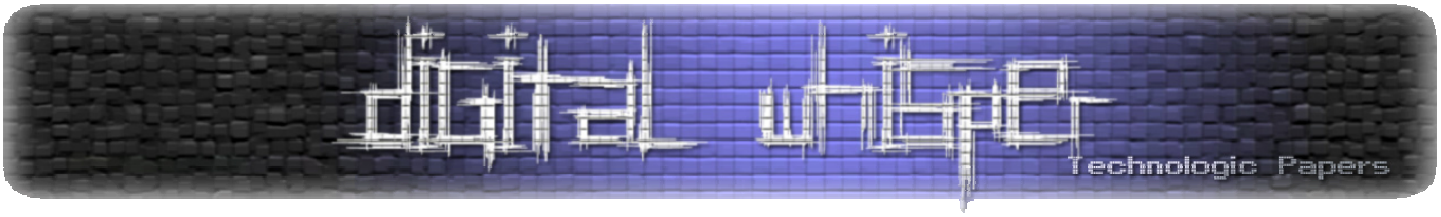


אלגוריתמים רקורסיביים

מאת ניר אדר (UnderWarrior)

פתיחה

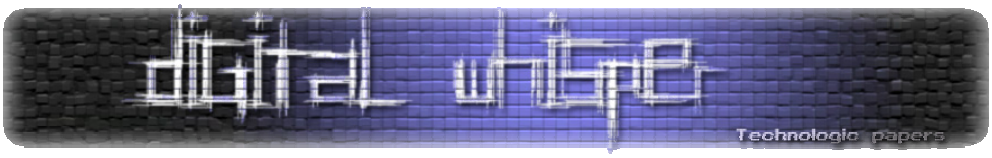
בגליון הראשון של Digital Whisper הצגתי לכם הקדמה לרקורסיה – מהם המנגנונים המאפשרים רקורסיה בשפת C. הבטחתי המשך לאותו המאמר והנה הוא מגיע. היום אציג לכם פרקטיקה – איך משתמשים ברקורסיה ככלי לפתירת בעיות. רקורסיה היא כלי שיאפשר לנו לפתור בעיות בקלות יחסית ובאופן קצר מאשר פיתרון איטרטיבי (פתרון ללא רקורסיה). לא כל בעיה מתאימה לרקורסיה, אך יש לא מעט בעיות שהפתרון הרקורסיבי שלהן יהיה שורות ספורות, לעומת פתרון איטרטיבי מסובך. מצד שני פתרונות רקורסיביים אינם "פתרונות קסם" וגם לפתרונות רקורסיביים יש חסרונות (א. שימוש ביותר זכרון, לרוב, מאשר פתרונות איטרטיביים. ב. באגים ו-stack overflows שנובעים מטעויות ברקורסיה שקל ליפול בהן). בסופו של דבר מומלץ להכיר רקורסיה בתור כלי נוסף בו ניתן להשתמש בתוכניות שלנו.

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.

אדגים שימוש ברקורסיה לפתרון בעיות שונות בשפת C. כמו שתוכלו לראות, ניתן להשתמש ברקורסיה במגוון שימושים רב, ובשילוב עם כל אספקט אחר בשפה, למעשה. נתחיל ברצף דוגמאות. בכל דוגמא אציג אספקט חדש של כתיבת אלגוריתמים רקורסיביים בשפת C וכך נסקור את הנושא.

חשוב לדעת שכדי להתמחות ברקורסיה עליכם לקודד בעצמכם, ולקודד הרבה בעיות, על מנת לקבל את האינטואציה הדרושה. מאמר זה מכיל טעימה מעולם האלגוריתמים הרקורסיביים. כדי להגיע לרמה סבירה, אמליץ לכם לפתור לפחות עשרות תרגילי רקורסיה פשוטים.



דוגמא ראשונה – עצרת

נרצה לכתוב פונקציה המחשבת עצרת. הפונקציה תקבל מספר n ותחזיר את העצרת שלו (int).

השאלה הראשונה שנשאל את עצמנו כשאנחנו מנסים לפתור בעיה רקורסיבית, היא "מה הוא הקלט הפשוט ביותר לפונקציה, הקלט שעבורו בכלל לא צריכים לעשות חישוב, ואפשר לדעת מה התוצאה?"

במקרה של עצרת, אנחנו יודעים באופן טריויאלי לחשב עצרת של 0. עצרת של 0 הינה 1.

```
int azeret(int n)
{
	if (n == 0) return 1;
}
```

שורה זו היא הבסיס של הרקורסיה.

כעת מגיע החלק שנראה בתחילה מעט כמו קסם - הצעד: נניח כי הפונקציה azeret() יודעת לפתור את הבעיה עבור n-1, ונרצה לפתור את הבעיה עבור n. איך נעשה זאת? בהנתן ש-azeret(n-1) יודעת לחשב את העצרת של n-1, כל שעלינו לעשות הוא להכפיל את התוצאה ב-n, ולהחזיר אותה.

```
int azeret(int n)
{
	if (n == 0) return 1;
	return azeret(n-1) * n;
}
```

זהו, הפונקציה מוכנה! תוכלו להפעיל את הפונקציה, להעביר אליה n כלשהו ולראות שהיא אכן חישבה את העצרת שלו. לדוגמא, תוכנית מלאה:

```
#include <stdio.h>

int azeret(int n)
{
	if (n == 0) return 1;
	return azeret(n-1) * n;
}

int main()
{
	printf("5! = %d\n", azeret(5));
	return 0;
}
```

מה בעצם קרה כאן? ננתח את הפונקציה:

- עבור n=0 הפונקציה מחזירה תוצאה נכונה. (ההנחה שהנחנו והמקרה בו טיפלנו ללא שום ריאה רקורסיבית).
- עבור n=1, אנחנו לוקחים את החישוב azeret(0), ומכפילים אותו ב-1. הראנו כי אנחנו יודעים לחשב את העצרת של 0, ולכן אנחנו הרגע חישבנו את העצרת של 1.
- באופן דומה, עבור n=2, אנחנו לוקחים את החישוב azeret(1) ומכפילים אותו ב-2. הראנו כי אנחנו יודעים לחשב את העצרת של 1, ולכן אנו רואים שאנחנו יודעים לחשב גם את העצרת של 2.
- אפשר להמשיך באופן דומה לכל n.

כל קריאה רקורסיבית קראה לעותק חדש של הפונקציה הרקורסיבית רק עם משימה פשוטה יותר ויותר. כאשר n=0 המשימה היתה טריויאלית ונפתרה, והתחלנו לעלות ברקורסיה ולפתור את הפונקציות אחת אחרי השניה.

דוגמא שניה – סדרת פיבונצי

לעתים יש לנו יותר מתנאי בסיס אחד הדרוש על מנת שהרקורסיה תעבוד כמו שצריך.

סדרת פיבונצי היא סדרה מאוד ידועה, ההולכת כך: האיבר הראשון בסדרה, a_0, הינו 0. האיבר השני בסדרה, a_1, הינו 1. כל איבר בהמשך הינו הסכום של שני האיברים הקודמים לו.

$$a_0 = 0$$
$$a_1 = 1$$
$$a_n = a_{n-1} + a_{n-2}$$

לדוגמא, האיברים הראשונים בסדרה הם:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, …
```

כעת נרצה לכתוב פונקציה רקורסיבית הקשורה לסדרת פיבונצי. נרצה לכתוב פונקציה המקבלת מספר n ומחזירה לנו את המספר ה-n בסדרת פיבונצי. לדוגמא, אם $n = 0$ - אנחנו מחזירים 0, ואם $n = 6$ נחזיר 8.

<u>הבסיס של הרקורסיה</u>: אם $n=0$ - אנחנו מחזירים 0, זהו ערכו של האיבר הראשון. אם $n=1$ אנחנו נחזיר 1, ערכו של האיבר השני בסדרה:

```
int fib(int n)
{
      if (n == 0) return 0;
      if (n == 1) return 0;
}
```

<u>צעד הרקורסיה</u>: בהנתן n, נניח כי אנחנו יודעים לחשב את האיברים a_{n-1} ו- a_{n-2}, על ידי שימוש בקריאות אל fib(n-1) ו-fib(n-2) בהתאמה. האיבר ה-n הוא פשוט חיבור של שני איברים אלה:

```
int fib(int n)
{
      if (n == 0) return 0;
      if (n == 1) return 0;
      return fib(n-1)+fib(n-2);
}
```

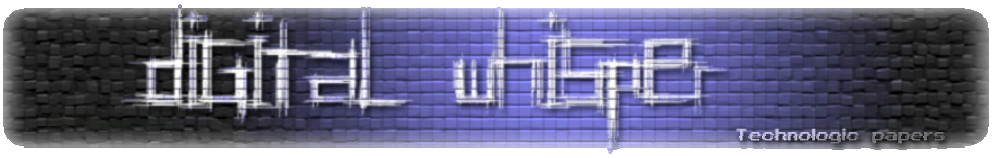
סיימנו את הפונקציה, והיא מחשבת כעת את מספר פיבונצי ה-n בסדרה.

שימו לב כי צעד הרקורסיה נותן לנו כאן רמז לכך שדרושים שני תנאי בסיס: צעד הרקורסיה מניח שבכל רגע ידועים לנו הערכים של 2 איברים קודמים. על מנת שהרקורסיה תעבוד – הנחה זו צריכה להתקיים מתישהו. אם אתם מגיעים למצב בו אינכם יודעים כמה תנאי התחלה אתם צריכים – ניסיון לחשוב על צעד הרקורסיה ועל המידע הנדרש עבורו יכול לתת לכם קריאת כיוון.

ניתן לכתוב את הפונקציה גם כך:

```
int fib(int n)
{
      if (n < 2) return n;
      return fib(n-1)+fib(n-2);
}
```

לכאורה יש לנו כאן רק תנאי עצירה אחד, אך למעשה זו פשוט דרך מקוצרת לרשום את שני תנאי העצירה הקודמים – <u>ברקורסיה זו אנו חייבים את שני תנאי העצירה</u>. (הסיבה לכך היא שהסדרה מוגדרת בעזרת 2 איברים – ולכן חייבים לפחות 2 הנחות).



מערכים ורקורסיה – מציאת האיבר הגדול ביותר במערך

צורת העבודה שלנו לא משתנה כאשר אנחנו באים לעבוד עם מערכים. גם במקרה כזה אנחנו מדברים על תנאי עצירה ועל צעד הרקורסיה. כאשר משלבים רקורסיה עם מערכים, המקרה הטריויאלי לרוב יהיה כאשר הפונקציה מקבלת מערך בגודל תא אחד. (מקרים אחרים יכולים להיות מערך בגודל 0).

ניקח כדוגמא כתיבת פונקציה בשם max_arr המקבלת מערך ואת גודלו, ומחזירה את האיבר הגדול ביותר במערך.

בסיס הרקורסיה: אם במערך תא אחד בלבד, נחזיר את הערך המופיע בתא זה.

```
int max arr(int arr[], int n)
{
      if (n == 1) return arr[0];
}
```

צעד הרקורסיה: נביט במצב שבו במערך ישנם n תאים. הנחת האינדוקציה שלנו כי max_arr יודעת למצוא את המקסימום של מערך בגודל n-1 תאים. נפעיל את הפונקציה על כל איברי המערך, למעט האיבר הראשון. נכניס את התוצאה למשתנה בשם max. max מכיל את המקסימום של שאר איברי המערך. נשווה את max לאיבר הראשון במערך, ונחזיר את המקסימלי מביניהם:

```
int max_arr(int arr[], int n)
{
      int max;
      if (n == 1) return arr[0];

      max = max_arr(arr+1,n-1);
      if (max > a[0]) return max;
      return a[0];
}
```

היינו יכולים בצורה מקבילה להחליט שכל פעם נפעיל את הפונקציה על כל איברי המערך, למעט האיבר האחרון, ונשווה את המקסימום של שלהם מול האיבר האחרון. במקרה כזה הפונקציה הרקורסיבית היתה נראית כך:

```
int max_arr(int arr[], int n)
{
      int max;
      if (n == 1) return arr[0];

      max = max_arr(arr,n-1);
      if (max > a[n-1]) return max;
      return a[n-1];
}
```

עבור שתי הדוגמאות האחרונות אני רוצה מעט להתעמק על הקריאה הרקורסיבית על-מנת שהיא תהיה ברורה יותר. במקרה הראשון כתבנו:

```
	max_arr(arr, n-1);
```

כלומר, אנחנו מעבירים את המערך, החל מהתא הראשון, ומציינים לקריאה הרקורסיבית שגודלו הוא n-1, כלומר בפועל אנחנו מעבירים את כל האיברים חוץ מהאחרון. בפונקציה השניה אנחנו מעבירים מצביע אל התא השני, ושוב מציינים n-1 איברים, כלומר כל האיברים חוץ מהאיבר הראשון:

```
	max_arr(arr+1, n-1);
```

אנחנו מסתמכים בכל המניפולציה הזו על העובדה שגודל המערך מועבר על ידינו לפונקציה, וכן המצביע להתחלתו. על ידי שינוי של פרמטרים אלו בקריאה הרקורסיבית, אנחנו מעבירים (מבחינה לוגית) "תת מערך" לקריאה הרקורסיבית.

רקורסיה ומחרוזות

תרגילים מומלצים למתכנתים הלומדים לראשונה על רקורסיה כוללים פעמים רבות מימוש אלגוריתמים רקורסיביים שונים על מחרוזות. מכיוון שלמחרוזות יש בדרך כלל משמעויות (מילים, משפטים וכדו') יש פתח לאלגוריתמים רבים ויצירתיים הקשורים למחרוזות. תרגיל התחלתי מוצלח הוא לממש את פונקציות הספריה השונות לטיפול במחרוזות (strcpy(), strcat(), strcmp(), strlen()) באמצעות רקורסיה. ראשית אמליץ לכם, לפני קריאת הפתרון, לעצור ולנסות לממש את הפתרון בעצמכם. אדגים ואסביר מעט מפונקציות אלו, כדי להעביר לכם את צורת המחשבה.

מימוש strlen()

strlen() מקבלת מחרוזת ומחזירה את אורכה (מספר התווים בה, לא כולל התו '\0').

<u>בסיס הרקורסיה</u>: כידוע מחרוזות ב-C ממומשות כמערכים. המקרה הטריוויאלי הוא המקרה בו התא הראשון הוא '\0' ואז המחרוזת היא באורך 0:

```
int strlen_rec(char *s)
{
	if (*s == '\0') return 0;
}
```

צעד הרקורסיה: "אני מניח שאני יודע לספור אורך של מחרוזת באורך n-1, ולכן אוסיף לסכום שאר המחרוזת 1 וקיבלתי את התוצאה". זה המשפט שאני אומר לעצמי כשאני בא לפתור את זה. למה אני רוצה להניח שאני יודע לחשב אורך של מחרוזת באורך n-1? כי זה מקרב אותי למקרה הטריויאלי של מחרוזת באורך 0 שאותו לפי הבסיס אני יודע לחשב. לפי ידע זה בחרתי את ההנחה שלי.

בואו נראה איך אני הופך את ההנחה הזו ממילים לקוד:

```
int strlen rec(char *s)
{
      if (*s == '\0') return 0;
      return strlen_rec(s+1)+1;
}
```

נפרק את השורה (הקצרה) לרכיבים:

- strlen_rec(s+1) – על פי ההנחה שלי, אני יודע לחשב את האורך של מחרוזת באורך n-1 המחרוזת המתחילה בתו s+1 היא בדיוק כזו, ולכן לפי ההנחה אני יודע לחשב אותה.
- strlen_rec(s+1)+1 אני מוסיף 1 למחרוזת שאני יודע לחשב, ומחזיר את התוצאה. תוספת ה-1 היא עבור התו ה-n שנספר כעת.

בנפנוף ידיים (וירטואלי) אראה לכם שהרקורסיה הזו עובדת:

- אם ניקח מחרוזת באורך 0, הרי שלא נקרא כלל קריאה רקורסיבית והפונקציה תחזיר 0.
- אם ניקח מחרוזת באורך 1, החישוב יהיה "אורך של מחרוזת באורך 0" ועוד 1. כלומר 1. תוצאה נכונה.
- אם ניקח מחרוזת באורך 2, החישוב יהיה "אורך של מחרוזת באורך 1" ועוד 1, כלומר 2.
- בצורה כזו, עבור כל n ניתן לראות שהפונקציה מקיימת את הנדרש ומחזירה את הערך.

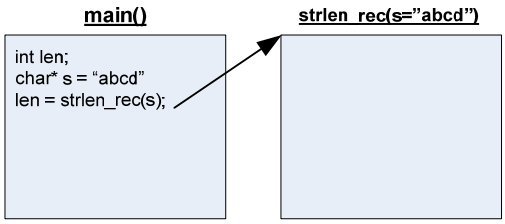
להמחשה נוספת, אשרטט לכם את העניין בצורה גרפית. ניקח לדוגמא את המחרוזת הבאה כקלט:

```
s = "abcd";
```

הקריאה שלנו תהיה

```
int main()
{
	int len;
	char* s = "abcd"
	len = strlen_rec(s);
	printf("%d\n", len);
	return 0;
}
```

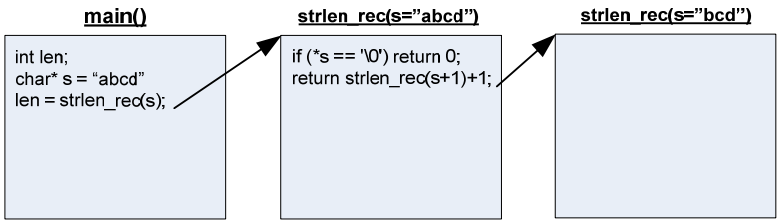
main() קורא לפונקציה strlen_rec, כאשר s במקרה הזה הינו "abcd":



הפונקציה בודקת את תנאי העצירה, התנאי לא מתקיים ולכן אנחנו הולכים לקריאה הרקורסיבית. בקריאה הראשונה אנחנו מבצעים קריאה רקורסיבית עם s+1. נשים לב כי s+1 הינו:

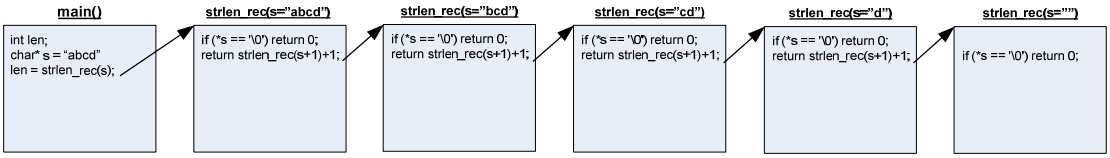
```
s+1 = "bcd"
```

זוהי המחרוזת המועברת לקריאה הרקורסיבית הראשונה:

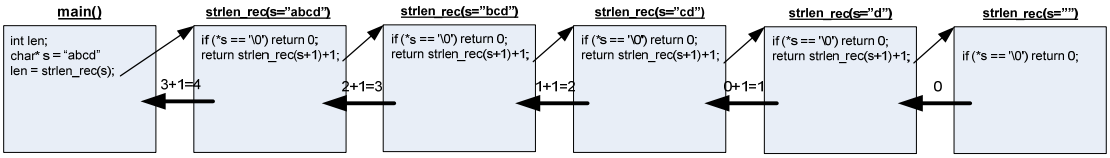


זהו הקטע החשוב הראשון בהבנה של הרקורסיה – רואים בשרטוט איך המחרוזת קטנה, בדרך אל תנאי העצירה שלנו, שהוא מחרוזת ריקה.

נקצר תהליכים, ונתקדם בשרטוט עד לרגע בו אנחנו מגיעים לתנאי העצירה.

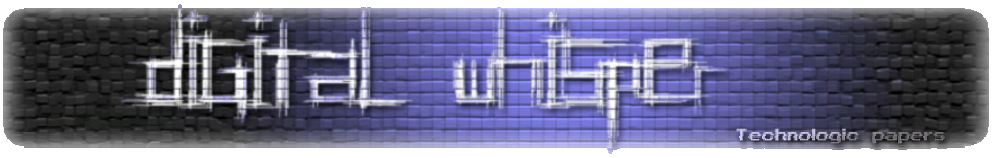


כעת נעקוב אחרי הערך המוחזר בכל שלב:



דברים שחשוב לשים לב אליהם בדוגמא זו:

1. למרות שכל הזמן אמרתי לכם את המשפט "אני מניח שאני יודע לחשב אורך של מחרוזת באורך n-1", שימו לב שאני לא יודע מה הוא ערכו של n! אני יודע לקבל מחרוזת באורך n-1 פשוט על ידי הסרת התו הראשון, אבל ערכו של n לא ידוע לי (אם הוא היה ידוע, לא הייתי צריך לחשב את אורך המחרוזת).
2. תוכלו לראות איך כל ושלב שלב ברקורסיה מחזיר תוצאה נכונה – כל שלב לוקח את האורך של המחרוזת באורך n-1 ומוסיף לו 1, וכך מחזיר את האורך הנכון של המחרוזת שהועברה לאותה הפונקציה.
3. מומלץ לעקוב ולראות איך האלגוריתם הפשוט בן 2 השורות מתנהג, וגם משיג את המטרה שלשמה כתבנו אותו. הבנה כזו תעזור לכם להבין איך הרקורסיה שלכם מתנהגת, ולדעת לנתח אותה כאשר משהו משתבש.



מימוש strcpy()

נממש פונקציה נוספת הקשורה למחרוזות – strcpy() המעתיקה מחרוזת מהמקור אל היעד. דוגמא זו מראה מעט את חשיבות בחירת בסיס הרקורסיה. הפונקציה תחזיר מצביע למחרוזת היעד.

בסיס הרקורסיה: מהו המקרה הבסיסי במקרה שלנו? קיימות 2 מחרוזות, ולכן חשוב לא להתבלבל ולהגדיר את התנאי הנכון. השאלה שאני שואל את עצמי "מתי אנחנו יודעים מיידית להעתיק את מחרוזת המקור, ללא צורך בשום פעולה נוספת?". התשובה היא "כאשר מחרוזת המקור ריקה. במקרה כזה אני צריך לשים '\0' במחרוזת היעד וסיימתי את הפעולה".

```
char* strcpy rec(char *dst, char *src)
{
	if (*src == '\0')
	{
		*dst = *src;
		return dst;
	}
}
```

נקודה שלעתים אפשר לטעות בה היא לשכוח לחזור מהפונקציה בתום העתקת ה-'\0'. אנו חייבים לזכור שברגע זה הסתיימה הרקורסיה ולכן אנחנו חייבים להשתמש ב-return.

צעד הרקורסיה: אבחר טענה דומה לזו שהשתמשתי בה ב-strlen(): אני מניח שאני יודע להעתיק מחרוזת באורך n-1. (למשל, המחרוזת שהיא כל התווים למעט התו הראשון). אני אשתמש בהנחה זו כדי להעתיק את כל שאר התווים, ואז אעתיק באופן ידני את התא הראשון, שנותר. איך זה נראה בקוד?

```
char* strcpy_rec(char *dst, char *src)
{
	if (*src == '\0')
	{
		*dst = *src;
		return dst;
	}

	/* העתק מחרוזת קטנה ב-1 אל התא המתחיל בכתובת dst+1 */
	strcpy_rec(dst+1, src+1);

	/* העתק את התא הראשון מהמקור אל היעד */
	*dst = *src;
	return dst;
}
```

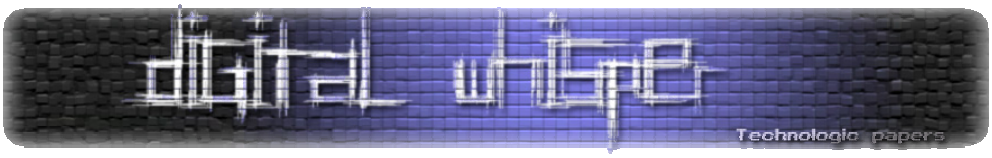
נשים לב שבכל שלב אנחנו זורקים את הערך שהקריאה הרקורסיבית הקודמת שלחה (אנחנו לא שומרים אותו בשום מקום). אין עם זה שום בעיה – כי אנחנו צריכים להחזיר מצביע למחרוזת היעד, שהוא נתון בקריאה הראשונה לפונקציה הרקורסיבית, והוא באמת זה שמוחזר בסוף.

סיכום ביניים

אני רוצה שנעצור רגע בכדי להסביר קצר יותר את כל מה שהצגתי עד כאן במאמר זה. אנחנו רואים רצף אלגוריתמים רקורסיבים אחד אחרי השני. בכל אחד מהאלגוריתמים שהדגמתי, הצגתי לכם אספקט חדש הקשור לאלגוריתמים הרקורסיביים. נעשה סיכום קצר של הנקודות שלמדנו:

1. השאלה הראשונה שנשאל את עצמנו כשאנחנו מנסים לפתור בעיה רקורסיבית, היא "מה הקלט הפשוט ביותר לפונקציה, הקלט שעבורו בכלל לא צריכים לעשות שום חישוב, ואפשר לדעת מה התוצאה?". התשובה לשאלה זו תהיה בסיס הרקורסיה שלנו.
2. לעתים יש לנו יותר מתנאי בסיס אחד הדרוש על מנת שהרקורסיה תעבוד כמו שצריך.
3. צורת העבודה שלנו לא משתנה אפילו במידה ואנחנו נרצה לעבוד עם מערכים. גם במקרה כזה אנחנו מדברים על תנאי העצירה ועל צעד הרקורסיה. כאשר אנו משלבים פתרון רקורסיבי עם מערכים, המקרה הטריויאלי לרוב יהיה כאשר הפונקציה מקבלת מערך בגודל תא אחד.
4. בשילוב רקורסיה ומחרוזות, המקרה הטריוויאלי במקרים רבים יהיה המקרה בו התא הראשון הוא '\0'.
5. במקרה שבו בתנאי העצירה מבצעים פעולות נוספות מלבד העצירה עצמה (למשל – העתקת תו), חשוב לא לשכוח לכתוב return על מנת לסיים את הקריאה הרקורסיבית.

כמו שציינתי, בחירה טובה של בסיס הרקורסיה היא קריטית להצלחה שלנו. כאשר תנסו לכתוב פונקציות רקורסיביות בעצמכם, סביר להניח שבפונקציות הראשונות לא תמיד תבחרו ישר את התנאי המתאים – תוכלו לראות זאת כאשר התנאי שבחרתם לא פשוט לבדיקה, או לחילופין כאשר הוא פשוט לבדיקה, אך לא נותן לכם כלים שמאפשרים לפתור את הבעיה. הכרת תבניות ותרגול תעזור לכם בנושא זה. בנוסף, חשוב להכיר טכניקות שונות הקשורות לאלמנטים השונים של השפה ולרקורסיה. ניתן ללמוד אותן על ידי תרגול. נראה כעת טכניקות נוספות כאלו.



החזרת התו האחרון במחרוזת

נחזור לאיפה שהפסקנו- אלגוריתמים רקורסיביים הפועלים על מחרוזות. הפונקציה שנרצה לכתוב כעת מקבלת מחרוזת ומחזירה את התו האחרון שבה. דרישה זו מגדירה את חתימת הפונקציה:

```
char last_char(char* str);
```

כרגיל, כמו עם כל מחרוזת שאנחנו מקבלים בשפת C, אנחנו מניחים שאיננו יודעים את אורך המחרוזת שאנחנו מקבלים, וכן שהמחרוזת מסתיימת ב-'\0'.

בסיס הרקורסיה: למרות שעד כה המקרה הבסיסי ביותר היה מקרה בו קיבלנו מחרוזת ריקה, במקרה זה המקרה הבסיסי ההגיוני ביותר הוא דווקא מקרה בו למחרוזת יש תו בודד. נניח, לצורך הדוגמא, כי נתון לנו שהמחרוזת לפחות באורך תו 1. אם התא השני במחרוזת הוא '\0' אנחנו יודעים שהמחרוזת בת תו אחד, ולכן אנחנו פשוט צריכים להחזיר תו זה.

```
char last_char(char* str)
{
	if (str[1] == '\0')
		return str[0];
}
```

צעד הרקורסיה: כרגיל, אני אבחר צעד שיקטין את המחרוזת, מכיוון שתנאי הרקורסיה הוא כזה היודע לטפל במחרוזת קטנה יותר. "אני אניח שהפונקציה יודעת להחזיר את התו האחרון של מחרוזת באורך n-1 תווים". למה (ואיך) אני בוחר הפעם את הנחה זו? א. כי זו הנחה נוחה – אני יכול להביא לה את שאר המחרוזת והיא תחזיר לי את המידע שאני צריך. ב. אני יודע שזו המטרה הסופית של הפונקציה. קל לי להניח שאני אדע לבצע אותה על קלט קטן יותר. אחרי שבחרתי את ההנחה, הצעד שלי יהיה פשוט – אני אפעיל את המחרוזת על שאר המערך, ואחזיר את מה שהיא תחזיר לי (אין צורך בחישוב נוסף מצידי כי הפונקציה כבר מחזירה את הערך הנדרש):

```
char last_char(char* str)
{
	if (str[1] == '\0')
		return str[0];
	return last_char(str+1);
}
```

אני משאיר לכם להשתכנע שהפונקציה הזו אכן עושה את הפעולה המבוקשת. למתקשים – נסו לצייר את תרשים הקריאות של הפונקציה ולהבין מה קורה.

מה למדנו מרקורסיה זו?

- למרות שלרוב במחרוזות אנחנו מדברים על תנאי עצירה כאשר המחרוזת הנתונה ריקה, יכולים להיות גם תנאי עצירה אחרים. חשוב לחשוב כל פעם מהו תנאי העצירה הנכון.
- ההנחה שהנחנו, שיש ברשותנו מחרוזת בת תו 1 לפחות, פישטה את הפונקציה. צריך תמיד לראות שבאמת מותר לנו להניח הנחות כאלה.
- בחירת ההנחה היתה קריטית בתרגיל זה. ההנחה עשתה למעשה את כל העבודה. כדי לחשוב על ההנחה בפעם הראשונה שנתקלתי בתרגיל הייתי צריך לשאול את עצמי שאלה כזו "מתי יש לי ביד את המידע לגבי זהות התו האחרון במחרוזת?". התשובה היא כמובן – בתנאי העצירה. מכאן עולה השאלה שלנו – "איך אני מעביר למעלה, לפונקציה הקוראת, את התשובה הסופית. התוצאה היא הדוגמא שראיתם.

סיכום

בזאת מגיע לסיומו המאמר על אלגוריתמים רקורסיבים. למרות אורכו היחסי וכמות התרגילים שהצגתי כאן, זו רק טעימה ראשונית של הנושא. אלגוריתמים רקורסיביים יכולים לפתור בעיות רבות: ניתן למיין מערכים בעזרת מיון רקורסיבי (מיון עם מימוש רקורסיבי מפורסם מאוד הוא merge sort), ניתן לפתור תרגילים מתמטימים ושאלות מתמטיות שונות. (למשל, מציאת מחלק משותף מקסימלי), ניתן גם לבצע חיפוש בינארי במערך והדוגמאות עוד רבות.

מתכנתים רבים שיצא לי לפגוש לא חושבים על רקורסיה בצורת "בסיס+צעד" אלא מסתכלים על כל עניין הרקורסיה בצורה מעורפלת משהו ומתקשים למצוא פתרונות רקורסיביים בסיסיים. חשוב היה לי להעביר לכם את הגישה שלי להסתכלות על בעיות רקורסיביות. ניסיתי להדגיש לכם במיוחד "מה עובר לי בראש" כשאני מסתכל על בעיה הדורשת פתרון רקורסיבי.

כשתפתרו לעצמכם תרגילים נוספים הקשורים לרקורסיה בהמשך – רישמו לעצמכם כל פעם מה למדתם מהתרגיל ומה אתם לוקחים איתכם הלאה. האם למשל למדתם איך לעשות רקורסיה שיש לה 2 תנאי עצירה שונים שגורמים לעצירה במצבים שונים? או אולי הרקורסיה שלכם מתפצלת ל-2 רקורסיות שונות לפי קיום/אי קיום של תנאי כלשהו? ישנה עוד טכניקה רבה הקשורה לרקורסיה שניתן לתרגל.

שיהיה לכולכם בהצלחה בעולם הרקורסיות!