2. חלק "יבש"

2.1.1 מציאת שגיאות:

בחלק זה נתבקשנו למצוא 8 שגיאות מסוגים שונים בקוד שקיבלנו.

הקוד:

```
3   #include <stdlib.h>
4   #include <string.h>
5   #include <assert.h>
6
7
8   char *stringduplicator(char *s, int times) {
9       assert(!s);
10      assert(times > 0);
11      int LEN = strlen(*s);
12      char *out = malloc( _Size: LEN * times);
13      assert(out);
14      for (int i = 0; i < times; i++) {
15      out = out + LEN;
16      strcpy(out, s);
17      }
18      return out;
19  }
```

כעת, נפרט את הטעויות שמצאנו בקוד, את סוג הטעות ואת תיקון הטעות, ולבסוף נציג את הקוד המתוקן:

שגיאות תכנות במימוש הפונקציה הנתונה:

1. **שורה 9; שימוש לא מתאים בassert:**

```
9        assert(!s);
```

לדעתי היה פה נסיון לקבוע שהמחרוזת שנשלחת לפונקציה איננה מצביע לNULL, אבל בעצם הפונקציה בודקת שהמחרוזת שאנו שולחים אליה היא לא עצמה (?) על ידי הassert הזה. לכן הassert תמיד יכשל.

תיקון:

נחליף את הassert בבדיקת תנאי רגילה (והגיונית) באמצעות if.

הערה: נצטרך לבחור ערך חזרה כלשהו במקרה של שגיאה, אני בחרתי בהשמת המחרוזת להיות מחרוזת ריקה, ואז מתבצעת "העתקה ריקה":

```
8        if (s == NULL)
9            s="";
```

2. **שורה 10; שימוש לא מתאים בassert:**

```
10       assert(times > 0);
```

הפונקציה בודקת שמספר הפעמים שהמשתמש רוצה לשכפל את המחרוזת הוא חיובי, אבל אכן יכול לקרות מצב שנשלח ערך אי-חיובי (שלילי או שווה ל-0) לפונקציה, ואז במצב DNDEBUG הassert ייפול ותיווצר שגיאה בהמשך התכנית.

תיקון:

נחליף את הassert בבדיקת תנאי רגיל באמצעות if.

הערה: נצטרך לבחור ערך חזרה כלשהו במקרה של שגיאה, אני בחרתי בהחזרת הודעת שגיאה במקרה שהערך שלילי, והחזרת מחרוזת ריקה במקרה שהערך שווה בדיוק 0:

```
10       if (times == 0)
11           return "";
12       if (times < 0)
13           return "Error! Number of duplicates must be positive!";
```

3. **<u>שורה 11; שליחת ערך לא נכון לפונקציה strlen:</u>**

```
11        int LEN = strlen(*s);
```

הפונקציה strlen מקבלת מחרוזת, ומחזירה את אורכה.

אולם, כאן הפונקציה איננה מקבלת מחרוזת, אלא תו אחד בלבד: המשתנה s הוא מסוג *char, ולכן מלבד עצם היותו מחרוזת, הוא גם מצביע לchar (לתו הראשון במחרוזת). לכן, כאשר הוא נשלח עם האופרטור ' * ' לפונקציה, הפונקציה מקבלת רק את התו הראשון במחרוזת, ותחזיר לנו אורך 1 במקום אורך המחרוזת האמיתי.

תיקון:

נוריד את השימוש באופרטור ' * ', ונשלח את כל המחרוזת s לפונקציה strlen:

```
11        int LEN = strlen(s);
```

4. **<u>שורה 13; שימוש לא מתאים בassert:</u>**

```
13        assert(out);
```

הפונקציה בודקת שout שהקצאנו באמצעות malloc הוא לא NULL באמצעות assert, אך זהו שימוש לא מתאים בassert, שכן זו לא הנחה "ודאית" (הקצאה דינמית יכולה להיכשל). במידה והקוד הוא במצב DNDEBUG וההקצאה הדינמית תיכשל – התכנית תדלג על שורת הבדיקה ותמשיך הלאה (ותיכשל בעצם כבר בשורה שאחריה כאשר היא תנסה לעבוד על out שיהיה מצביע לNULL).

תיקון:

נחליף את הassert בבדיקת תנאי רגיל באמצעות if.

הערה: נצטרך לבחור ערך חזרה כלשהו במקרה של שגיאה, אני בחרתי בהחזרת הודעת שגיאה:

```
13        if (out == NULL)
14            return "Error! Memory allocation failed!";
```

5. <u>העתקה לא נכונה של המחרוזת:</u>

```
for (int i = 0; i < times; i++) {
    out = out + LEN;
    strcpy(out, s);
}
```

פעולות ההעתקה והשכפול של המחרוזת המקורית s לתוך המחרוזת החדשה out נעשית על ידי הלולאה הזו, וזה לא עובד. המחשבה מאחורי הלולאה היא שבכל איטרציה out יתקדם LEN צעדים קדימה, ואז באמצעות strcpy נכניס את תוכן המחרוזת s לתוך LEN המקומות הבאים (שמתחילים במצביע העדכני בout. אולם אריתמטיקת המצביעים כאן נכשלת, ובעצם רק LEN התווים הראשונים של out מקבלים את ערכי התווים של המחרוזת s.

תיקון:

נרשום את הלולאה באופן הבא:

```
for (int i = 0; i < times; i++) {
    strcpy( _Dest: out + (LEN * i), s);
}
```

<u>הערה: לאחר שהורדתי את השימוש בassert-ים, ניתן למחוק את הinclude של ספריית assert.h</u>

שגיאות קונבנציה במימוש הפונקציה הנתונה:

1. **שורה 8; שם הפונקציה לא כתוב כנדרש:**

שם הפונקציה כתוב כולו באותיות קטנות, וזה לא תקין. שם הפונקציה צריך להכיל אותיות קטנות בלבד ואות גדולה בתחילת כל מילה שאינה ראשונה.

תיקון:

נחליף את ה-d בתחילת המילה duplicator ב-D:

```
7    char *stringDuplicator(char *s, int times) {
```

2. **שורה 11; שם המשתנה LEN לא כתוב כנדרש:**

```
11        int LEN = strlen(*s);
```

שם המשתנה LEN כתוב כולו באותיות גדולות, וזה לא תקין. שם הפונקציה צריך להכיל אותיות קטנות בלבד.

תיקון:

נחליף את האותיות הגדולות באותיות קטנות:

```
14        int len = strlen(s);
```

3. **שורות 14-17; אין שימוש בהזחות בבלוק לולאת הfor:**

```
14    for (int i = 0; i < times; i++) {
15    out = out + LEN;
16    strcpy(out, s);
17    }
```

תיקון:

נבצע הזחות בתוך בלוק לולאת הfor:

```
18    for (int i = 0; i < times; i++) {
19        strcpy( _Dest: out + (len * i), s);
20    }
```

4. שימוש בקיצור אסור למילה string:

לפי הוראות הקורס, הקיצור המותר עבור string הוא str, אך בפונקציה זו הקיצור בו משתמשים הוא s, וזה אסור.

תיקון:

נחליף את s ב-str בכל מקום בפונקציה:

```
7   char *stringDuplicator(char *str, int times) {
8       if (str == NULL){
9           str="";
10      }
11      if (times == 0){
12          return "";
13      }
14      if (times < 0){
15          return "Error! Number of duplicates must be positive!";
16      }
17      int len = (int)strlen(str);
18      char *out = malloc( _Size: sizeof(*out)*(len * times));
19      if (out == NULL)
20          return "Error! Memory allocation failed!";
21      for (int i = 0; i < times; i++) {
22          strcpy( _Dest: out + (len * i), str);
23      }
24      return out;
25  }
```

2.1.2 גרסה מתוקנת של הפונקציה:

```c
char *stringDuplicator(char *str, int times) {
    if (str == NULL){
        str="";
    }
    if (times == 0){
        return "";
    }
    if (times < 0){
        return "Error! Number of duplicates must be positive!";
    }
    int len = (int)strlen(str);
    char *out = malloc( _Size: sizeof(*out)*(len * times));
    if (out == NULL)
        return "Error! Memory allocation failed!";
    for (int i = 0; i < times; i++) {
        strcpy( _Dest: out + (len * i), str);
    }
    return out;
}
```