גירסה 1.00 – 10.10.2002

תכנון ובדיקת תוכניות – מספר נקודות

ניר אדר

מסמך זה הורד מהאתר www.underwar.co.il

אין להפיץ מסמך זה במדיה כלשהי, ללא אישור מפורש מאת המחבר.

מחבר המסמך איננו אחראי לכל נזק, ישיר או עקיף, שיגרם עקב השימוש במידע המופיע במסמך, וכן לנכונות התוכן של הנושאים המופיעים במסמך. עם זאת, המחבר עשה את מירב המאמצים כדי לספק את המידע המדויק והמלא ביותר.

כל הזכויות שמורות ל**ניר אדר**

אנא שלחו תיקונים והערות אל המחבר.

תכנון תוכנה

תכנון נכון חוסך זמן במימוש. כאשר אנו באים לכתוב תוכנית, לעתים רבות מתעורר הרצון להתחיל לממש את התוכנית לפני שכל החלקים השונים של התוכנית עוצבו, ולפני שיש ידיעה ברורה לאן התוכנית צריכה להגיע.

אם מתחילים לממש את התוכנית כבר בשלב זה, "ניתקע" בטיפול בבאגים ובבעיות שיצוצו במהלך הכתיבה, שעלולים לגרום לנו לשכתב חלקים גדולים מהתוכנית.

השקעה של זמן נוסף בתכנון תקצר את תהליך המימוש של התוכנית.

הצורך בתכנון קיים גם עבור מתכנתים מתחילים, וגם עבור מתכנתים מתקדמים יותר.

למשל, נביט בבעיה הבעיה:

צריך לממש פונקציה ב-C, המקבלת מערך ואת גודלו, ומחזירה את האיבר השני בגודלו במערך, תוך כדי סריקה אחת בלבד של המערך.

הקווים הכלליים של הפתרון: נסרוק את המערך, ונשמור כל העת את שני האיברים הגדולים ביותר. אם בתא הנסרק באיטרציה כלשהי של הלולאה קיים איבר הגדול משני איברים אלו, נקבע שהאיבר שעד עכשיו נשמר כהכי גדול, יהפוך לשני הכי גדול, והאיבר החדש ייזכר כאיבר הכי גדול. אם בתא באיטרציה הנוכחית איבר הגדול מהאיבר השני הכי גדול בלבד, נחליף את האיבר השני הכי גדול באיבר הכי גדול שמצאנו. בסוף התהליך נחזיר את האיבר השני הכי גדגול.

מתכנתים מנוסים יפתרו בעיה זו במהרה, אולם, מנסיוני בהוראת אנשים ששפת C חדשה להם, ראיתי כי תרגיל זה מסובך להם, וללא תכנון מוקדם של המקרים השונים שצריך לשים לב אליהם, לרוב יטעו בפתרונו.

מתכנתים מנוסים יפתרו את הבעיה הנ"ל במהרה, אולם אם יידרשו לממש עץ AVL עם תכונות נוספות מסויימות, סביר להניח שגם הם יצטרכו תכנון מוקדם על מנת לא לטעות.

לפיכך – קיים צורך לתכנן את התוכנות שאנו כותבים, ללא קשר אם אנו מתכנתים מתחילים או מנוסים. תמיד ימצאו האתגרים שנאלץ לתכנן אותם מראש.

אופי התכנון תלוי כבר באישיותו של כל מתכנת, אולם מומלץ תמיד שתהיה למתכנת תמונה כללית בראש של כל שלבי כתיבת התוכנית עד לסיומה, לפני התחלת כתיבתה.

נביט בדוגמא נוספת, בה נפתור בעיה בסיסית נוספת, ונביט כיצד יש לגשת גם לבעיות מורכבות יותר.

עלינו ליצור תוכנית שתנהל מכירת כרטיסים לבית קולנוע.

בבית הקולנוע יש NxN מקומות.

כל מבקר מציין כמה כרטיסים הוא רוצה להזמין. התוכנה בודקת האם יש שורה בה רצף כיסאות כמבוקש. אם כן, ההזמנה נרשמת והמקומות נתפסים. אחרת מודפסת הודעת שגיאה.

במבט ראשוני, משימה זו עלולה להראות מסובכת קמעה, אולם כאשר נחלק משימה זו לשלבים ניווכח כי הפתרון טריוואלי:

1. צור מערך בגודל NxN ואפס אותו.
2. כל עוד לא נגמר הקלט מהמשתמש:
   א. סרוק את המערך לחיפוש תאים פנויים כפי שהמשתמש ביקש.
   ב. אם לא נמצאו תאים, הדפס שגיאה והמתן לקלט הבא, להתחלת התהליך מחדש.
   ג. אם נמצאו תאים, סמן אותם כתפוסים והדפס את מיקומם.

כעת רואים כי מימוש כל אחד מהשלבים פשוט ביותר.

על ידי תכנון התוכנית הגדולה וחלוקתה למשימות קטנות, נקל את עבודת התכנות שלנו, ונוכל לזהות בעיות ביתר קלות.

דרישות מתוכנה

כאשר אנו כותבים תוכנה מסחרית, נרצה שתעמוד בדרישות הבאות:

1. **נכונות** – התוכנה מבצעת את המשימה. ברור כי זוהי הדרישה החשובה ביותר. ככל הידוע למחבר, אין בעולם ביקוש לתוכנות שאינם מבצעות את מה שהן אמורות לבצע.
2. **שימושית** – עומדת בדרישות הלקוח. זוהי למעשה הרחבה של הדרישה הראשונה. לא רק שנדרוש כי התוכנית שנכתוב תפעל בצורה נכונה, נדרוש גם שהיא תפעל בצורה שהלקוח שישתמש בתוכנה רצה.
3. **עמידות** – עמידה לתקלות (אירועים לא צפויים שקורים, כגון תקלות במחשב, חוסר משאבים וכו'), קלה לתחזוקה (כלומר קלה להוספת קטעי קוד חדשים, לשינוי קטעי קוד ישנים בלי כתיבה מחדש של חלקים אחרים).
4. ידידותית למשתמש – קל לעבוד עם התוכנה וקל ללמוד כיצד להשתמש בה.

בדיקות תוכנה

על מנת לבדוק שתוכנה עובדת כמו שצריך, נרצה לבדוק אותה.

ישנם שני סוגי בדיקות:

1. הרצת התוכנית מול קלטים, בדיקה שמול קלט מסוים יצא פלט מסוים.
בדיקה מסוג זה מכונה "קופסה שחורה" - Black Box – אנו בודקים את התוכנית בלי להתייחס לאופן בה היא כתובה, אלא מנסה קלטים שונים ובודקים האם התוכנית עובדת כראוי עבור קלטים אלו.

למשל – נניח שאנו כותבים תוכנת מחשבון, אז נוכל לבצע את הבדיקות הבאות: נוכל לקחת שני מספרים ולחבר אותם, לראות שהתוצאה יוצאת כראוי. נוכל לבצע חלוקה באפס, לראות שהתוכנית מציגה הודעת שגיאה כראוי וכדומה. אלו בדיקות "קופסה שחורה".

2. בדיקת הקוד – נימוקים לוגיים למה הקוד שנכתב נכון. בדיקה זו נקראת "קופסה לבנה" - White Box.
למשל – בדוגמת המחשבון, נביט בקוד ונראה שבאמת בדקנו את מקרה החלוקה באפס, או נבדוק בקוד שבפעולת חיבור לא התעלמנו מ-carry וכו'.

את הבדיקה הראשונה קל יותר לבצע, אולם אנחנו יכולים לפספס מקרי קצה שלא בדקנו. את הבדיקה השניה קשה יותר (ולפעמים לא אפשרי) לבצע.