**תרגיל בית 1**

**משימה 1**

1. המטרה של Rust היא להיות שפת תוכנה לתכנות מערכות, מעשית, בטוחה ולהפחית בעיות של ניהול זיכרון וכתיבת קוד גדול (כמו למשל מרוץ תהליכים).

Rust קיימת כי שפות אחרות באותה רמה של הפשטה ויעילות הן לא מספקות, לטענת המפתחים.

העקרונות שהציבו לעצמם מפתחי השפה הם ניהול זיכרון בטוח וללא התקורה הגבוהה שלGarbage Collector כנהוג בשפות אחרות המנהלות זיכרון, תמיכה בכמה פרדיגמות תוכנה שונות, תכנות מקבילי בטוח וזיהוי של בעיות (בביצועים, מרוץ תהליכים וכדומה) עוד בשלב הקומפילציה כדי להפחית תקלות של זמן ריצה.

[מקור](https://www.rust-lang.org/en-US/faq.html)

1. מפתחי Rust משחררים גרסאות חדשות לשפה על בסיס 6-שבועי. פעם בשנתיים עד שלוש שנים יוצאת מהדורה חדשה.

אפשר להסיק מכך שהשפה עדיין צעירה, והמפתחים מנסים לענות על הדרישות של קהילת המתכנתים. על כן הם מעדכנים לעיתים די תכופות את השפה. אפשר לראות למשל שכדי לממש את המטרה הראשונית של השפה (שהובעה בסעיף 1) המפתחים מנסים כמה רעיונות חדשום ושונים ולא כולם נשארו. למשל, lifetimes ו-traits נשארו, אבל type state systems ו-green threading לא. כמו כן, חלק מהספרייה הסטנדרטית נכתב מחדש כך שתשתמש בצורה הטובה ביותר בשפה וכך שתתמוך בכל פלטפורמה.

[מקור 1](https://rust-lang-nursery.github.io/edition-guide/editions/index.html), [מקור 2](https://github.com/rust-lang/rust/blob/master/RELEASES.md), [מקור 3](https://www.rust-lang.org/en-US/faq.html)

1. הישויות שהן nameable ב-Rust הן:

functions, variables, parameters, struct fields, modules, crates, constants, macros, static values, attributes, types, traits, or lifetimes

[מקור](https://doc.rust-lang.org/book/2018-edition/appendix-01-keywords.html)

1. ב-Rust קיים מנגנון [name mangling](https://en.wikipedia.org/wiki/Name_mangling#Rust) המאפשר שימוש חוזר בשמות טובים בכך שבזמן ההידור של קובץ, הוא מערבל (hash) את השמות וכך בזמן הקישור, גם אם יש שימוש חוזר בשמות, הם יזוהו באופן שונה.
2. הישויות המוגדרות רקורסיבית ב-Rust הן: טיפוסים, מאקרו-ים, ביטויים ופונקציות.
3. [אוסף המילים השמורות ב-Rust](https://doc.rust-lang.org/book/2018-edition/appendix-01-keywords.html).

מזהים שמורים:

as, crate, enum, false, fn, match, mod, move, mut, Self, self, super, trait, true, use, where(?)

סימני פיסוק:

break, const, continue, else, extern, for, if, in, let, loop, pub, ref, return, static, struct, type, unsafe, while

1. ב-Rust גבולות התוכנית בדרך כלל הם מעבר לקובץ אחד. התוכנית מתחילה את ריצתה מהפונקציה main, אבל שם הפונקציה הוא לא חלק מהשפה אלא נקבע על ידי המהדר (לוודא). יש ב-Rust ספרייה סטנדרטית בשם std בה ממומשים פונקציות העזר בקבצים נפרדים והן לא חלק מהשפה. לכן Rust אינה שפה אוטרקית.

**משימה 2**

1. להלן תוכנית ב-AWK המדפיסה עצמה, מבוססת על [דוגמה מויקיפדיה ל-Quine ב-C](https://en.wikipedia.org/wiki/Quine_%28computing%29#Examples):

|  |
| --- |
| 1. BEGIN{prog="BEGIN{s=%c%s%c;printf(s,34,s,34);}";printf(s,34,s,34);} |

1. בדומה לתוכנית לעיל, בתוספת לולאת for נוכל להדפיס את התוכנית שלוש פעמים (זו שורה אחת ארוכה):

|  |  |
| --- | --- |
| BEGIN{prog="BEGIN{s=%c%s%c;for(i=1;i<=3;++i)printf(s,34,s,34);}" ;for(i=1;i<=3;++i)printf(s,34,s,34);} | 1. |

1. AWK היא שפה אוטרקית. כנהוג בגישה האוטרקית, AWK מגדירה מילה שמורה המציינת את תחילת התוכנית (BEGIN), כל פונקציות העזר הן חלק מהשפה עצמה - כלומר אין ספריות חיצוניות, וכל התוכנית נמצאת בקובץ יחיד.
2. כפי שציינו בסעיף הקודם, ב-AWK כל התוכנית כולה כתובה בקובץ יחיד והוא כל התוכנית. לכן, אם נרצה למשל ליצור ספרייה שימושית שתשמור שגרות ומבני נתונים (בדומה ל-STL ב-++C למשל), בכל תכנית שנרצה להשתמש בה נצטרך לבצע "העתק-הדבק" של כל אותו תוכן הספרייה שכתבנו לתוך קובץ התכנית.

בואריאנט של AWK בו קיימת הוראת INCLUDE ניתן להימנע מבעיה זו על ידי יצירת קובץ (או קבצים) שיכיל את הספרייה שלנו, ובכל פעם שנרצה להשתמש בה רק נוסיף את הוראת ה-INCLUDE המתאימה. חוץ מכך שזה פותר את הסרבול שבהעתקה לתוך כל תוכנית המשתמשת בספרייה, זה גם מקטין את גודל התכניות המשתמשות בספרייה, וחשוב מכך - פוטר אותנו מהצורך לתחזק את הקבצים על כל שינוי בספרייה. בהיעדר הוראת INCLUDE, אם נשפר, נשנה או נתקן מימושים בספרייה, נצטרך לדאוג לעדכן את כל התכניות שמשתמשות בספרייה זו.

**משימה 3**

[תיעוד של הדקדוק של ביטויים רגולריים](https://www.gnu.org/software/gawk/manual/html_node/Regexp.html).

ביטויים רגולריים מוגדרים רקורסיבית מכיוון ש:

1. קיים אלפבית סופי שסימניו מהווים את הביטויים הרגולריים היסודיים.
2. כל ביטוי רגולרי אחר מורכב מביטויים יסודיים או מביטויים מורכבים אחרים באמצעות:
   1. הוספת סימן שלא נמצא באלפבית - למשל הסימנים ?,+, \*, | וכדומה לאחר ביטוי רגולרי (אלה סימנים מעל אלפבית סופי ושונה מהאלפבית של הביטויים היסודיים).
   2. שרשור שני ביטויים רגולריים.
   3. קיבוץ ביטוי רגולרי בסוגריים ( ).
   4. כאשר במקום ניתן לשים אוסף של ביטויים יסודיים.
   5. שרשור או או לאחר ביטוי רגולרי (יסודי או מורכב) כאשר מספרים טבעיים.

כלומר, 1 מתאר את האיברים היסודיים בקבוצה ואילו 2 מתאר מנגנונים לבניית איברים מורכבים מהערכים היסודיים.