



תרגול 6 טיפול בשגיאות

רקע

אחד הדברים שכל מתכנת/ת צריך לדעת זה לטפל בשגיאות שעלולות להופיע בקוד ובתוכניות שהוא או היא כתבו.

בשיעור התנסנו בפיצ'רים של ה-Debugger של Visual Studio כדי לאתר ולפתור באגים מסוגים שונים, ראינו שמעבר לבאגים תכנותיים, בתוכניות שלנו עשויים להופיע גם באגים עיצוביים, ושזה לא תמיד פשוט לבנות תוכניות עמידות וחסיונות.

כשנתכנת נרצה להוסיף לתוכניות שלנו מנגנונים שיעזרו באיתור והבנה של בעיות ושגיאות, ויש יותר מדרך אחת לעשות את זה, בתרגיל נראה שני מנגנונים שיאפשרו לנו לנהל את השגיאות בקוד.

מטרה

בתרגיל נעשה שני דברים,

1. **כתיבת תשתית לניהול שגיאות** – נממש שני מנגנונים לטיפול בשגיאות.

- **החזרת ערכי שגיאה (error codes)**

- **חריגות (exceptions)**

2. **השתלבות בקוד קיים** – נעשה שינויים על קוד שנכתב עבורנו:

- נפתור באגים הקשורים לניהול שגיאות וחריגות.

- נוסיף פונקציונליות חסרה.

- נבצע refactoring – עדכון וכתיבה מחדש של חלקים בקוד כדי לשפר את איכותו.

אלה השלבים שנעבור:

משימה 1 – מחשבון	משימה 2 – Shapes
<ul style="list-style-type: none">• הוספת קודי שגיאה• הוספת exceptions	<ul style="list-style-type: none">• תיקון באגים• Refactoring והוספת פונקציונליות

נתרגל מיומנויות חשובות:

- ניהול שגיאות בתוכנית
- השתלבות בקוד קיים
- פתרון באגים בתוכנית
- את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT](#).
כדאי לקרוא גם [דגשים לתכנות נכון](#).



"PRACTICE MAKES PERFECT"

בהצלחה יא אלופות ואלופים!

שלב 1: מחשבון

שלב 2 Shapes	שלב 1 מחשבון
-----------------	-----------------

תרגיל 1 – חריגות במחשבון

התחלתם לעבוד על קוד למחשבון שישתלב באפליקציה הרשמית של תכנית מגשימים, אך לפתע

קיבלתם/ממנהלי התכנית **דרישה חדשה למימוש**:



- בפעולה חוקית של התכנית אסורות ההתנהגויות הבאות:

א. שהערך **8200** יוחזר מאחת הפונקציות בתוכנית

ב. שמשתנה שמוכרז בתוך אחת הפונקציות יחזיק את הערך **8200**

כשמצב זה קורה, יש להדפיס למשתמש הודעה

- *"This user is not authorized to access 8200, please enter different numbers, or try to get clearance in 1 year"*

לרשותכם שני קבצים זהים. בכל אחד מהם עליכם לממש את הפתרון בדרך אחרת:

א. **basic_calc.cpp** - עם שימוש בערכי חזרה, וללא שימוש בחריגות. אין לשנות את האלגוריתמים של הפונקציות השונות - כלומר, חזקה יכולה להתבצע רק באמצעות קריאה לכפל, וכפל רק באמצעות קריאה לחיבור.

- שימו לב שעל פעילות המחשבון עבור מספרים שאינם מערבים את 8200 להישאר תקינה. לכן, מבלי לשנות את חתימות הפונקציות לא תוכלו להשתמש בערך החזרה של הפונקציות כדי לטפל בשגיאות. תהיו חייבים לשנות את חתימת הפונקציות ולהוסיף להן ארגומנט. כתבו קוד קצר ב-main שבודק קלטים חוקיים ולא חוקיים לשלוש הפונקציות.

ב. **excep_calc.cpp** - עם שימוש בחריגות. זה אמור להיות החלק הפשוט.

כתבו קוד קצר ב-main שבודק קלטים חוקיים ולא חוקיים לשלוש הפונקציות. השתמשו בכמה בלוקים של try ו-catch כדי שיזרקו כמה חריגות בריצה אחת של התוכנה.

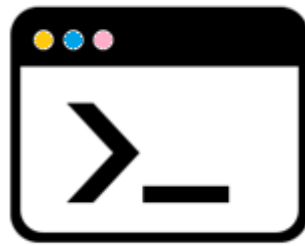
שלב 2: Shapes

שלב 1 מחשבון	שלב 2 Shapes
-----------------	-----------------

תרגיל 2 – *Command Line Interface* עבור צורות

לרשותנו תוכנה שמאפשרת למשתמש ליצור אובייקטים של צורות עם תכונות שונות, בדומה לתוכנה שכתבנו עבור תרגיל 5, אך עם ממשק טקסט במקום ממש גרפי.

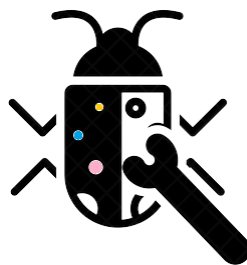
התוכנה כתובה עם טעויות רבות, עם קונבנציות גרועות, והיא לא נעימה לקריאה. רצוי לא ללמוד ממנה על איך לכתוב קוד נכון, לתקן רק את החלקים בתרגיל שאתם מתבקשים לתקן, ולא לכתוב דברים מחדש – זה צפוי רק לבזבז לכם זמן. בהצלחה 😊



[חלק ראשון – תיקון באגים:](#)

בודקי תוכנה חרוצים התלוננו על איכות הקוד, וגילו בתוכנה **שלושה** באגים:


א. כשנזרקות חריגות הן לא תמיד מדפיסות את הטקסט המצופה מהן. למשל, אם המשתמש בוחר למעגל **רדיוס שלילי**, לא יודפס "*This is a shape exception!*" כמצופה (*ShapeException* יורש מ-*std::exception*, ולכן אמור להיתפס ב-catch של *std::exception*). פתרו באג זה.



ב. כשהמשתמש מכניס קלט של טקסט במקום בו התוכנה מצפה לקבל מספר (למשל, רדיוס המעגל), התוכנה מתחילה בהתנהגות בלתי צפויה והופכת ללא שמישה. כדי לפתור באג זה נדרש מעט מחקר אינטרנטי והוספת קוד.

- מחקר בגוגל הוא כלי מאוד חשוב (אולי הכי חשוב?) עבור מתכנת/ת. נסו למצוא את הפתרון לבעיה בעצמכם/ן. אם תרצו לוודא שחיפשתם נכון, בסוף התרגיל יש מילות חיפוש לדוגמא שמוצאות את מה שצריך.

a. הוסיפו תשתית ניהול שגיאות מתאימה, המשתמשת בחריגות. צרו לשם כך מחלקת `InputException` היורשת מ-`std::exception`, וזרקו `InputException` בכל מקום בו הבאג הנ"ל קורה.

b. שימו , `InputException` ו-`ShapeException` יצטרכו להיתפס בבלוקי `catch` שונים, כי בתפיסת `InputException` יצטרך לרוץ קוד נוסף חוץ מהדפסת השגיאה. במקרה זה סדר בלוקי ה-`catch` יהיה חשוב.

ג. כשהמשתמש מתבקש לבחור צורה לעבוד איתה, אם הוא מכניס יותר מאות אחת במקום את אחת האותיות הנדרשות (למשל "cpf" במקום 'c'), קורה משהו מוזר. תקנו את ההתנהגות המוזרה בכל צורה שתצליחו, והמשיכו בריצה תקינה של התוכנית לאחר הדפסת ההודעה הבאה

- ("Warning – Don't try to build more than one shape at once")

⚠ הערה

פונקציונליות התוכנה צריכה לא להשתנות בעקבות תיקון הבאגים, אלא רק לעבוד בצורה תקינה ולטפל בשגיאות הנ"ל בצורה תקינה.



חריגות:

- א. יש לוודא שהמקבילית מקבלת רק קלטי זוויות חוקיים, ואם לא לזרוק `ShapeExecption`. מותר למקבילית להיות "קו" – עם זוויות של 0 ו-180 מעלות.
- ב. מחלקת `Circle` שקיבלה רדיוס שלילי, תזרוק חריגה רק בעת חישוב היקף המעגל. התנהגות זו פותחת מקום לבאגים, שכן אפשר לבנות אובייקט `Circle` עם רדיוס שלילי ולבצע עליו פעולות אחרות, שלא דווקא זורקות חריגה. העבירו את החריגות במחלקת `Circle` ובמחלקות הצורות האחרות לנקודה המוקדמת ביותר האפשרית בחיי האובייקט, בין אם הוא נוצר דרך הבנאי, או "נוצר מחדש" עם עדכון דרך `Setter`-ים.

פונקציות סטטיות:

- א. היזכרו במשתנים ובפונקציות סטטיות בעזרת מצגת הלימוד העצמי של שיעור 4.
- ב. צרו מחלקה בשם `MathUtils`, שמטרתה לרכז את כל הפונקציות המתמטיות המסובכות בתוכנה במקום אחד, ולייצא אותן כפונקציות סטטיות לשאר מחלקות התוכנה. כתבו בה שתי פונקציות סטטיות:
- a. `CalPentagonArea` – מקבלת `double` המייצג אורך צלע של מחומש משוכלל ומחזירה את שטחו (הנוסחא בויקיפדיה)
- b. `CalHexagonArea` – מקבלת `double` המייצג אורך צלע של משושה משוכלל ומחזירה את שטחו (הנוסחא בויקיפדיה)
- ג. כתבו שתי מחלקות חדשות – `Pentagon` ו-`Hexagon`, היורשות מ-`Shape`, מממשות את המתודות האבסטרקטיות שלה, מחזיקות member בודד המייצג צלע, ומממשות setter ל-member זה. כדי לחשב את השטח, קראו לפונקציות הסטטיות שכתבתם ב-`MathUtils`.
- ד. הוסיפו את שתי הצורות החדשות לתפריט ב-main



סיכום תרגיל 6

בסיום התרגיל יש להעלות ל-GIT את ה-VS Solution עם שני פרויקטים נפרדים:

1. פרויקט 1 שמכיל את הקבצים:

- `basic_calc.cpp`
- `excep_calc.cpp`
- קבצי קוד נוספים אם נדרשים

2. פרויקט 2 שמכיל את הקבצים:

- כל הקבצים שקיבלתם עם הבאגים הנ"ל פתורים, קבצי המחלקות `Pentagon`, `Hexagon` ו-`MathUtils`, הקובץ `InputException.h`, וקבצים נוספים שכתבתם אם נדרשו כאלו

! רמז לבאג 2 !

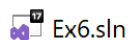
חיפוש ***cin infinite loop*** מגיע בתוצאה הראשונה לדוגמא מלאה

חיפוש ***cin bad input*** מגיע בתוצאה הראשונה לדוגמא בא נמצאות שתיים מהפונקציות בהן צריך להשתמש, ואז אפשר לחפש את השלישית

נספחים

הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו). חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](#)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
- הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ gitignore. אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטוני עזר בנושא GIT](#).
- כסיימתם/ן, בדקו שניתן להריץ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה



דגשים:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm).
- יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
- עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
- יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
- במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות בסרטוני ההדרכה בנושא GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. להזכירכם! העבודה היא עצמית, ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה.

דגשים לתכנות נכון

- כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
- כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
- בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
- חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.