1. תרגול שימוש חריגות (**Exceptions**), בקוד קיים.
2. אהרון משה שמסיאן: 205779598, עקיבא משה סודרנסקי: 326382322

**3. הסבר כללי של התרגיל:**

התרגיל מתמקד בשימוש במנגנון החריגות של C++ על מנת להבטיח טיפול נכון ויעיל במצבי שגיאה בתוכנית.

התרגיל דרש לתמוך בשתי סוגי זרמי לקט, מהטרמינל או מקריאת קובצים. המחשבון תומך בקריאת פקודות גם מקלט רגיל מהטרמינל (std::cin) וגם מקובץ, כאשר יש הבחנה ברורה באופן הטיפול בשגיאות בכל אחד מהמקרים.

בעת קריאה מהטרמינל:  
שגיאות מתורגמות להודעת שגיאה בלבד, והמשתמש יכול להמשיך לעבוד בצורה אינטראקטיבית.

בעת קריאה מקובץ:

במצב של שגיאה, התוכנית תציע למשתמש לבחור האם להמשיך לקרוא את הקובץ או לחזור לקלט מהטרמינל, וכן בפקודה resize שבה המשתמש רוצה לשנות את מספר הפונקציות שהמחשבון יכול להכיל, אך במקרה שבא לדרוס פונקציות קיימות, יוצג למשתמש אזהרה, האם הוא בטוח בבחירתו.

כלומר, קיימת פקודה מיוחדת (resize) שמאפשרת למשתמש לשנות את מספר הפונקציות הפעילות במחשבון (להקטין את גודל הרשימה). גם במקרה זה, אם נוצר מצב של שגיאה (למשל ניסיון להקטין למספר נמוך מדי או מחיקת פעולות בשימוש), התוכנית תדרוש מהמשתמש החלטה — האם להמשיך עם השינוי או לבטל את הפעולה ולהישאר במצב הנוכחי.

התרגיל מדגיש את חשיבות בתפיסת החריגות כמה שיותר למעלה בראש הקוד, כדי לאפשר כמה יותר תפיסות של החריגות, וזריקה כמה שיותר פנימה במקום הספציפי של החריגה. וכן שמירה על מבנה אחיד לקריאה והפרדה ברורה בין חריגות המיועדות להודעת שגיאה בלבד, לבין חריגות שדורשות תגובה פעילה מהמשתמש.

**4. רשימות קבצי התרגיל:**

**מימוש המחשבון.**

* **FunctionCalculator**  
  המחלקה הראשית המחזיקה את הליבה של המחשבון. אחראית על ניהול לולאת ההרצה של המחשבון, קריאת פקודות מהמשתמש או מקובץ, ניהול רשימת הפונקציות הפעילות, והפעלת הפקודות הנבחרות.  
  בנוסף, מרכזת את טיפול החריגות לכל אורך פעולתה.
* **SquareMatrix**  
  מחלקה המייצגת מטריצה ריבועית, ומאפשרת לבצע עליה פעולות מתמטיות שונות.  
  המטריצה משמשת כבסיס לכל החישובים במחשבון.
* **Operation**  
  מחלקת בסיס מופשטת שמייצגת פעולה כללית על מטריצות.  
  כל פעולה במחשבון (חיבור, חיסור, קומפוזיציה וכו') יורשת ממנה ומיישמת את הלוגיקה הספציפית שלה.
* **UnaryOperation**  
  מחלקת ביניים שמייצגת פעולות על מטריצה אחת.  
  פעולות חד-אופרטוריות כמו טרנספוזיציה והכפלה בסקלר מבוססות עליה.
* **BinaryOperation**  
  מחלקת ביניים שמייצגת פעולות על שתי מטריצות.  
  פעולות דו-אופרטוריות כמו חיבור, חיסור וקומפוזיציה מבוססות עליה.
* **Identity**  
  פעולה פשוטה שמחזירה את המטריצה כפי שהיא (זהות).
* **Scalar**  
  פעולה שמכפילה מטריצה בסקלר שהוזן על ידי המשתמש.
* **Add**  
  פעולה שמבצעת חיבור של שתי מטריצות.
* **Sub**  
  פעולה שמבצעת חיסור של מטריצה אחת ממטריצה אחרת.
* **Comp**  
  פעולה שמבצעת קומפוזיציה של שתי פעולות על מטריצות.

**מחלקות חריגה (Exceptions):**

**MatrixException**

מחלקת בסיס לכל מחלקות החריגה במערכת. יורשת מ־std::exception ומספקת מנגנון אחיד לניהול והצגת הודעות שגיאה עבור כל חריגה במערכת. כל מחלקות החריגה הספציפיות נגזרות ממנה ומיישמות את ההתנהגות הייחודית להן.

**MessageException**

מחלקת חריגה המיועדת למצבים של שגיאות קלט כלליות, אך סגנון טיפול החריגה הוא רק זריקה של פלט שגיאה אל המשתמש ללא אפשרויות לעבור מצב מקובץ אל טרמינאל. בעת זריקה של חריגה זו, המערכת מציגה הודעת שגיאה, מנקה את מצב השגיאה ואת כל שאריות השורה מהחוצץ, ומאפשרת למשתמש להמשיך להזין פקודות חדשות בצורה נקייה.

**ResizeException**

מחלקת חריגה המיועדת למקרים בהם המשתמש מנסה לבצע פקודת resize שגורמת לבחירה חדשה של מספר הפונקציות שיכולות להיות במחשבון.

במקרה שהמשתמש מנסה להקטין את מספר הפונקציות של המחשבון ובכך דורס פונקציות שקיימות במחשבון, יוצג אל המשתמש הודעה אזהרה! האם הוא בטוח בבחירה שלו, לחצן "Y" "y" מהווה אישור, כל מקש אחר מהווה סירוב, והפעולה מתבטלת.

**ManyArgumentsException**

מחלקת חריגה המשמשת למצבים בהם המשתמש מזין יותר מדי פרמטרים לאחר פקודה (לדוגמה: הכנסת יתר ארגומנטים בפקודת eval).  
במקרה זה, המערכת מנקה רק את מצב השגיאה (clear) אך משאירה את החוצץ כמו שהוא (ללא ignore), כדי לאפשר למשתמש לראות את שאר הפרמטרים שהוזנו ולהחליט כיצד להמשיך לפעול.

יחדנו לו מחלקה בפני עצמה משום שבחריגה שלו לא נבצע את השורה istr.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

משום שאם כן הוא ימתין לזרם חדש נוסף, ולא יציג נתונים בenter בלבד.

**5. פירוט מבנה נתונים עיקריים:**

במחשבון יש כמה מבני נתונים מרכזיים:

**וקטור פעולות (m\_operations)** — שומר את כל הפעולות שהמשתמש יוצר (חיבור, חיסור, קומפוזיציה וכו'), כולן יורשות מאותה מחלקת בסיס (Operation).

**רשימת פקודות (m\_actions)** — וקטור של פקודות שהמחשבון תומך בהן, עם שם ותיאור של כל פקודה.

**זרם קלט**  — מטיפוס istream שמנהל את הקלט שנכנס למחשבון, בין אם זה מקובץ ובין אם זה מהטרמינל, והכל מנוהל באותו קוד.

**מחלקות חריגה** — היררכיית חריגות שמפרידה בין שגיאות רגילות, שגיאות הדורשות החלטה מהמשתמש, ושגיאות של קלט עודף.

**6. אלגוריתמים חשובים שהשתמשנו בהם.**

במערכת השתמשנו בעיקר בלולאות קריאה מהקלט ובשימוש בפונקציות גנריות (run(std::istream&)) שמאפשרות להריץ את אותו הקוד גם על קלט רגיל וגם על קובץ.

**7. תיכון התרגיל:**

בתיכון המחשבון יישמנו היררכיית חריגות ברורה ומובנית בהתאם לדרישות התרגיל, תוך הפרדה בין סוגי השגיאות השונים וההתנהגות הרצויה בכל מקרה.  
במקום שימוש בחריגה אחת גנרית לכל מצבי השגיאה, הוחלט על הפרדה ברורה לשני סוגים עיקריים של חריגות, לצד חריגה ייחודית נוספת:

**חריגות שכל מטרתן הצגת הודעה בלבד (MessageException)**

מחלקה זו מטפלת בכל השגיאות הכלליות שנובעות מטעויות משתמש או קלט שגוי, כמו פקודה לא חוקית, ערכים מחוץ לטווח, וכו'.  
החריגה מדווחת למשתמש על הבעיה, מנקה את מצב השגיאה ואת שאריות השורה מהקלט (ignore), ומאפשרת המשך עבודה תקין.  
חריגה זו **אינה דורשת מהמשתמש בחירה אקטיבית**, אלא מספקת הודעה בלבד.

**חריגת resize דורשות החלטה מהמשתמש (ResizeException)**

מחלקה זו מייצגת מקרים שבהם יש סכנה לפעולה שמשנה את מצב המערכת בצורה משמעותית (כמו מחיקת פונקציות קיימות בפקודת resize).  
במקרה זה, לאחר הצגת הודעת השגיאה, המערכת מצפה מהמשתמש לקבל החלטה אקטיבית האם להמשיך בפעולה או לא. גם כאן הזרם נוקה לחלוטין מהשגיאה ומהקלט הנותר (ignore), כדי למנוע בעיות בקלט הבא.

**חריגה ייחודית למצבים עם קלט עודף (ManyArgumentsException)**

מחלקה זו מיועדת למצב ייחודי בו המשתמש מזין **יותר מדי פרמטרים אחרי פקודה** לדוגמה:( eval 0 2 3) להבדיל מהחריגות הקודמות, כאן המערכת מנקה רק את מצב השגיאה (clear) **ללא קריאה ל־ ignore**,כדי לאפשר למשתמש להמשיך לטפל בשאר הקלט שהוזן — ולא למחוק אוטומטית את כל החוצץ .כך המערכת מאפשרת טיפול גמיש ומדויק יותר במקרים של קלט עודף.



בנוסף, במקרים שבהם נדרש מהמשתמש לאשר פעולה (כמו המשך קריאה מקובץ או ביצוע אימות בפקודה resize מימשנו שאלה ברורה למשתמש שבה הוא מתבקש להקליד y או Y לאישור.

כל תו אחר נחשב לסירוב או ביטול פעולה, ואין צורך לזרוק שגיאה במקרים כאלה, כי זו לא טעות — זו בחירה לגיטימית של המשתמש לעצור או לצאת ממצב הקריאה מהקובץ.

**8. אין באגים ידועים.**

**9. הערות כלליות:**

בתרגיל הזה החלטנו לבנות את המחשבון בצורה שבה יש לנו שתי פונקציות run.  
האחת היא ההפעלה הראשית של המחשבון run(), והשנייה מקבלת כפרמטר זרם קלט (run(std::istream&istr)).

המבנה הזה מאפשר לנו להעביר בצורה פשוטה את סוג הקלט — בין אם מדובר בזרם מהטרמינל ובין אם מדובר בקובץ — וכל הקוד שנמצא בתוך run(std::istream&) מתנהל בצורה אחידה על שני סוגי הקלט בלי צורך בשכפול קוד. בזכות זה, המעבר בין קובץ לטרמינל מתבצע בקלות, פשוט מחליפים את הזרם ואותו קוד בדיוק ממשיך לעבוד, מה שחוסך הרבה קוד מיותר ומאפשר לנהל את כל הלוגיקה במקום אחד.

בנוגע לשאלה אם המחשבון תומך בפקודת read מתוך קובץ, בחרנו שלא לתמוך בזה.  
כלומר, אם בתוך קובץ פקודות יש פקודת read, אז כשהקובץ החדש מסתיים, נחזור ישר לטרמינל ולא נחזור להמשך הקריאה מהקובץ הראשון.עשינו את זה כי רצינו להימנע מלנהל מבנה מסובך של מחסנית זרמים או קבצים פתוחים, מה שהיה יכול לסבך את הקוד בלי להוסיף ערך אמיתי לתרגיל.

בנוסף, החלטנו לקרוא את הקלט כזרם מלא (ולא שורה־שורה עם getline) כדי לתת למשתמש גמישות להכניס פקודות ומטריצות בצורה נוחה יותר, ולנו זה מאפשר לטפל בשגיאות כמו עודף פרמטרים בצורה הרבה יותר נקייה וברורה.

**דוגמה מבנה הקובץ שנתמך לבדיקת התרגיל:**

eval 0 2

1 2

3 4

eva

eval w

eval -1

eval 0 k

eval 3 0

eval 0 1 6

eval 0 2

1 500

1 2

eval 0 2

4 1

2 3

add 0 1

eval 2 2

1 2

3 4

1 2

3 4

eval 2 2

1 2

3 4

999 999

999 999

scal 2

eval 3 2

1 2

3 4

sub 0 1

eval 4 2

1 2

3 4

1 2

3 4

comp 0 1

eval 5 2

1 2

3 4

del 10

del 2

resize 200

resize 50

resize 2

eval 1 2

add 0 3

add 0 0

**וכדו׳, שהפקודות בקובץ ממש זהות למבנה של הטרמינל, וכדי לדעת איזה מספר שורה השגיאה, נקפיד על מבנה הקלט תקין לפי שורות.**