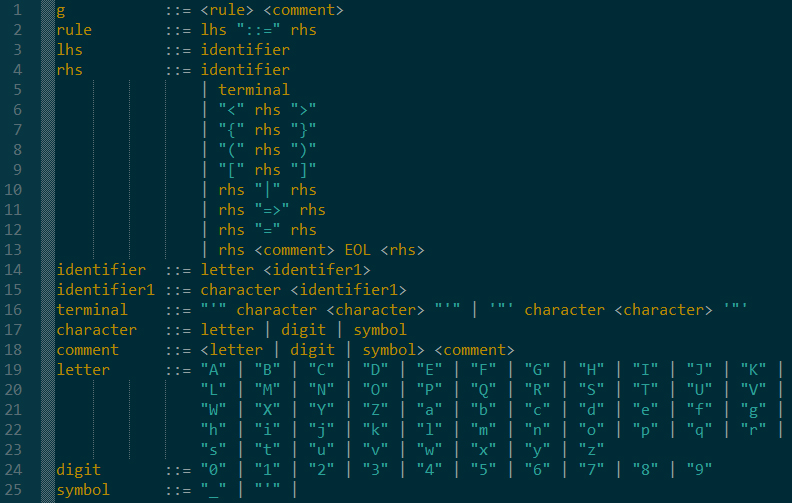
**תרגיל בית 2**



אחד הקשיים שנתקלנו הוא איך לאפשר חוק עם יותר משורה אחת, זה בעצם מה שמאפשרת שורה מספר 13. כדי לפתור את הבעיה אפשרנו הכנסת "אגף ימין" שאחריו תו שורה חדשה ואז באופן רקורסיבי לאפשר עוד "אגף ימני".

ב-ML, ממה שהבנו, לא נהוג לכתוב את התווים באופן מפורש כמו שאנחנו כתבנו למשל בהגדרת letter, אלא לציין אותם כקבוצה. למשל, ML מציינת את הקבועים המיוחדים בתור הקבוצה SCons, ואם צריך לציין אחד מהם, אז כותבים באותיות קטנים scon. בכל זאת, כתבנו זאת כך כדי שיהיה ברור יותר.

ייתכן מאוד שלא כיסינו את כל המקריים האפשריים של הגדרת חוקים בכתיב של ML, אבל לכל הפחות זה תופס את הרוח הכללית. 😊

* 1. שיטת הערות בשפה היא תחימה של רצף תווים בין "(\*” ל-”\*)”. כלומר, משהו כזה:

|  |
| --- |
| (\* some comment \*) |

* 1. נמיין את המילים השמורות (מתוך הסטנדרט):

סימני פיסוק:

abstype(?), and, as(?), case, datatype, do(?), else, end, exception, fun, handle, if, in, let, local, raise, rec, then, type, val, while, \_(?), |, =, ->

מזהים:

andalso, fn, infix, infixr, nonfix, not, op, open, orelse, =>, ->, |, =, with, withtype,=, …, #

* 1. ב-ML יש כמה סוגים של ליטרלים:

תו בודד – מוגדר באמצעות הסימן # ואחרי גרשיים סביב התו. למשל, #”a” עבור התו a.

מחרוזת – מוגדר באמצעות רצף תווים תחומים בגרשיים. למשל "abcd”.

מספר – מוגדר באמצעות המספר עצמו.

כדי לפתור בעיות של גידור של תווים מיוחדים השפה מאפשר מילוט באמצעות התו /. למשל עבור גרשיים יש לכתוב /”.

* 1. בנאי הטיפוסים התיאורטיים:

Power Set – אין בשפה.

Product – הבנאי של tuple.

Integral Exponent – הבנאי של list.

Branding – באמצעות datatype.

Records – הבנאי של records.

Disjoint Union – אין בשפה.

Mapping – הבנאי של פונקציות, עם fn.

Subrange – באמצעות cons.

Simple Recursive – באמצעות Datatype. דוגמה משקפי ההרצאות של 2017:

**Datatype** intlist = nil | :: of int \* intlist;

Multiple Recursive – באמצעות Datatype.

1. nameable הוא דבר מה הניתן לשיום. nameables הינם ישויות, כגון פונקציות מודולים, טיפוסים, קבועים ומשתנים להם המתכנת יכול לספק שם.
2. במאמר אין הסבר מפורש למה מתכוון Knuth כשהוא מציין צורה קנונית וצורה נורמלית. ממקורות שונים באינטרנט ומקריאת המאמר ([ויקיפדיה](https://en.wikipedia.org/wiki/Canonical_form), למשל), נראה שהוא התכוון שהצורה הנורמלית משקפת הצגה מיוחדת שהיא לאו דווקא קנונית. כלומר, אילו BNF הייתה צורה נורמלית אז היה ניתן לכתוב דקדוק של שפה בצורה יחידה. באופן דומה, צורה קנונית היא הצגה סטנדרטית כלשהי.

באופן עקרוני, אין הבדל בין Backus Normal Form ל-Backus Naur Form פרט לשם. Knuth הציע את החלפת השם כדי להוקיר תודה לתרומתו של Peter Naur ל-BNF, וגם כי השם לא משקף שכן BNF הוא לא בעל צורה נורמלית או קנונית – אין רק דרך אחת לכתוב דקדוק. בכל זאת, Knuth נותן במאמר מספר הבדלים שניתן לייחס לכתיב של Naur. Naur הוסיף ל-BNF את הסימן “::=” כדי להבדיל בין ימין לשמאל, את הסימן "|" המפריד בין חלופות שונות, ציון נונטרמינלים באמצעות שמות מלאים. אלה מה שמכנה במאמר Knuth כ-(iii), (iv) ו-(v).

1. מתוך ויקיפדיה: "מכונה וירטואלית היא תוכנה היוצרת סביבה הנחוצה להפעלתה של תוכנה אחרת, מבלי שיהיה צורך במימוש פיזי של סביבה זו".

מבין השפות שצוינו, השפות המיועדות למימוש על מכונה וירטואלית הן MetaPort, Java, JavaScript.

Dart היא שפה שנועדה לפיתוח Web, להחליף את JavaScript והיא מטפלת בכמה בעיות מרכזיות של שפת JavaScript. מהנדסי גוגל מתארים את Dart כ"שפה שקל לכתוב לה כלי פיתוח, מתאימה לאפליקציות מודרניות ומאפשרת מימושים שמתמקדים בביצועים גבוהים" ([מקור](https://webdev.dartlang.org/faq#q-why-dart)). היות ו-Dart נועדה לרוץ על מכונה וריטואלית, נבחר Lars Bak להוביל את פיתוח השפה. גוגל כתבו על לארס שהוא "מומחה במכונות וירטואליות, והותיר חותם על מספר מערכות תוכנה, מבניהן: Beta, Self, Strongtalk, Sun’s Hotspot and CLDC HI, OOVM Smalltalk, V8” ([מקור](https://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html)).

* 1. ה-frontend מתרגם את קוד המקור של התכנות לייצוג ביניים, וה-backend משתמש בייצוג הביניים על מנת ליצור את תוצר השפה, ומבצע אופטימיזציות לקוד בהתאם לארכיטקטורת המעבד ([מקור](https://en.wikipedia.org/wiki/Compiler#Three-stage_compiler_structure)).

דוגמה לשלושה backend-ים של Kotlin הם JavaScript, Android ו-Native ([קישור 1](https://kotlinlang.org/docs/reference/android-overview.html), [קישור 2](https://kotlinlang.org/docs/reference/js-overview.html), [קישור 3](https://kotlinlang.org/docs/reference/native-overview.html)).

* 1. Kotlin משתמשת בארבעה מנגנוני הגדרות:
     1. מדריכים ודוגמאות לשימוש בשפה וכתיבה ([קישור](https://kotlinlang.org/docs/reference/)).
     2. הגדרה מילולית של הסמנטיקה ו-EBNF ([קישור](https://kotlinlang.org/docs/reference/grammar.html)).
     3. מימוש (קוד המקור) של המהדר ([קישור](https://github.com/JetBrains/kotlin/tree/master/compiler)).
     4. מספקת סביבת התנסות עם דוגמאות, דרכה ניתן ללמוד את השפה ([קישור](https://try.kotlinlang.org/)).
  2. Kotlin היא לא אוטרקית. ניתן לייבא קוד מספריות שאין חלק מהקוד, נקודת ההתחלה של הריצה היא פונקציית main והתוכנית מסתיימת עם סיום פונקציית main.
  3. הישויות namable ב-Kotlin:

functions, variables, parameters, classes, packages, class fields, class members, interface, static values, inline classes

* 1. לא ניתן לתת לשתי ישויות את אותו השם באותו הסקופ, אבל ניתן לעשות shadowing לישות בסקופ פנימי. מתוך ויקיפדיה:

“For example Kotlin allow an inner variable in a function to shadow a passed

argument and a variable in inner block to shadow another in outer block”

* 1. בנאי הטיפוסים ב-Kotlin:

Power Set – הבנאי של Set הכי קרוב, למרות שהוא לא בדיוק תואם את ההגדרה של Power Set מתוך שקפי ההרצאות.

Product – אין בשפה.

Integral Exponent – הבנאי של List.

Branding – אין בשפה.

Records – הבנאי של class.

Disjoint Union – אין בשפה.

Mapping – בנאי הפונקציות ובנאי Array.

Subrange – הבנאי של Sequence.

Simple + Multiple Recursive – אין בשפה.