סדנת דבאג

מדריך למידה

עקרונות מתקדמים בתכנות

תוכן עניינים

מבוא

שלום

לפניכם מדריך למידה בנושא debugging.

דבאגינג יכול להפוך למשימה קשה ומורכבת. על מנת להצליח במשימה זו צריך לשלוט בשלושה דברים:

1. שפת התכנות ומבנה המחשב
2. הקוד של התוכנה
3. סביבת העבודה והכלים (דבאגר, פרופיילר וכו'..)

במדריך זה תמצאו סדרת תרגילים אשר יאפשרו לכם להתנסות בפתרון באגים שונים ובשיטות עבודה מגוונות עם ה- debugger.

כמו כן, מצורף נספח הכולל הסברים נוספים על סביבת הדבאגר. **מומלץ מאוד לקרוא אותו.**

אופן ההגשה

- מגישים את ה- solution כולל כל הפרויקטים עם הקוד המתוקן.
- את הקבצים מעלים לגיט. וודאו כי העלתם רק את הקבצים הנחוצים לבניית הפרוייקטים.

באג

באג (בעברית תֶּקֶל; באנגלית Bug) הוא תקלה במערכת מבוססת תוכנה, שמתנהגת בצורה שגויה, לא צפויה או שאיננה מתוכננת.

סוגי באגים

בדר"כ מחלקים את סוגי הבאגים לשתי קטגוריות:

1. באגים תכנותיים – באגים שנובעים מטעות בתכנות (למשל אי קידום של משתנה בלולאה יוביל ללולאה אינסופית).
2. באגים עיצוביים – באגים שנובעים מתקלות בעיצוב התוכנה

דבאגינג

נִפּוּי שְׁגִיאוֹת או דבאגינג (באנגלית debugging) הוא התהליך של זיהוי והסרה של שגיאות מתוכנת מחשב.[1]

מדוע חשוב לדעת לדבאג?

- ככל שהתוכנית גדולה יותר, רבים הסיכויים שיהיו באגים.
- תיקון בעיות הוא שלב הכרחי בתהליך פיתוח תוכנה.
- דיבוג עוזר לפתח הבנה טובה לגבי מהות הבאגים ומקורם ומוביל לכתיבת קוד טוב יותר, נכון יותר ועם פחות באגים.

דבאגרים

בקורס עקרונות נעבוד עם ה- debugger המוטמע בתוך Visual Studio[2] אבל ישנן דבאגרים אחרים וכדאי להכירם:

1. WinDBG – בעל יכולות רבות, אך יחד עם זאת לא אינטואיטיבי לשימוש.
2. GDB – מתאים לסביבת לינוקס.
3. LLDB – חוצה פלטפורמות, אך לרוב בשימוש בסביבת אפל (מק, אייפון) ואנדרואיד.

ניתוח קוד

סטטי

לרוב יעשה באופן ידני ע"י קריאה של הקוד והבנה של פעולתו או באופן אוטומטי ע"י כלי תוכנה מתאימים. בניתוח קוד סטטי לא מפעילים את התוכנה ולא מריצים את הקוד.

דינמי

מחייב הרצה של הקוד אותו נרצה לבדוק. לעיתים תתבצע הריצה באמצעות דבאגר וניתוח ידני של איש תוכנה, אך יחד עם זאת ישנם כלים שמאפשרים ניתוח דינמי מורכב מאוד ע"י שינוי הקוד באופן זמני ומעקב אחר פעולת הקוד[3].

---

[1] מתוך מילון אוקספורד

[2] Visual Studio הוא תוכנה הנקראת IDE – Integrated development environment. כלומר, סביבת פיתוח משולבת הכוללת לרוב מהדר (קומפיילר) ודבאגר, יחד עם כלים נוספים המקלים על פיתוח תוכנה.

[3] טכניקה זו נקראת אינסטרומנטציה (באנגלית instrumentation). לקריאה נוספת: https://en.wikipedia.org/wiki/Instrumentation_(computer_programming)

כללי אצבע לדיבוג

שיחזור הבאג

- באג שחוזר על עצמו באופן דטרמיניסטי (קבוע וצפוי) קל יותר לדיבוג. באגים לא דטרמיניסטים הם מהסוג הנורא ביותר וקשה מאוד לגלות אותם.
- במידה והקוד גדול ומורכב, כדאי לנסות לצמצם אותו ע"י כתיבת תוכנית קטנה יותר הגורמת לבאג.

זיהוי מקור הבאג

- קראו היטב את הקוד ובחנו את המידע המועבר לפונקציות השונות ואת סדר הקריאות על המחסנית (callstack).
- הציעו השערה (רעיון כלשהו שמסביר את הבעיה) ובחנו אותה.
- ניסוי וטעיה - נסו קלטים שונים ובחנו את התנהגות התוכנית.
- החליפו קטעי קוד באחרים או בטלו אותם לגמרי.

דיבוג מודרך

MyPointExactly

לפניכם תרגיל בעל מספר שלבים.

במהלך העבודה תלמדו להכיר את הכלים שהדבאגר מעמיד לרשותנו.

עקבו אחר השלבים ובצעו אותם **לפי הסדר**.

חלק 1

פיתחו את הפרוייקט **MyPointExactly**

א. קראו את הקוד והבינו מה מטרתו.
ב. הריצו את התוכנית. האם הודפס למסך מה שציפיתם?
ג. דבאג
   a. קבעו breakpoint (BP) בתחילת הפונקציה operator>>. תוכלו לעשות זאת ע"י סימון השורה ולחיצה על F9.
   b. הריצו את התוכנית ע"י לחיצה על F5.
   c. האם הדבאגר עצר ב- BP? למה?
ד. פונקציית ההדפסה של המחלקה Point לא נקראה. תקנו את הבעיה והריצו את התוכנית.

חלק 2

א. עכשיו מתבצעת הדפסה של האובייקט Point למסך. האם אלו הערכים שציפיתם שיודפסו?
ב. דבאג
   a. יתכן ויש באג בקוד של אופרטור ההשמה.. קבעו BP בתחילת הפונקציה operator=
   b. פתחו **חלון Memory** והזינו לתוך שורת הכתובת את שם המשתנה coord_.
   c. התקדמו עם ריצת התוכנית שורה אחר שורה באמצעות **Step Over**. עשו זאת ע"י לחיצה על F10.
   d. שימו לב לשינויים בחלון הזיכרון כשפקודת memcpy מתבצעת. תוכלו לראות כי רק שני בתים התעדכנו.
   e. פתחו חלון Memory נוסף והזינו לתוכו את שם המשתנה other._coord. כך תוכלו לראות את הזיכרון שממנו נרצה להעתיק ואת הזיכרון שאליו נרצה להעתיק.
   f. תוכלו לראות את ההבדלים בין הערכים בזיכרון?
ג. תקנו את הבאג, כך שההעתקה תתבצע כנדרש והריצו את התוכנית.

חלק 3

א. תקנו את הקוד כך שההשמה תתבצע בזמן ההצהרה על משתנה p2 כך:

```
Point p2;
p2 = p1;
```

→

```
Point p2 = p1;
```

ב. הריצו את התוכנית... מה קרה?

ג. דבאג

a. קבעו BP בסוף main (על שורת ה- return). האם הדבאגר עצר?

b. הריצו את הדבאגר באמצעות F10 והתקדמו שורה אחרי שורה. מתי נזרקת השגיאה?

c. כנראה שהבעיה היא בשחרור האובייקט. קבעו BP ב- destructor של Point והריצו את הדבאגר.

d. וודאו שהדבאגר עוצר ב- destructor ופתחו את **חלון Auto**. בחלון זה תוכלו לראות את הערכים של האובייקטים בסקופ הנוכחי. במקרה שלנו אלה יהיו _coord ו- this.

e. המשיכו את ריצת התוכנית (ע"י F5). כעת הדבאגר נעצר ב- Point::~Point שוב והערכים בחלון Auto השתנו. מדוע?

f. הפסיקו את ריצת התוכנית (תוכלו לעשות זאת ע"י Shift+F5) והריצו את התוכנית שוב.

g. כעת כשהדבאגר עוצר ב- destructor פיתחו את **חלון Immediate** (חלון זו מאפשר לנו לבחון ערכים של אובייקטים, הדפסת תוכן זיכרון, ביצוע חישובים ועוד פעולות רבות.)

   i. כתבו את המילה this.

   ii. כתבו את המילה _coord

h. המשיכו את ריצת התוכנית עד לעצירה נוספת ב- Point::~Point והשוו את הערכים החדשים המופיעים בחלון Auto עם הערכים הקודמים שמופיעים בחלון Immediate.

i. מה השתנה? מה נשאר אותו דבר? מה לדעתכם הסיבה לכך?

j. הפסיקו את ריצת התוכנית וקבעו BP באופרטור ההשמה Point::operator=

k. הריצו את הדבאגר שוב. האם הוא עצר ב- BP? למה?

ד. התבוננו שוב בשינויי הקוד שהכנסנו. איזו פונקציה תקרא?

a. קבעו BP בתחילת הפונקציה הנכונה ובצעו Step by step.

b. איזו סוג העתקה מתבצעת כאן?

ה. תקנו את הקוד והריצו את התוכנית שוב

חלק 4

אופרטור ההעתקה במחלקה Point לא ממומש נכון. האם תוכלו לאתר את הבאג?

רמז:

הוסיפו את השורות הבאות לקוד:

```
p2 = p2;
std::cout << "p2=" << p2 << std::endl;
```

בונוס: תקנו את הבאג[4]

[4] קראו על self-assignment כאן:
https://stackoverflow.com/questions/12015156/what-is-wrong-with-checking-for-self-assignment-and-what-does-it-mean

תרגילים

את הבאגים הבאים עליכם למצוא בעצמכם. השתמשו בכל הכלים שלמדתם.

בהצלחה!

LoopAgain

פיתחו את הפרוייקט **LoopAgain.**

התבוננו בקוד שלפניכם. נסו להבין מה מטרתו.

הריצו אותו. מצאו את הבאג ותקנו אותו!

Shapes

פיתחו את הפרוייקט **Shapes.**

התבוננו בקוד שלפניכם. שני המשולשים זהים ולכן שטחם אמור להיות זהה.

אך האם זה המצב? הריצו את הקוד.

מדוע שטח המשולש השני הוא אפס?.. מצאו את הבאג ותקנו אותו!

SafeAndSound

פיתחו את הפרוייקט **SafeAndSound.**

לתרגיל זה שני חלקים: part1 ו- part2

חלק 1

התבוננו בקוד שבקובץ **part1.cpp**. מה עושה הקוד?

וודאו כי בקובץ SafeAndSounds.cpp בפונקציה main ישנה קריאה ל- **part1** והריצו את הקוד.

מה קרה? מצאו את הבאג ותקנו אותו!

חלק 2

בקובץ **part2.cpp** תמצאו מימוש בטוח יותר של הפונקציה מחלק 1.
גירסה זו אמורה להגן מפני buffer overflow[5].

וודאו כי בקובץ SafeAndSounds.cpp בפונקציה main ישנה קריאה ל- **part2** והריצו את הקוד.

התבוננו היטב בפלט... למסך הודפס **מידע סודי**, במקרה שלנו סיסמה פרטית. איך זה אפשרי?

מצאו את הבאג ותקנו אותו!

[5] https://en.wikipedia.org/wiki/Buffer_overflow

Password

פיתחו את הפרוייקט **Password.**

האם תוכלו למצוא את הסיסמה הנכונה?

האם צריך בכלל סיסמה?

בקוד שלפניכם קיים באג המאפשר לכל אחד להכנס למערכת גם ללא סיסמה..

קראו את הקוד היטב, הבינו את פעולתו ואז, בעזרת דבאגר, נסו למצוא דרך לעקוף את המערכת.

שימו לב:

אין צורך לתקן את הבאג בתרגיל זה.

הפתרון יכלול קלט הגורם להדפסת Congratulations על המסך ובנוסף, הסבר קצר על מהות הבאג.

ImagesAndWords

פיתחו את הפרוייקט **ImagesAndWords.**

לפניכם קוד לקריאה וכתיבה של תמונה (בפורמט דמיוני בשם MAGI).

שימו לב כי בתיקיית הקוד (לצד קובץ ה- cpp) יש שני קבצי תמונות. קבצים אלו הם חלק מהתרגיל.

התמונה הראשונה (**img1.magi**) היא תקינה ועובדת היטב.

התמונה השנייה (**img2.magi**) היא תמונה זדונית, שנוצרה ע"י האקר. קריאה של תמונה זו גורמת לקריסה של התוכנית.

מצאו את הבאג ותקנו אותו!

נספח

Breakpoints

קיימים 4 סוגי Breakpoints

1. Software breakpoint: מאפשר לנו לעצור את ריצת התוכנית בכל מקום בקוד (ע"י int 3)
2. Hardware breakpoint: ע"י שימוש ברגיסטרים מיוחדים למטרה זו.
3. Memory breakpoint – ממומש ע"י הדבאגר ומאפשר לעצור בזמן קריאה או כתיבה לזיכרון.
4. Conditional breakpoint – ממומש ע"י הדבאגר ומאפשר לעצור רק כשתנאי מסויים מתקיים.

שליטה בריצה

Single Step: ריצה של שורת קוד אחת ועצירה (ע"י **F10**)

Step Into: כניסה לתוך פונקציה ועצירה בתחילתה (ע"י **F11**)

Continue: המשך ריצת התוכנית (ע"י **F5**)

Run To Cursor: ריצה עד המקום הנוכחי של הסמן ועצירה. מאוד שימושי לדילוג על קטעי קוד ארוכים (ע"י **Ctrl+F10**).

חלונות מידע

Memory

מאפשר לנו לצפות בזיכרון. כל שינוי בערכים יתעדכן בזמן אמת.

ניתן לפתוח עד 4 חלונות זיכרון.

הקישו על **Ctrl+Alt+M** ולאחר מכן **1** כדי לפתוח את חלון הזיכרון הראשון. Ctrl+Alt+M ו- 2 יפתח את חלון הזיכרון השני וכך הלאה.

Call Stack

מאפשר לנו לראות את כל קריאות הפונקציות במחסנית עד לנקודת העצירה

הקישו על **Ctrl+Alt+C** לפתוח את חלון זה.

Autos

מתעדכן בזמן אמת ומציג את ערכי האובייקטים בסקופ הנוכחי.

הקישו על **Ctrl+Alt+V** ולאחר מכן **A** לפתוח חלון זה

Immediate Window

חלון מאוד שימושי המאפשר הדפסה של ערכים שונים (משתנים, כתובות, זיכרון, רגיסטרים ועוד..)

בנוסף מאפשר לעדכן ערכים, לבצע חישובים חשבוניים ועוד.

הקישו על **Ctrl+Alt+I** לפתוח חלון זה

Registers

מציג את ערכי הרגיסטרים. הערכים מתעדכנים בזמן אמת.

שימושי מאוד לדיבוג בלי קוד מקור.

טיפ: רגיסטר eax (או rax) מכיל את ערך החזרה מפונקציות.

הקישו על **Ctrl+Alt+G** לפתוח חלון זה.

Disassembly

מציג את הקוד הבינארי של התוכנית. ניתן לבצע Single Step לפקודות אסמבלי.

מאוד שימושי למחקר low level.

הקישו על **Ctrl+Alt+D** לפתוח חלון זה.