שפת BrainFuck

cp77fk4r

מסמך זה הורד מהאתר http://www.underwar.co.il.

על סדר היום:

I - הקדמה.

II - מכונת טיורינג.

III - הסבר,בסיס+מספר דוגמאות.

IV - קלט ופלט.

V - לולאות.

VI - סיכום.

I - הקדמה:

במסמך זה נלמד את שפת התכנות BrainFuck, למה? סתם כי היא מגניבה.

מה מגניב בה? שימו לב לקוד הבא:

```
>+++++++++[<++++++++>-]<.>+++++++[<++++>-]<+.+++++++..+++.[-]>++++++++[<++++>-]
<.>++++++++++[<+++++>-]<.>++++++++[<+++>-]<.+++.------.--------.[-]>++++++++[
<++++>-]<+.[-]++++++++++.
```

כל הקוד הזה מדפיס את המילים "Hello World!". תודו שזה מגניב.

למה נועדה BrainFuck ?

BrainFuck נוצרה על-מנת לממש את "מכונת טיורינג". מכונת טיורינג היא מכונה דמיונית, בעלת זכרון וכוח אינסופי לחישוב ועיבוד נתונים, משהו כמו מחשב עם זכרון וכח עיבוד אינסופי ;). מכונת טיורינג היא נושא מאוד גדול בענף האלגוריתמיקה, ולכן ראיתי לנכון לכתוב על BrainFuck טקסט.

II - מכונת טיורינג:

בכדי להצליח להבין איך לתכנת ב-BF, אנחנו צריכים להבין איך פועלת מכונת טיורינג.
ההסבר הבא לקוח מויקיפדיה:

מכונת טיורינג, מקור: ***(http://he.wikipedia.org/wiki)***
מכונת טיורינג היא מודל מופשט לאופן פעולתו של מחשב, רעיון זה נוצר בשנת 1936 על-ידי אלן טיורינג, כדי ליצור הגדרה מתמטית מדויקת של אלגוריתם, או "תהליך מכני".
עוצמתו של הרעיון נעוצה בפשטות הקיצונית של המודל (בהשוואה למורכבותם של מחשבים אמיתיים).
התזה של צ'רץ' (או, בשמה האחר, התזה של צ'רץ'-טיורינג) קובעת שמכונת טיורינג, חרף פשטותה, מסוגלת לבצע כל חישוב או אלגוריתם שהוא בר-ביצוע במחשב כלשהו. מבחינה זו, מכונת טיורינג שקולה לכל מחשב, ולכן משמשת עד היום במדעי המחשב, בעיקר בתורת הסיבוכיות ובתורת החישוביות, כבסיס לחקר יכולותיו ומגבלותיו של המחשב תוך התעלמות מתכונותיו של מחשב מסוים זה או אחר.

טיורינג פיתח את מכונת טיורינג האוניברסלית כניסוי מחשבתי, בניסיון לזהות שאלות מתמטיות, שאינן ניתנות להכרעה ובכך להבדילן משאלות מתמטיות הניתנות להכרעה, שלגביהן משפט אי השלמות של גדל אינו תקף.

מכונת טיורינג מורכבת מהרכיבים הבאים:

1. סרט המחולק לתאים הנמצאים זה אחר זה. בכל תא יש סמל יחיד מתוך אלפבית סופי כלשהו. אלפבית זה כולל סמל
   "ריק" (המיוצג ע"י 0), ועוד סמל אחד לפחות. הסרט ניתן להארכה לימין ולשמאל ללא הגבלה, כלומר-
   למכונת טיורינג יש סרט בכל כמות שתזדקק לה. תאים שטרם נכתב בהם דבר נחשבים לכאלה שמכילים את הסמל "ריק".
2. ראש שמסוגל לקרוא ולכתוב את תוכנו של תא, ולזוז ימינה או שמאלה לאורך הסרט.
3. רשימת מצבים סופית, אחד מהמצבים מסומן כמצב ההתחלתי.
4. אוגר מצב, שבו נשמר מצבה הנוכחי של המכונה. בתחילת פעולת המכונה, מצבה הנוכחי הוא המצב ההתחלתי.

5. טבלת פעולה, שמורה לראש מה לכתוב בתא, לאן לזוז (תא אחד ימינה או תא אחד שמאלה), ולאיזה מצב חדש לעבור כל זאת בהתאם למצב הנוכחי (כפי שהוא רשום באוגר המצב) ולסמל שנקרא מהתא הנוכחי.
   אם אין בטבלה התייחסות לצירוף של המצב והסמל הנוכחיים, המכונה עוצרת.
   כל מרכיביה של מכונת טיורינג הם סופיים, מלבד הסרט שאינו מוגבל באורכו.

מכונת טיורינג ספציפית, בהתאם לתיאור המופיע לעיל, היא מעין מחשב שנועד לבצע תוכנית מסוימת.
טיורינג הראה שניתן לרשום על הסרט טבלת פעולה כלשהי, ולאחריה קלט כלשהו, כך שמכונת טיורינג אוניברסלית תקרא את טבלת הפעולה, לאחר מכן תקרא את הקלט, ואז תרשום על הסרט את תוצאות פעולתה של מכונת טיורינג על-פי טבלת הפעולה הנתונה על הקלט הנתון.

אחד מסוגי השאלות שניתן להציב בפני מכונת טיורינג הוא שאלות אודות אופן פעולתן של מכונות טיורינג אחרות.
רבות משאלות אלה אינן ניתנות להכרעה, כלומר, אי אפשר לענות עליהן.

דוגמה לשאלה כזאת שבה עסק טיורינג, היא השאלה האם מכונת טיורינג מסוימת תגיע לעצירה כאשר היא פועלת על קלט נתון או על קלט כלשהו. בעיה זו ידועה בשם בעיית העצירה, וטיורינג הראה שהיא אינה ניתנת להכרעה.

מכונת טיורינג היא מודל מופשט לחלוטין, והמסקנות הנובעות ממנו אינן דורשות מימוש פיזי של המודל, יחד עם זאת ברור שניתן לממש את מכונת טיורינג על כל מחשב מודרני (מלבד אינסופיות הסרט שאינה מתיישבת עם סופיות הזיכרון של מחשב ממשי) מימוש ישיר של מכונת טיורינג נעשה בשפת התכנות BF.

III - הסבר,בסיס+מספר דוגמאות.

כמו שכבר הבנתם, השפה מבוססת על מעין סרט אינסופי (הוא אינסופי, אך בגלל שאין לנו זכרון אינסופי, במחשב
הוא סופי (ב BrainFuck הוא באורך של 3000 bytes).
בכל תא ותא על הסרט קיים תו מסוים, אחרי רצף כל התווים- מתחיל רצף חדש.
אם לדוגמא, כל התווים שלנו הם: "1234567890", אז הסרט שלנו יראה כך:

NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	...

(כמובן שהתא "NULL" זה תא ריק, ולא המילה "NULL").
אוקיי, אם למשל אנחנו רוצים להציג את הסיפרה 8 למשל, אנחנו נהיה צריכים להתקדם ברשימה שלנו עד למקום שבו הסיפרה שמונה קיימת ואז להגיד למכונה להדפיס.

בכדי להתקדם קדימה ברשימה, אנחנו נשתמש בפקודה: "+" (פלוס).

(בכדי להדפיס את תוכן הבא, אנחנו נשתמש בפקודה: "." (נקודה).

זאת אומרת, שהקוד הבא:

```
++++++++.
```

ידפיס לנו את הסיפרה 8.

אם למשל אנחנו רוצים להדפיס את המספר 666, אנחנו נשתמש בקוד הבא:

```
++++++...
```

אנחנו מתקדמים עד לתא השישי, ואז מדפיסים שלוש פעמים.

בכדי לחזור אחורה ברשימה, אנחנו נשתמש בפקודה: "-" (מינוס).

לדוגמא, הקוד הבא:

```
+++++++++.---.---.
```

יתקדם קדימה עד למיקום של הסיפרה 9, ידפיס אותו, יחזור אחורה שלוש צעדים, ידפיס את הסיפרה שנמצאת שם, יחזור אחורה עוד שלוש צעדים, וידפיס גם את מה שנמצא שם, מה שיתן לנו את המספר: 963.

פשוט ביותר, לא? בואו נתקדם.

אני מקווה שעד לכאן הכל מובן, כי מכאן הדברים נהיים קצת יותר מסובכים.

אז כמו שכבר אמרתי, קיימת הרשימה ההיא, עכשיו, זאת רק חצי מהמציאות, באמת קיימת רשימת דו ממדית (רשימת רשימות), ובכל רשימה ברשימה קיימים כל מחרוזת התווים בצורה אינסופית. על מנת להבהיר את הנושא, נדגים בעזרת ציור.

NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	...
NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	...
NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	...
NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	NULL	1	2	3	4	5	6	7	8	9	0	...
...																																	
...																																	
...																																	

אוקיי, למה אני מספר לכם את זה? בגלל שהעובדה הזאת עוזרת בהרבה מצבים, אבל לפני הדוגמאות, אני אסביר את הממשק של השפה.

בכדי לעבור לרשימה הבאה, אנחנו נשתמש בפקודה: ">" (גדול מ-).

בכדי לחזור לרשימה הקודמת, אנחנו נשתמש בפקודה: "<" (קטן מ-).

חשוב לציין כמה דברים:

1. כל הרשימות דומות.
2. אם אנחנו עוברים לרשימה מסוימת- הסמן יקפוץ לתחילת הרשימה.
3. אם אנחנו עוברים לרשימה מסוימת שכבר היינו בה- הסמן יקפוץ למקום שבו הוא היה בזמן שעזבנו אותה.
4. במכונת טיורינג המקורית, אין סוף לרשימות, אבל ב BrainFuck יש.

שימוש פשוט שאפשר לבצע בעזרת מספר רשימות הוא הדפסת רווח.
לדוגמא, נניח שאנחנו למשל רוצים להציג את המספר 123 ואחריו את המספר 456 כשרווח מפריד ביניהם. על-יד רשימה אחת, הפעולה תיראה ככה:

```
+.+.+.----.++++.+.+.
```

לעומת זאת, ביצוע הפעולה הזאת בעזרת שימוש במספר רשימות יראה כך:

```
+.+.+.>.<.+.+.
```

איך זה קורה? בואו נעקוב ביחד:
בהתחלה, הסמן נמצא בערך NULL ברשימה מסוימת. לאחר מכן, אנחנו מתקדמים תא קדימה (1) ומדפיסים אותו, מתקדמים עוד תא קדימה (2) ומדפיסים אותו, ושוב, מתקדמים קדימה (3) ומדפיסים.
אחרי כל זה, אנחנו עוברים לרשימה הבאה, ומפני שזאת רשימה חדשה, הסמן קופץ לערך NULL אנחנו מדפיסים אותו, וחוזרים לרשימה הקודמת, הסמן קופץ לתא השלישי (כי שם עזבנו את הרשימה!) לאחר מכן, אנחנו מתקדמים תא קדימה (4) ומדפיסים, מתקדמים עוד קדימה (5) ומדפיסים, מתקדמים עוד קדימה (6) ומדפיסים.

הקטע הזה של מספר רשימות עוזר לנו לבצע פעולות כפל, איך? בשביל זה אנחנו צריכים ללמוד לולאות, נסביר את זה בהמשך.

IV - קלט ופלט:

לגרום לתוכנה לפלוט מידע כבר למדנו, אבל הייתי חייב לקרוא לפרק הזה "קלט ופלט" סתם כי הם תמיד באים ביחד ;)
בכדי לעצור את התוכנה ולגרום לה לבקש קלט מהמשתמש, אנחנו נשמש בפקודה: "," (פסיק).

כמה דברים לפני שממשיכים:
I) התוכנית לא תמשיך עד שהיא תקבל קלט מהמשתמש.
II) הקלט שיכנס יקפץ את הסמן שלנו למיקום הערך באותה הרשימה.
בואו ניקח דוגמא קטנה, אנחנו רוצים למשל לבצע תוכנית קטנה שתקבל קלט מהמשתמש ותדפיס את המיקום של התו אחריו ברשימה ואת התו שנמצא לפניו ברשימה.
הקוד שלנו יראה ככה:

```
,+.--.
```

כן, פשוט ביותר, אנחנו מקבלים קלט מסוים, מוסיפים לו אחד בכדי להגיע לתו שאחריו ברשימה- מדפיסים, מחסירים שתי תווים בכדי להגיע לתו שלפניו- ומדפיסים.
פשוט ביותר.

V - לולאות:

לולאות הן כלי מאוד שימושי וחזק ב BrainFuck, הן הכלי העיקרי לחישוב ערכים.
ב-BrainFuck יש רק סוג אחד של לולאה- לולאת While (כל עוד../בזמן ש..).

מה שמיוחד פה, זה שיש רק תנאי אחד ללולאה הזאת, התנאי להמשך הלולאה הוא שהסמן לא מורה על תא ברשימה שערכו הוא "NULL".

בכדי לפתוח לולאה, אנחנו נשתמש בפקודה: "]" (פתיחת סוגרים מרובעות).
בכדי לפתוח לולאה, אנחנו נשתמש בפקודה: "[" (סגירת סוגרים מרובעות).

ניקח לדוגמא את הקוד הבא:

[.+]

אנחנו מתחילים לולאה, מקדמים באחד את הסמן (1) מדפיסים את מה שכתוב, בודקים- האם הסמן לא מורה על תא ריק?
התשובה היא True, אנחנו ממשיכים- מקדמים את הסמן באחד (2) ומדפיסים, שוב מבצעים את הבדיקה, התשובה היא שוב True, אנחנו ממשיכים, מעלים את הסמן באחד (3) ומדפיסים.
כשהסמן יצביע על התא 9 ואנחנו נקדם אותו באחד- הוא יצביע על התא שמאכסן בתוכו NULL והלולאה תסתיים.
וכן הלאה וכן הלאה... אז מה הקוד עושה? הקוד מדפיס את כל סידרת התווים הראשונה ברשימה, במקרה שלנו הפלט יהיה:

1234567890

דוגמא נוספת, יותר מורכבת. הביטו על הקוד הבא:

```
++++[>++++<-]>.
```

נעבור שלב שלב וניראה מה הקוד עושה.
דבר ראשון, אנחנו רואים שהקוד משתמש בשתי סרטים, שימו לב שבהתחלה הוא מבצע משהו עם הסרט הראשון, ובתוך הלולאה הוא עולה סרט, משתמש בו- ויורד לסרט הקודם, בסוף הלולאה הוא שוב עולה סרט, מה שאומר שיש לנו כאן שימוש בשתי סרטים.
את הסרט הראשון אנחנו נבטא בעזרת המערך A.
את הסרט השני אנחנו נבטא בעזרת המערך B.

אוקיי, אז.. מתחילים.
אנחנו מתחילים -([A[Null],B[Null]).
מעלים את A ארבעה פעמים - ([A[4],B[Null]).
מתחילים לולאה.
עוברים לשימוש במערך B.
מעלים את B ארבעה פעמים - ([A[4],B[4]).
חוזרים לשימוש במערך A.
מחסרים את A ב1 - ([3],A B[4]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 3) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מעלים את B ארבעה פעמים - ([A[3],B[8]).
חוזרים לשימוש במערך A.
מחסרים את A ב1 - ([8],A B[2]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 3) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מעלים את B ארבעה פעמים - ([A[2],B[12]).
חוזרים לשימוש במערך A.
מחסרים את A ב1 - ([12],A B[1]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 3) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מעלים את B ארבעה פעמים - ([A[1],B[16]).
חוזרים לשימוש במערך A.

מחסרים את A ב1 - ([12],A B[1]).
הסמן מצביע על- NULL, ולכן אנחנו יוצאים מהלולאה.
עוברים לשימוש במערך B.
מדפיסים את ערך שקיים בB.
מה קיבלנו? יפה, הפלט הוא 16.

מה הלך פה? למה הבאתי את הקוד הזה? שימו לב למספר הפלוסים שקיימים בקוד, לפני הלולאה קיימים 4 פלוסים
ובתוך הלולאה קיימים עוד 4 פלוסים, מה התוכנה עושה? מחשבת את המכפלה של מספר הפלוסים מחוץ ללולאה
במספר הפלוסים שבתוך הלולאה!
נחמד, לא?

אני אביא עוד קוד לדוגמא בכדי שיהיה יותר מובן, בואו נעקוב אחרי הקוד הבא:

```
,>,<[>-<+]>.
```

(אוקיי, אנחנו יכולים לראות ממבט ראשון שהתוכנה משתמשת בסרט ראשון (1), עולה סרט (2), יורדת סרט(1),
עולה סרט (2) ויורדת סרט (1) ואחרי הלולאה היא שוב יורדת סרט (2),
זאת אומרת שגם כאן יש לנו שימוש רק בשני סרטים.
אנחנו גם יכולים לראות שיש לנו שימוש בשני קלטים (שני פסיקים , והם מחוץ ללולאה- שניים). את הסרט הראשון אנחנו נבטא בעזרת המערך A.
את הסרט השני אנחנו נבטא בעזרת המערך B.
מתחילים, הקלט שלנו יהיה 3 ו4:

אנחנו מתחילים - ([A[Null],B[Null]).
קולטים 3 לתוך - A ([A[3],B[Null]).
עוברים לשימוש במערך B.
קולטים 4 לתוך - A ([A[3],B[4]).
חוזרים לשימוש במערך A.
מתחילים בלולאה. עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[3],B[3]).
חוזרים לשימוש במערך A.

מעלים את A ב1 - ([A[4],B[3]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 4) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[4],B[2]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[5],B[2]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 5) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[5],B[1]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[6],B[1]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 6) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[6],B[Null]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[7],B[Null]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 7) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[7],B[0]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[8],B[0]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 8) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[8],B[9]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[9],B[9]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 9) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.
מחסרים את B ב1 -([A[9],B[8]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[0],B[9]).
הסמן אינו מצביע על- NULL, (הוא מצביע על- 0) ולכן אנחנו ממשיכים בלולאה.
עוברים לשימוש במערך B.

מחסרים את B ב1 -([A[0],B[7]).
חוזרים לשימוש במערך A.
מעלים את A ב1 - ([A[Null],B[7]).
הסמן מצביע על- NULL, ולכן אנחנו יוצאים מהלולאה.
עוברים לשימוש במערך B.
מדפיסים את ערך התא, הפלט שלנו יהיה 7.

מה התוכנה עושה? מחשבת את סכום הקלטים!

VI - סיכום:

לפני שאני מסיים עם הטקסט, הייתי רוצה להגיד משהו חשוב:
בטקסט הזה,בשביל הנוחות והסדר השתמשתי במציאות ששפת הקלט היא רק מספרית (בין 0 ל9), אבל קיימים כל התווים שהמחשב מסוגל להוציא, היא מתחילה BrainFuck במציאות שלנו- בשפת במספר סוגים של רווחים, ואז מתחילה עם התווים, ככה שלפני התוכנית תאלצו להוסיף מספר פלוסים, בכדי לדעת את הסדר (אסקי..) של התווים, אתם יכולים להכין תוכנה פשוטה (זאת : [+.]) שתדפיס את כל התווים לפי הסדר.

אז זהו, עוד סוף טקסט שלי, אני מקווה שעכשיו אתם מבינים למה קוראים לשפה הזאת **BrainFuck!** ;)