (החריגה std::bad_alloc תיזרק באופן אוטומטי במקרה זה. לפירוט קראו: https://en.cppreference.com/w/cpp/memory/new/bad_alloc). באופן כללי, בתרגיל זה אינכם נדרשים לשחרר את הזכרון במקרה של שגיאה, כיוון שזאת אחריות משתמש הספרייה לשחרר את הזיכרון. **אין זה אומר שלא יהיה לכם דליפות זיכרון**.

  רמז: שימו לב לדרישות ניהול הזיכרון במקרה של זריקת חריגה מ-constructor.

5 קימפול והרצה

בקבצי העזר לתרגיל הניתנים לכם, מצורף קובץ Makefile על מנת לקמפל את התוכנה. על התוכנה להתקמפל באמצעות הפקודה הבאה:

```
make mlpnetwork
```

נריץ את התוכנית כמפורט בסעיף 2.3.

5.1 Presubmit

קובץ ה-presubmit זמין בנתיב

```
~labcc/presubmit/ex4/srcs/presubmit.cpp
```

6 הקבצים להגשה

- MlpNetwork.h
- MlpNetwork.cpp
- Dense.h
- Dense.cpp
- Activation.h
- Activation.cpp
- Matrix.h
- Matrix.cpp

7 הערות וסיכום

7.1 הנחיות כלליות

- קראו בקפידה את הוראות תרגיל זה ואת ההנחיות להגשת תרגילים שבאתר הקורס.
- מעבר ל-C++: זכרו להשתמש בפונקציות ובספריות של C++ ולא C. למשל, נשתמש ב-new ו-delete ולא ב-malloc ו-free, ב-std::string ולא ב-char*, ובספריה <cmath> במקום math.h.
- איסור על שימוש ב-STL: נזכיר בשנית כי בתרגיל זה נאסר השימוש בספריית STL של C++, ובפרט אסור להשתמש במבני נתונים כגון std::vector
- ניהול זיכרון דינמי: היעזרו בתוכנה valgrind כדי לחפש דליפות זיכרון בתוכנית שכתבתם.
- **שימוש ב-reference**: הקפידו לא להעתיק by value משתנים כבדים, אלא להעבירם היכן שניתן by reference.
- **שימוש ב-const**: הקפידו להשתמש במילה השמורה const היכן שנדרש מכם בהגדרת הפונקציות והארגומנטים.
- סקריפט Pre-submission: ודאו כי התרגיל שלכם עובר את ה-Pre-submission Script ללא שגיאות או אזהרות.
- **כפילויות קוד**: שימו לב שיש בתרגיל זה **המון** הזדמנויות למחזר קוד. אל תעבדו קשה, תעבדו חכם!
- אין להניח שום הנחה על הגודל של קבצי הקלט.
- שימו לב: הפעם אתם לא צריכים להגיש טסטים. חרף זאת, אנו מעודדים אתכם לכתוב טסטים ולבדוק את עצמכם במהלך פתירת התרגיל.

בהצלחה! 🚀