מבוא לרקורסיה בשפת C

מאת ניר אדר (UnderWarrior)

עבור אנשים הניגשים ללמוד לראשונה את שפת C נושא הרקורסיה נחשב כאחד הנושאים הקשים. עם זאת עבור המתכנת המיומן, רקורסיה אינה אמורה להציב קושי מיוחד, ולהפך – היא כלי שיכול לתת פתרון אלגנטי לבעיות. לרוב כאשר ניגשים להבין רקורסיה שוכחים מספר נושאים בשפה שחסרונם גורם לרקורסיה להראות כמו איזה black voodoo לא ברור.

מאמר זה פותח סדרת מאמרים שיציגו את נושא הרקורסיה. במאמר היום כמעט לא תראו פונקציות רקורסיביות. נתחיל בהצגה של מרכיבי שפת C המאפשרים לנו ליצור רקורסיה, ובמאמר הבא נציג כיצד בעזרת מרכיבים אלה כותבים פונקציות רקורסיביות.

לפני הכל צריך לענות על השאלה הבאה: **מה זאת פונקציה רקורסיבית?**

פונקציה רקורסיבית היא פונקציה המכילה קריאה לעצמה. **קוד רקורסיבי** הוא קוד המכיל פונקציה רקורסיבית. נביט לדוגמא בתוכנית הבאה:

```
#include <stdio.h>

void star()
{
      putchar('*');
      star();
}


int main()
{
      star();
      return 0;
}
```

מה יקרה כאשר נריץ תוכנית זו? הפונקציה ()main קוראת עם תחילת ריצת התוכנית לפונקציה ()star. הפונקציה star מדפיסה כוכב אחד, ואז קוראת לעצמה. שוב יודפס כוכב יחיד, ושוב הפונקציה תקרא לעצמה. עד אינסוף? לא – בשלב מסויים כאשר נריץ תוכנית זו יגמר הזיכרון הפנוי על המחסנית והתוכנית תסתיים. ברור שהמצב בו התוכנית תקרוס אינו המצב הרצוי, וגם המצב התאורטי בו זכרון בלתי מוגבל והתוכנית רצה עד אינסוף – גם הוא לרוב אינו רצוי.

עד כאן לא נראה שהפונקציה הזו שימושית יותר מדי (אלא אם ברצוננו לגרום לתוכנית שלנו לקרוס ☺). איך נהפוך את הפונקציה לשימושית? נוסיף לפונקציה הרקורסיבית שלנו **תנאי עצירה** – תנאי שכאשר הוא יקרה, הפונקציה לא תקרא שוב לעצמה – ותסתיים.

לדוגמא:

```
#include <stdio.h>

void star(int n)
{
      if (n == 0) return;
      putchar('*');
      star(n-1);
}


int main()
{
      star(5);
      return 0;
}
```

מה יקרה כעת? תנאי העצירה שלנו הוא:

```
      if (n == 0) return;
```

כעת יודפסו 5 כוכביות, ולאחר מכן לא נקרא לרקורסיה והיא תסתיים. נשים לב שכאשר אנחנו מגיעים אל תנאי העצירה, אנחנו לא יוצאים לגמרי משרשרת קריאות הפונקציות – אנחנו פשוט מפסיקים להכנס לעומק הרקורסיה. אם נקודה זו לא ברורה אל תדאגו – זו הנקודה עליה נדבר ונרחיב במאמר זה.

קריאה לפונקציה, חזרה מפונקציה ומשתנים לוקלים – איך רקורסיה עובדת?

הדגש שלנו היום הוא על הבנת מנגנונים בשפת C בהם רקורסיה משתמשת. הנקודה הראשונה שנרצה להדגיש היא מנגנון העברת הפרמטרים אל פונקציה.

לאחר מכן נרצה להבין מעט כיצד קריאה וחזרה לפונקציות עובדות

נושא ראשון: פרמטרים של פונקציה

הפרמטרים שכל פונקציה מקבלת הם <u>משתנים לוקליים</u>, שהם עותקים של הערכים שהועברו אל הפונקציה.

נביט בקוד <u>הלא רקורסיבי</u> הבא:

```
void f1(int x)
{
      printf("x = %d\n", x);
      x = 8;
      printf("x = %d\n", x);
}
int main()
{
      int x = 5;
      f1(x);
      printf("x = %d\n", x);
      return 0;
}
```

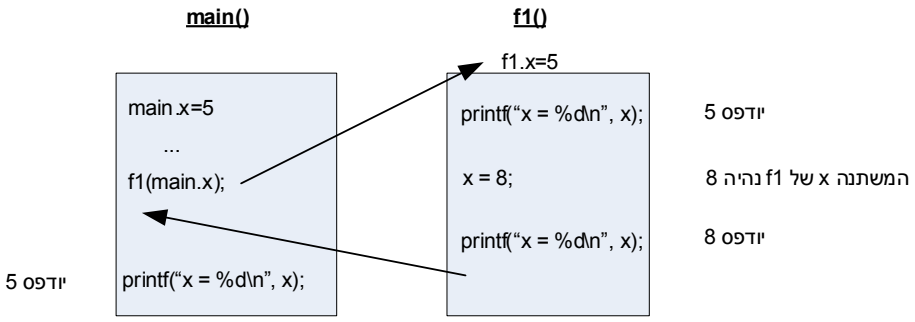
אנחנו רוצים להבהיר את הנקודה הבאה:

ל-f1() יש משתנה בשם int x, שהוא שונה מהמשתנה int x של main(). שינויים במשתנה של הפונקציה f1() לא ישפיעו על main().

בואו נראה מה יקרה כאשר נריץ את הקוד: הפלט יהיה:

```
x = 5
x = 8
x = 5
```

נצייר את מה שקורה בתוכנית זו בצורה גרפית. בשרטוט אנחנו מסמנים main.x כאשר מדברים על המשתנה x שהוגדר בפונקציה ()main.



השלבים:

- בהתחלה המשתנה x של main שווה ל-5.
- כאשר אנחנו קוראים לפונקציה ()f1 (הקריאה מסומנת על ידי החץ) נוצר משתנה חדש - המשתנה x של ()f1, והוא מקבל את הערך של x של ()main.
- אנחנו יכולים לשנות את הערך של f1.x כרצוננו. כאשר אנחנו חוזרים ל-main (החזרה מסומנת על ידי החץ השני), המשתנה f1.x נעלם, וכשאנחנו מדפיסים את x אנחנו שוב מדפיסים את המשתנה x של ()main, שערכו כאמור לא השתנה והיה כל הזמן 5.

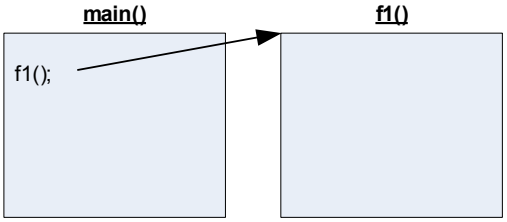
כאשר ביצוע של פונקציה מסתיים התוכנית חוזרת אל הפונקציה שקראה לה. פונקציה מסתיימת כאשר הגענו לשורה בה כתוב return או כאשר הגענו לשורה האחרונה בתוכנית.

נביט בקוד הלא רקורסיבי הבא:

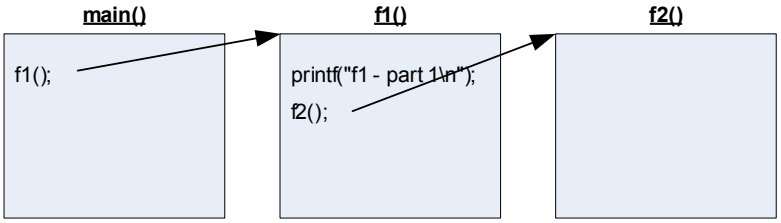
```
void f1()
{
      printf("f1 - part 1\n");
      f2();
      printf("f1 – part 2\n");
}

void f2()
{
      printf("f2 – part 1\n");
      return;
}


int main()
{
      f1();
      return 0;
}
```

השאלה שנשאל את עצמנו: מה התוכנית תדפיס כפלט? ומה שאנחנו בעצם רוצים לראות: כיצד מתנהלת זרימת התוכנית?

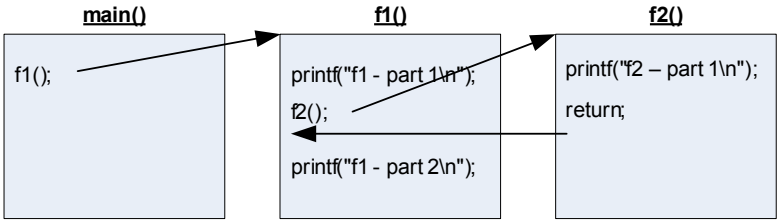
ראשית ()main קוראת ל-()f1.



()f1 מדפיסה את השורה f1 - part 1 וקוראת ל-()f2.



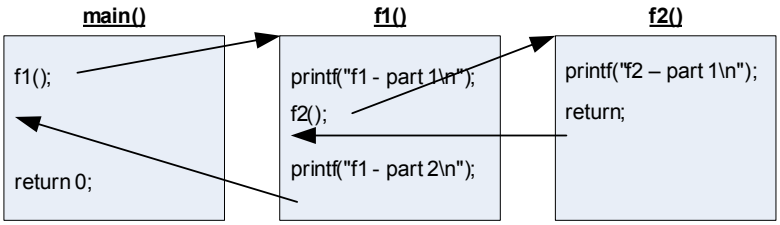
()f2 מדפיסה את השורה f2 - part 1 ולאחר מכן חוזרת. לאן היא חוזרת? זו הנקודה החשובה – הפונקציה חוזרת אל הפונקציה שקראה לה, אל השורה שאחרי הקריאה – כלומר, במקרה זה, אל ()f1 אל השורה המדפיסה f1 - part 2.



לאחר מכן ()f1 מסתיימת. גם אם לא ציינו במפורש את פקודת ה-return, כאשר פונקציה מסתיימת היא חוזרת אל הפונקציה שקראה לה, במקרה זה ל-()main. מכיוון שאין הוראות נוספות בתוכנית, התוכנית מסתיימת והפלט שלה הוא:

```
f1 - part 1
f2 - part 1
f1 - part 2
```

השרטוט הסופי:



סיכום - פונקציות

נסכם את התופעות שראינו:

- **הפרמטרים שכל פונקציה מקבלת הם <u>משתנים לוקליים</u>, שהם עותקים של הערכים שהועברו אל הפונקציה.**
- **כאשר ביצוע של פונקציה מסתיים התוכנית חוזרת אל הפונקציה שקראה לה. פונקציה מסתיימת כאשר הגענו לשורה בה כתוב return או כאשר הגענו לשורה האחרונה בתוכנית.**

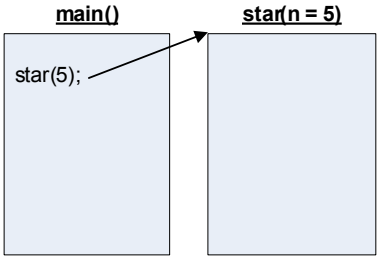
אחרי הקדמה זו נדגיש את הנושא המרכזי: פונקציה רקורסיבית מתנהגת בדיוק באותו אופן כמו פונקציה רגילה! כאשר אנחנו קוראים לפונקציה רקורסיבית אנו בעצם קוראים לפונקציה חדשה (שבמקרה יש לה את אותו שם כמו הפונקציה שלנו). לעותק זה של הפונקציה <u>יש משתנים משלו</u>. כאשר פונקציה רקורסיבית חוזרת (כאשר היא הגיעה אל תנאי העצירה), היא חוזרת אל הפונקציה שקראה לה. זו יכולה להיות היא עצמה במידה ואנחנו בתוך הרקורסיה.

הדוגמא הבאה, שהיא השינוי האחרון שנציג של פונקצית star תדגים זאת.

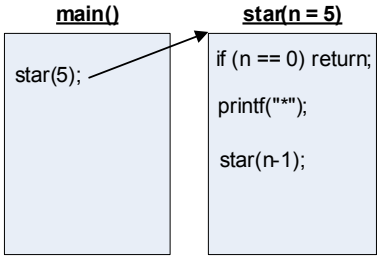
```
#include <stdio.h>

void star(int n)
{
      if (n == 0) return;
      printf("*");
      star(n-1);
      printf("%d", n);
}


int main()
{
      star(5);
      return 0;
}
```

שוב נתעניין בפלט של התוכנית. גם במקרה זה – המטרה שלנו במעקב אחרי הפלט היא להבין כיצד הרקורסיה מתנהלת.

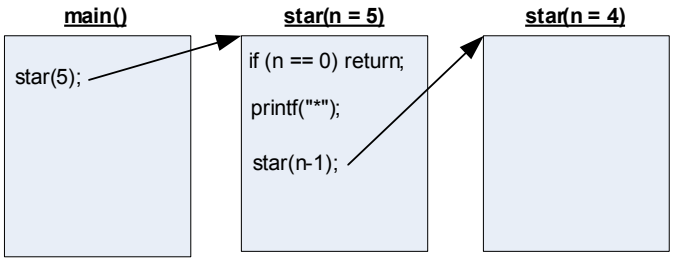
main() קורא ל-star() עם הפרמטר 5. ערך זה נכנס לתוך המשתנה הלוקלי n של הפונקציה star.



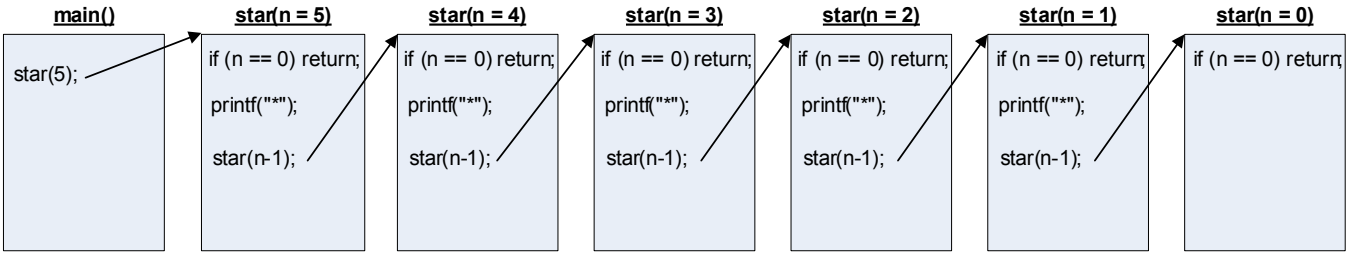
star() בודקת את תנאי העצירה, מכיוון ש-n הינו 5 ולא 0, התנאי לא מתקיים. הפונקציה מדפיסה כוכבית בודדת, וקוראת לעצמה עם הערך n-1, כלומר 4.



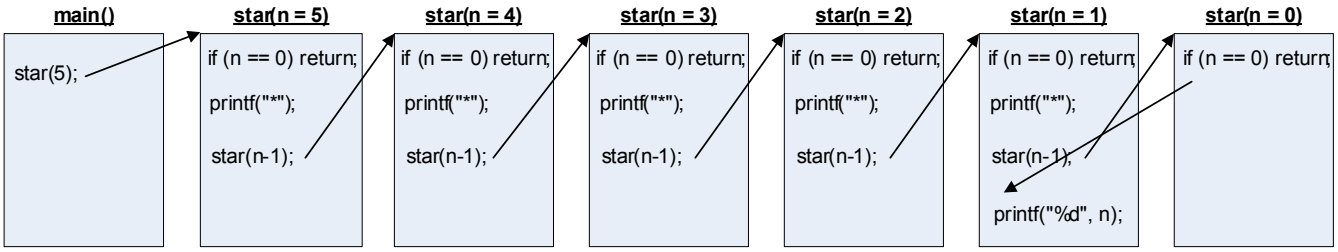
בשלב זה מתחיל עותק נוסף של הפונקציה. עותק זה מתחיל לרוץ מתחילת הפונקציה. יש לו משתנה לוקלי n משלו, שערכו 4.



באופן דומה, הפונקציה בה n=4 קוראת קריאה רקורסיבית לפונקציה עם n=3, הפונקציה עם n=3 קוראת קריאה רקורסיבית עם n=2, הפונקציה עם n=2 קוראת קריאה רקורסיבית עם n=1 ולבסוף הפונקציה עם n=1 קוראת קריאה רקורסיבית עם הערך n=0.



בשלב זה תנאי העצירה מתקיים. הפונקציה בה n=0 בודקת את תנאי העצירה, רואה שהוא לא מתקיים וחוזרת. לאן היא חוזרת? לפונקציה שקראה לה, הפונקציה בה n=1, אל השורה מיד אחרי הקריאה הרקורסיבית.



כאן הקטע בו אנשים מתבלבלים, ואני מקווה שאחרי השרטוט שראינו האמירה הבאה תהיה ברורה לכם: כעת יודפס על המסך המספר 1. אנחנו בעותק של הפונקציה בו n=1, והפקודה הבאה אומרת להדפיס את n.

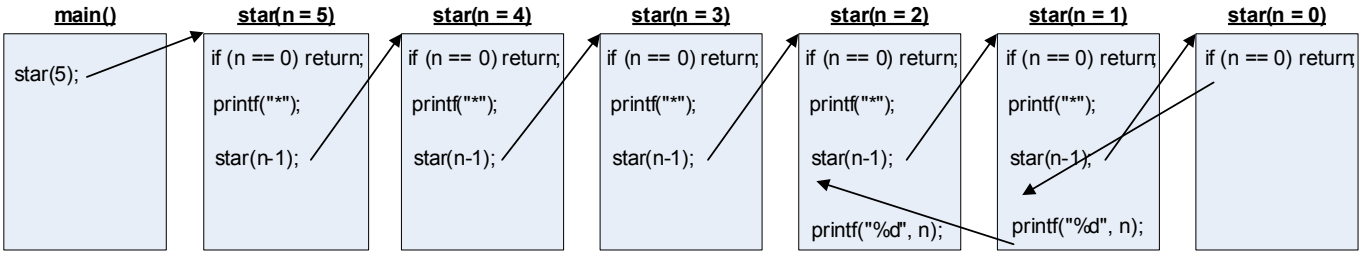
איפה הטעות הנפוצה? נראה רגע שוב את הקוד של star:

```
void star(int n)
{
      if (n == 0) return;
      printf("*");
      star(n-1);
      printf("%d", n);
}
```

הפעם הראשונה שמגיעים ל-return היא כאשר n=0. הרבה אנשים טועים וחושבים מחשבה בסגנון הבא: "return יוצא מ-star, ובגלל זה חוזרים ל-main והתוכנית נגמרת". הטעות היא בחוסר ההבנה שכל קריאה רקורסיבית היא קריאה לפונקציה חדשה בפני עצמה. כשהפונקציה הרקורסיבית בה n=0 מסתיימת על ידי return, אנו חוזרים לפונקציה שקראה לה, שזוהי star בה n=1, ולא הפונקציה main.

נמשיך בניתוח פעולת התוכנית.

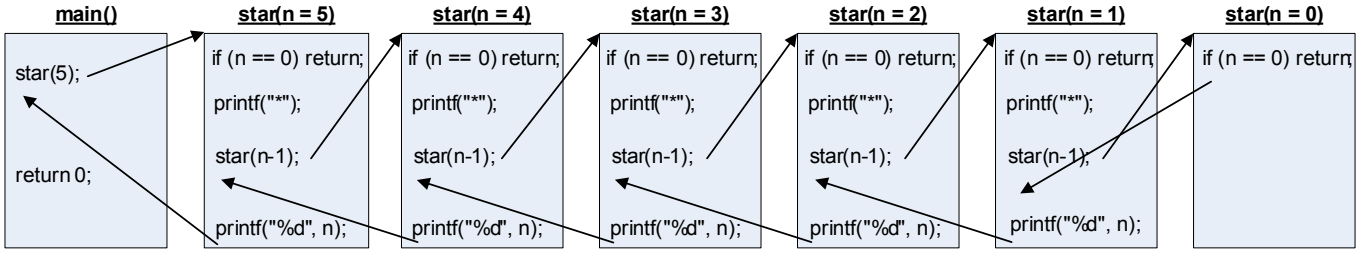
הפונקציה star (עם n=1) מסתיימת, ולכן אנחנו חוזרים לפונקציה שקראה לנו. הפונקציה שקראה לנו היא star בה המשתנה n=2. אנחנו חוזרים אל מיד אחרי הקריאה הרקורסיבית.



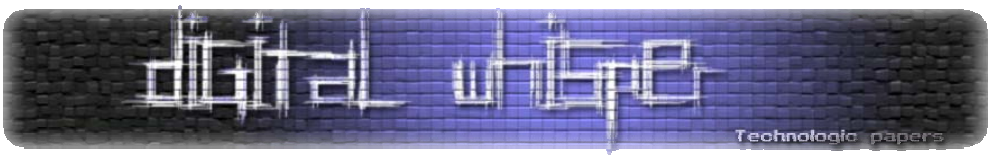
על המסך מודפס 2, שזהו הערך של n בעותק הנוכחי של הפונקציה. באופן דומה, מודפסים 3, 4, 5. הפלט הסופי של הפונקציה הוא:

```
******12345
```

השרטוט הסופי של הקריאות:



כמו שהצגנו בהקדמה הארוכה בנושא הפונקציות – ניתן לראות כי פונקציה רקורסיבית מתנהגת לחלוטין כמו פונקציה רגילה.



סיכום

רקורסיה היא כלי שיאפשר לנו לפתור בעיות רבות. הרעיון של אלגוריתמים רקורסיבים מזכיר מאוד אינדוקציה מתמטית:

1. **בסיס (/תנאי עצירה)**: נפתור את הבעיה עבור המצב הפשוט ביותר – **המצב הטריויאלי**.
2. **צעד**: נניח שהפונקציה שלנו יודעת לטפל במצב פשוט יותר מהנוכחי, ונגרום לה להיות נכונה עבור קלט מסובך יותר. הצעד צריך לקרב אותנו אל המצב הטריויאלי.

רקורסיה משתמשת במנגנונים שהצגנו במאמר זה לצורך פעולה נכונה. במאמר הבא נציג כיצד אנו פותרים בעיות שונות באמצעות פונקציות רקורסיביות.