קובץ שגיאות נפוצות - תרגיל 2 - שפת סי - RailwayPlanner

הערות כלליות

1. **שגיאות פריסבמישן**
רבים מכם קיבלתם שגיאות בפרסיבמישן וקודינג-סטייל ולא תיקנתם. היה עדיף להגיש באיחור מסוים אבל לא להגיש כך. מי שקיבל שגיאות בפריסבמישן סביר להניח שלא יעבור חלק מהטסטים האוטומטיים. בפעמים הבאות, כדאי לפנות לשעות קבלה של המתרגלים או לאב-סופורט. הרבה שגיאות מהסוג הזה ניתן לתקן די מהר.

2. **לא עוקבים אחרי ה coding style**
באתר הקורס מפורסם מסמך coding-style-example ואתם נדרשים לעקוב אחריו. כולל הכותרת בתחילת הקוד שאתם מגישים, כולל תיעוד לכל קבוע וקבוע (בין אם מוכרז בעזרת define ובין אם בעזרת const). בנוסף, אתם נדרשים לתעד גם סטרקטים (struct). תיעוד נכון של סטרקט כולל תיעוד של האלמנטים אותם הוא כולל. למשל :

```
/**
 * @struct PersonAge
 * @brief This structs includes the id of a person and his age.
 * @par id - the id of a person (9-10 digits).
 * @par age - the age of the person (in range of 0 to 120).
 */
struct PersonAge
{
    int id;
    int age;
};
```

קישור למסמך במודל -
https://moodle2.cs.huji.ac.il/nu19/pluginfile.php/444003/mod_resource/content/1/CodingStyleExampleC.c

הערות כלליות הקשורות לC

1. **שימוש בקבועים**
רבים מכם לא מקפידים כלל על קבועים.
כלל אצבע - לא צריכים להיות מספרים או מחרוזות בקוד שהן קבועות. כלומר - הדפוס לקובץ/stdout , ערך החזרה 0 או 1 של פונקציה וכו'.
אם פונקציה הצליחה ומחזירה ערך 1, הפכו את הערך 1 לקבוע ותנו לו שם ברור :
SUCCESS, SUCCESS_CODE
או כל שם אחר המבהיר באופן חד חד ערכי שהפונקציה הצליחה. כנ"ל עבור כישלון.
ניתן להגדיר בעזרת define או בעזרת const.
במקרים בהם ראינו אי שימוש בקבועים במספר רב של מקרים - הורדנו נקודות.

2. **שימוש בconst**
רבים מכם הזניחו את השימוש בconst לאורך הפונקציות.
הקפידו להשתמש בכך בתרגילים הבאים!.
אם ראינו אי שימוש בconst במספר רב של מקרים - הורדנו נקודות.

3. **פונקציות ארוכות מדי**
רבים מכם לא הקפידו על פונקציות באורך מתאים. בחלק מההגשות היו פונקציות באורכים שהגיעו ל100,150 ואף 200 שורות.
הורדנו על כך נקודות.

4. **השארת קוד "זבל"**
חלקכם שכחתם למחוק קטעי קוד שנועדו לצורכי דיבוג או הדפסות כאלה ואחרות.
הקפידו לא לעשות כך בתרגילים הבאים.

הורדנו על כך נקודות במקרים בהם היו קטעים גדולים של קוד "זבל".

5. **שמות לא ברורים עבור משתנים ופונקציות**
רבים מכם נותנים שמות לא ברורים לפונקציות ולמשתנים.
לא יכול להיות שפונקציה תקבל משתנה בשם int j ואפילו לא מופיע תיעוד מאוד ברור מה זה אומר.
במקרים מוגזמים הורדנו נקודות.

6. **קונבנציות של שמות משתנים ופונקציות**
בשפת c ביקשנו מכם לכתוב בcammel case,
כלומר שם של משתנה לא יהיה :

VariableName
Variable_name

אלא יהיה :

variableName

שיטה דומה לפונקציות.
לא הורדנו על כך נקודות למעט מקרים בודדים ונקודתיים.

7. **כתיבת תנאי if הפוכים**
חלקכם כותבים תנאי if הפוכים. כלומר, התנאי נכתב לא נכון.
דוגמא:

```
if (0 == counter)   // BAD!!!
{}

if (counter == 0) // Good.
{}
```

לא הורדנו על כך נקודות. אך אתם נדרשים להקפיד על זה.
זה פוגע משמעותית בקריאות.

8. **פונקציית main ארוכה מדי**
פונקציית המain צריכה להיות קצרה ותכליתית ככל הניתן.
פונקציית המain הארוכה ביותר הייתה 150 שורות.
אתם צריכים לשאוף למינימום שם.
את כל הבדיקות וכל הפעולות ניתן להוציא לפונקציות אחרות.
במקרים שזה היה בולט, הורדנו על כך נקודות.

9. **שימוש לא טוב במשתנים גלובליים כמשתני עזר**
משתני עזר כמו מונים, או buffer אמורים להיות לוקאליים. אין סיבה להשתמש בזה בצורה גלובלית.
נוסף על כך, אם כבר בחרתם להשתמש במשתנה גלובלי, תעדו אותו והסבירו מה תפקידו בקוד.
אחרת, זה נראה כמו שורה קוד ש"התפלחה" למקום לא מתאים.
כלל - הימנעו משימוש במשתנים גלובליים!!!
במקרים חריגים הורדנו נקודות על כך.

10. שימוש לא נכון ב EXIT_FAILURE/SUCCESS

שני הקבועים הללו שהכוונו אתכם להשתמש בהם נועדו על מנת לאפשר יציאה מהקוד בצורה מסוימת (למשל - כאשר משתמשים בפונקציה exit()).
אלו לא אמורים להיות ערכים שפונקציה מחזירה במקרים של הצלחה או כישלון.
לא הורדנו על כך נקודות אך הקפידו על כך מהתרגיל הקרוב.

11. תיעוד הפוך של פונקציות

הערה זו מתקשרת להערה למעלה לגבי המעבר על המסמך לדוגמא.
תיעוד פונקציה בשפת C נעשה מעל שורת ההצהרה של הפונקציה.
דוגמא:

```
// ------------------ GOOD  --------------------- //
/**
 * This function gets a number as an input and
 * prints to stdin the following number.
 * i.e. : num -> print(num + 1)
 * @param numToPrint - (int) input value.
 */
void printNextNum(int numToPrint)
{
    int temp = numToPrint + 1;
    printf("%d", temp);
}


// ------------------ BAD  --------------------- //

void printNextNum(int numToPrint)
/**
 * This function gets a number as an input and
 * prints to stdin the following number.
 * i.e. : num -> print(num + 1)
 * @param numToPrint - (int) input value.
 */
{
    int temp = numToPrint + 1;
    printf("%d", temp);
}
```

לא הורדנו על כך נקודות אך שימו לב לעצור את התופעה הזו.
זה נראה רע.

12. הצהרה מוקדמת (forward declaration) צריכה להיות בתחילת הקוד או לפחות לפני כל המימושים

רובכם בוחרים לכתוב forward declaration לפונקציות וזה נראה מצוין.
בודדים הסטודנטים שעושים את אחד מהדברים הבאים:

- "מצהירים מראש" רק חלק מהפונקציות.
- "מצהירים מראש" בשורה "מתגנבת" באמצע הקוד.

אם החלטתם לכתוב forward-declaration תשקיעו ותכתבו את זה מסודר ואלגנטי בתחילת הקוד.
לא הורדנו על זה נקודות.

הערות ספציפיות לתרגיל 2

1. מערך חלקים בגודל קבוע מראש

חלקכם הגדלתם מערכים בגודל 512 של char מתוך הנחה שניתן לסרוק את שורת החלקים ולקחת כל אינדקס זוגי ולשמור אותו. המימוש הזו לא סביר מכיוון שאנחנו מספקים לכם את גודל ה"מערך" האמיתי והמימוש הנכון הוא להקצות זיכרון באופן דינמי במקרה זה. לא הורדו נקודות על כך.

2. בקריאה של שורת החיבורים הסתמכתם על כך שהם באורך 1
לא הורדו נקודות על כך.

3. הפרדה לפונקציה שבונה את הטבלה ופונקציה אחרת שממלאת את הטבלה
החלוקה לשתי פונקציות לא הייתה הכרחית. אך לא הורדנו נקודות על כך.

4. **מימושים רקורסיביים**
כפי שהובהר בפורום ובדרישתנו בתרגיל לזמן ריצה מסוים, מימוש האלגוריתם היה צריך להיעשות באופן דינמי ולא באופן רקורסיבי. מימושים רקורסיביים איבדו מספר נקודות מכיוון שלא עונים על דרישת זמן הריצה. אם סטודנט חושב שהוא ענה על דרישת זמן הריצה למרות המימוש הרקורסיבי, יוכל לערער ונבדוק את המקרה שנית.