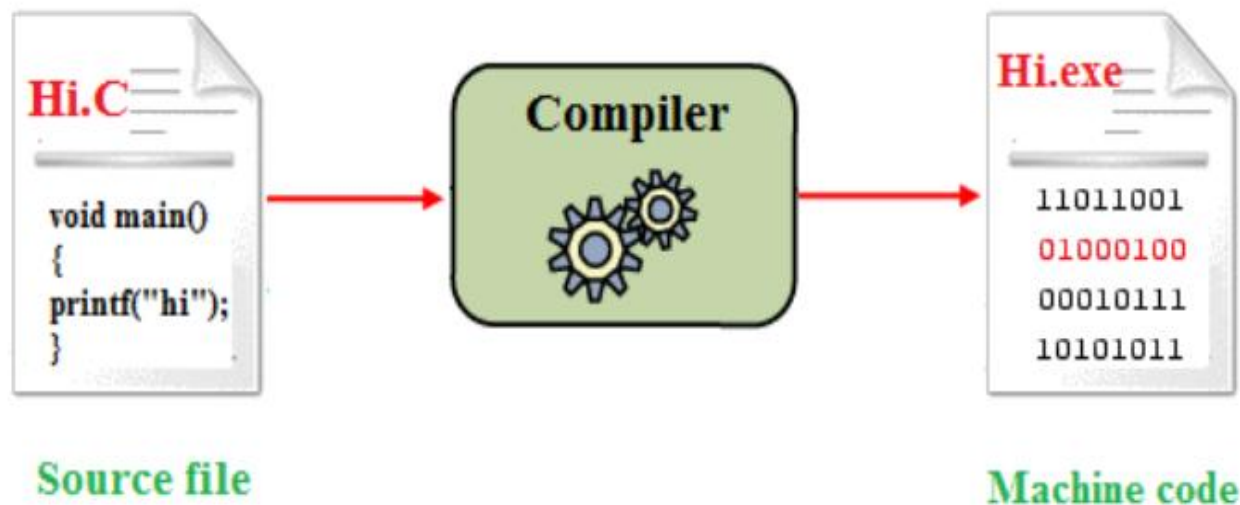


Compiler design

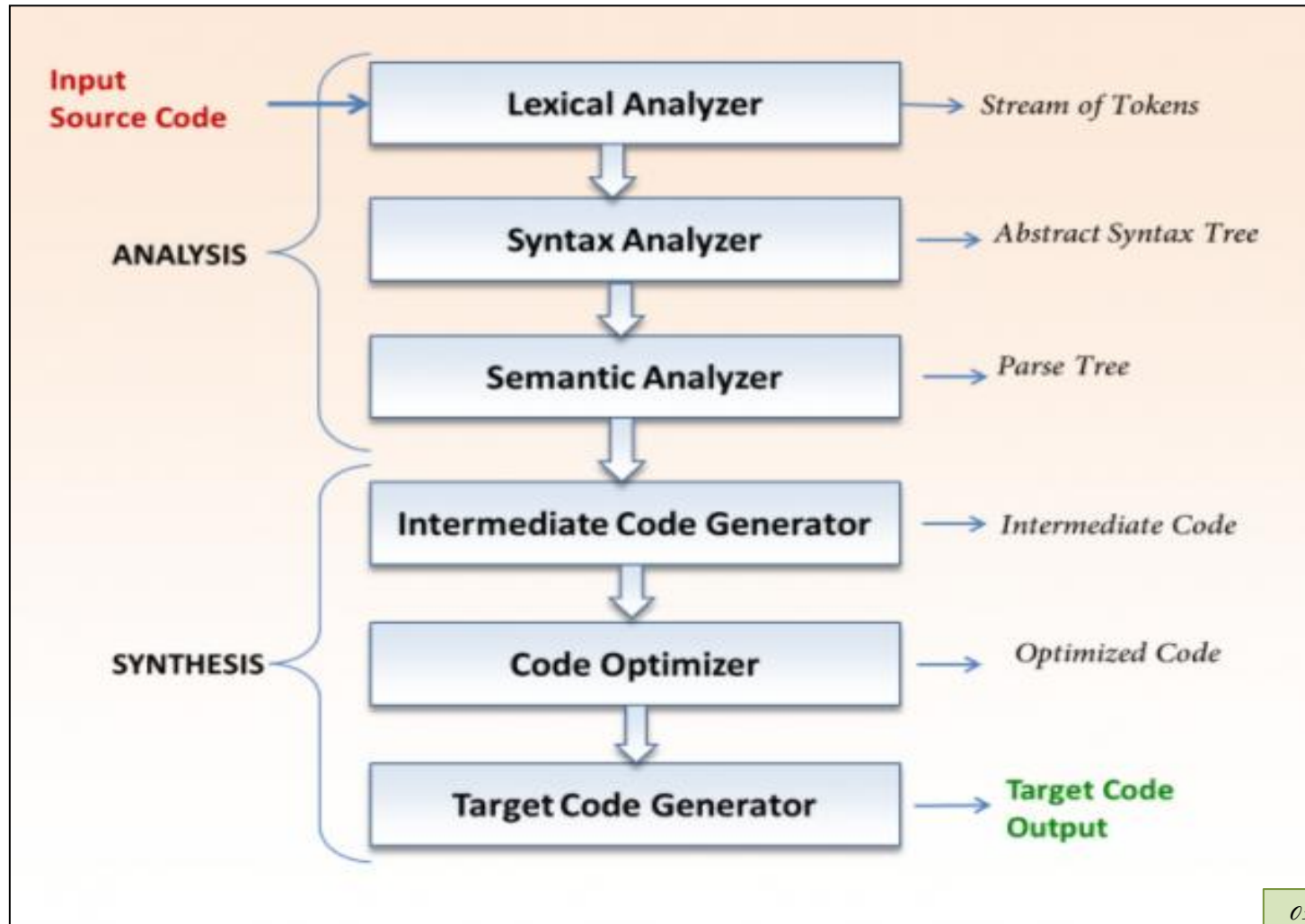
Compiler

- **קומפיילר (מהדר)** היא מערכת תכונות הקוראת תכנית הכתובה בשפה אחת, **שפת המקור**, ומתרגם אותו לתוכנית זהה בשפה אחרת – **שפת היעד**. במידה ויש שגיאות בתוכנית בשפת המקור הקומפיילר מודיע על כך למשתמש



Compiler

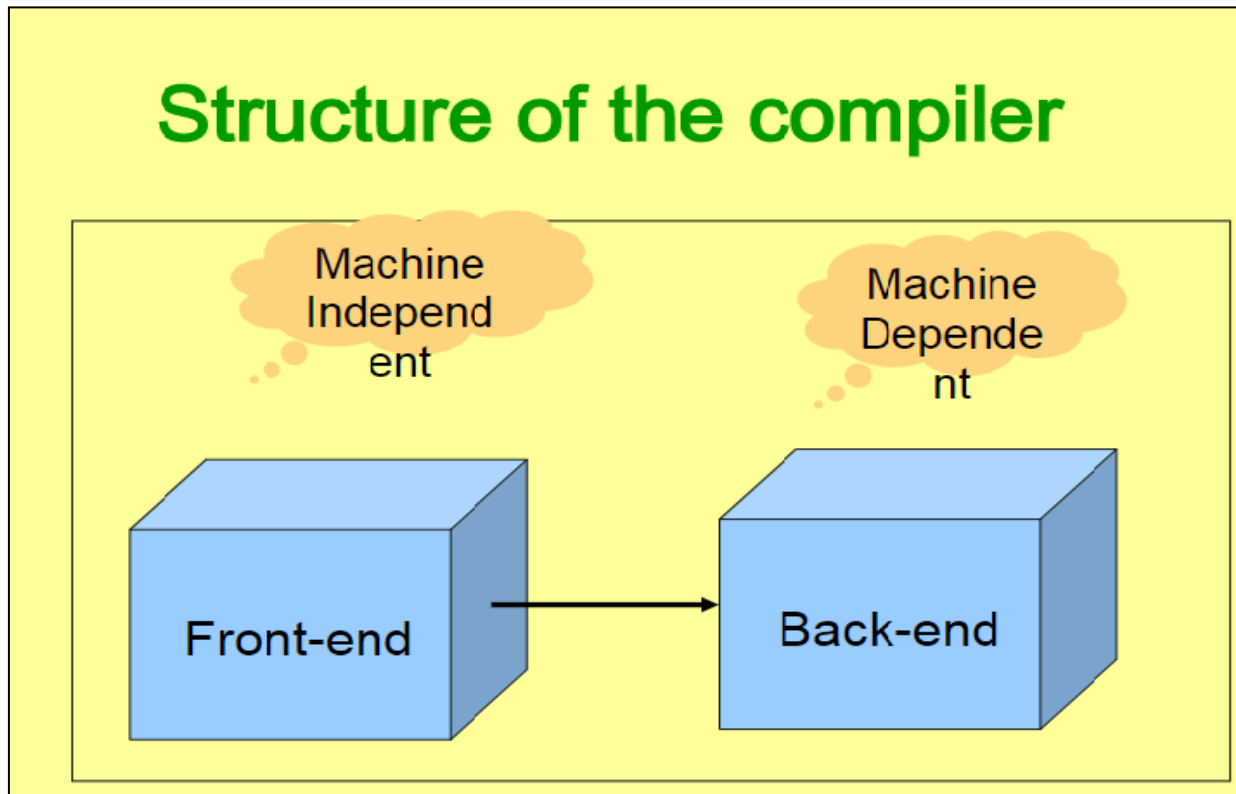
compiler phases / שלבי ההידור



Compiler

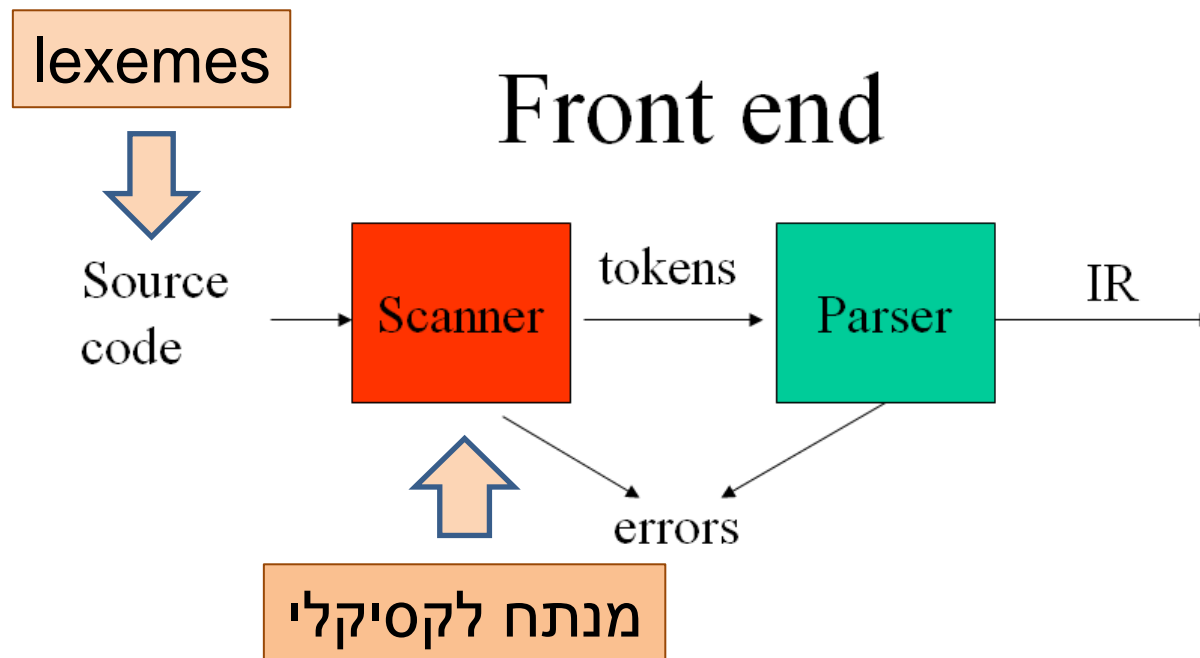
נחלק את שלבי ההידור ל 2- חלקים: **front end-** ו **back end**

ה **front - end** כולל שלבים **התלויים בשפת המקור** ולא תלויים בשפת היעד.
ה **back - end** מייצר קוד של שפת המטרה.



Lexical Analyzer

- **מנתח לקסיקלי (Scanner)** הינו השלב הראשון בקומפילציה, כל טקסט ובפרט שפת תוכנה הינה רצף תווים, תפקידו של המנתח הלקסיקלי לקלוט רצף תווים זה ולחלקו לתתי רצפים הנקראים **לקסמות (Lexeme)**, ולתייג כל לקסמה שכזו ליחידת דקדוק בסיסית הנקראת **אסימון (Token)**.



Lexical Analyzer

- **אסימון (Token)** – יחידה בסיסית המשמשת כטרמינל בדקדוק שגוזר את שפת התכנות.
- **לקסמה (Lexeme)** – מחרוזת בקלט (קוד המקור) שהמנתח הלקסיקלי התאים לאסימון כלשהו.

דוגמה:

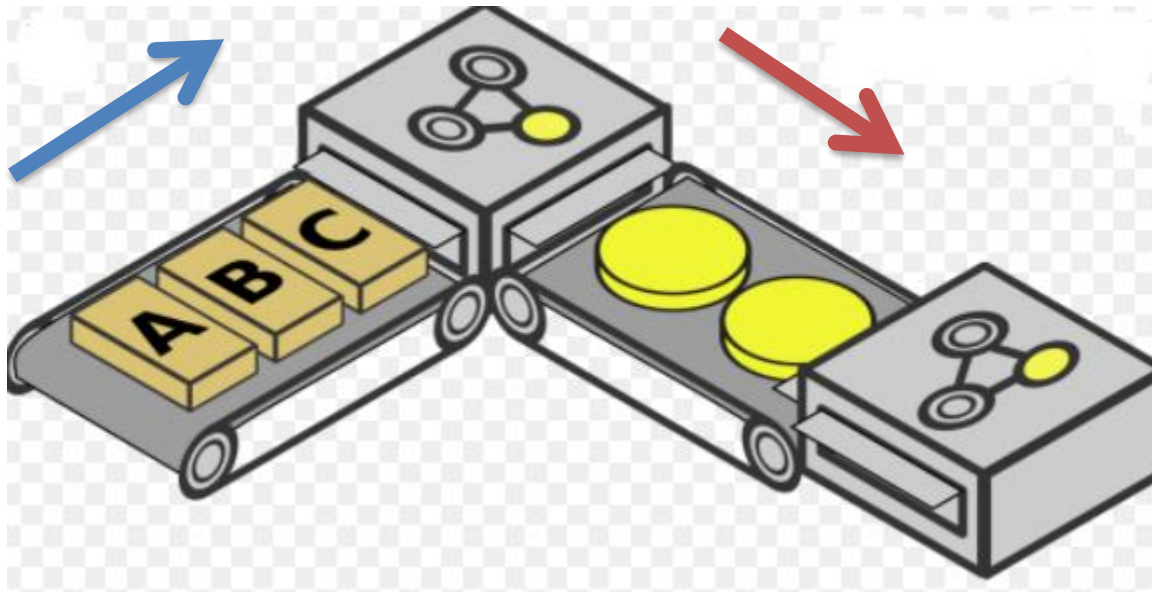
ניתוח לקסיקלי עבור "x = size + 29;" :

לקסמות	x	=	size	+	29	;	\n
אסימונים	id	assign	id	op	num	sep	

המנתח לא
יחזיר אסימון

LEX

Scanner מנתח לקסיקלי LEX



מנתח תחבירי YACC

מומלץ להוריד חוברות :

<https://silcnitc.github.io/lex.html>

<https://klasses.cs.uchicago.edu/archive/2003/spring/22600-1/docs/lexyacc.pdf>

<https://silcnitc.github.io/yacc.html>

LEX

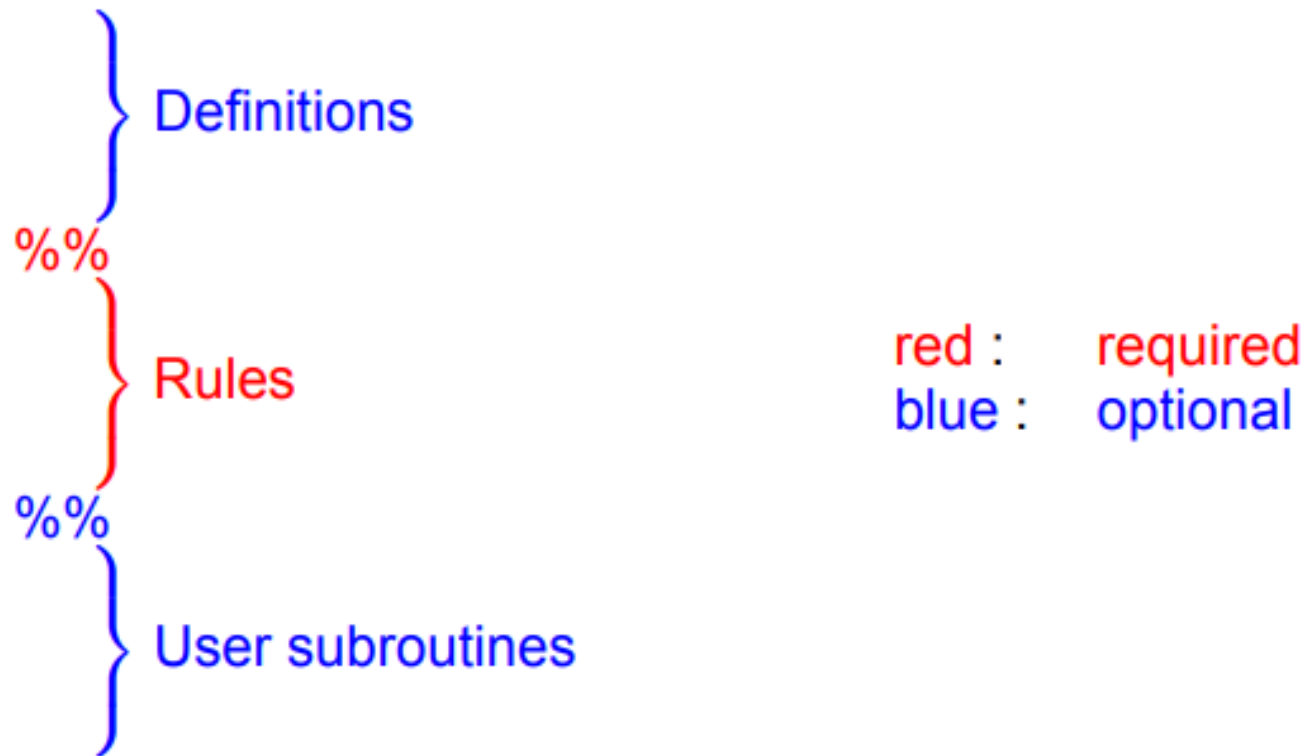
מבנה תכנית בשפת LEX :

```
{ definitions }  
%%  
{ rules }  
%%  
{ user subroutines }
```

- תכנית **LEX** מורכבת מ - 3 חלקים : הגדרות, כללים ופרוצדורות. הסימן **%%** (**double percent signs**) מפריד בין החלקים.
- החלק היחיד שהוא הכרחי הוא החלק של הכללים (**rules**).
- בהגדרות (**definitions**) אנו נצהיר על משתנים וקבועים שנשתמש בהם בכללים.
- בפרוצדורות (**subroutines**) נגדיר פרוצדורות בהן אנו צריכים להשתמש בכללים.

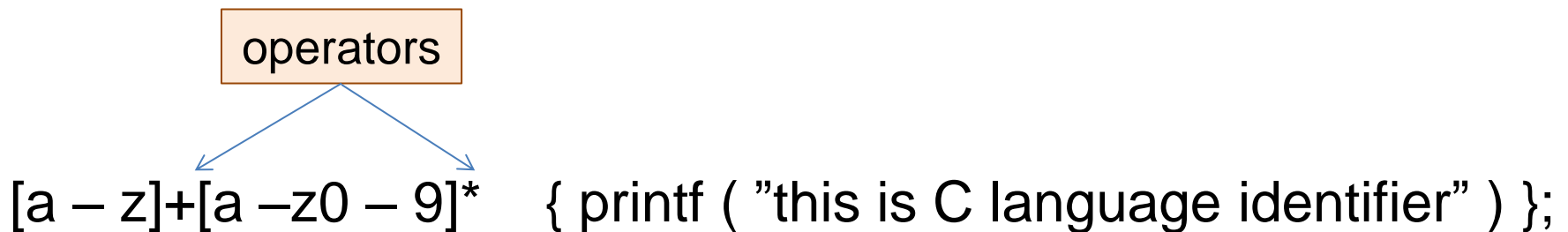
LEX

Structure of Lex Specification File



LEX

- The rules represent the user's control decisions, they are a table, in which the *left column contains regular expressions* and the *right column contains C program fragments*.
- We specify the *lexeme* you are interested in with a notation called a *regular expression*.
- A *regular expression* contains text characters (which match the corresponding characters in the *lexeme*) and operator characters.



LEX

ביטויים רגולריים של Lex

משמעות	ביטוי רגולרי "רגיל"	ביטוי רגולרי של LEX
התו a	a	a
כל תו פרט לירידת שורה	$\Sigma \setminus \{\backslash n\}$.
אחד מהתווים שבתוך הסוגריים	x+y+z a+b+c+...+z	[xyz] [a-z]
מספר כלשהו של z-ים כולל אפס / לא כולל אפס	r* r+	r* r+

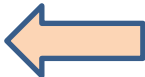
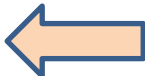
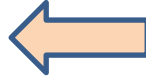
LEX

Regular expressions in `lex`

- `a` matches `a`
- `abc` matches `abc`
- `[abc]` matches `a`, `b` or `c`
- `[a-f]` matches `a`, `b`, `c`, `d`, `e`, or `f`
- `[0-9]` matches any digit
- `x+` matches one or more of `x`
- `x*` matches zero or more of `x`
- `[0-9]+` matches any integer
- `(...)` grouping an expression into a single unit
- `|` alternation (or)
- `(a|b|c)*` is equivalent to `[a-c]*`

LEX

Regular expressions in `lex`

- `x?` `x` is optional (0 or 1 occurrence) 
- `if(def)?` matches `if` or `ifdef` (equivalent to `if|ifdef`)
- `[A-Za-z]` matches any alphabetical character
- `.` matches any character except newline character 
- `\.` matches the `.` character
- `\n` matches the newline character 
- `\t` matches the tab character
- `\\` matches the `\` character
- `[\t]` matches either a space or tab character
- `[^a-d]` matches any character other than `a`, `b`, `c` and `d`

LEX

Examples

Real numbers, e.g., 0, 27, 2.10, .17

$[0-9]^* (\backslash .) ? [0-9]^+$

To include an optional preceding sign:

$[+-] ? [0-9]^* (\backslash .) ? [0-9]^+$

Integer or floating point number

$[0-9]^+ (\backslash . [0-9]^+) ?$


Integer, floating point, or scientific notation.

$[+-] ? [0-9]^+ (\backslash . [0-9]^+) ? ([eE] [+-] ? [0-9]^+) ?$

LEX

דוגמה 1:


```
%%  
int|INT { printf ("found keyword INTEGER\n" ) ; }  
.|\\n ;  
%%
```



C program fragments.

דוגמה 2:

```
%%  
[a - z]+[a - z0 - 9]* { printf ("this is C language identifier" ) } ;  
[a - z0 - 9]* { printf ("this is not C language identifier" ) } ;  
%%
```



pattern/regexp

תבנית (pattern, regexp) ביטוי רגולרי שמגדיר את ההתאמה בין אוסף הלקסמות לאסימון מסוים.

LEX

לפחות רווח אחד חובה !

definitions

דוגמה 3:

→ int l1, l2;

%%

[a - z]+ {

l1 = **yyleng**; printf ("ytext length is %d\n" , l1) ;

l2 = myLength(**ytext**); printf ("My length is %d\n" , l2);

}

rules

%%

#include<string.h>

int myLength(char *str)

{

int myL;

myL = strlen(str);

return myL;

}

subroutines

LEX

Two Notes on Using Lex

1. Lex matches token with **longest match**

Input: *abc*

Rule: `[a-z]+`

→ Token: *abc* (not “a” or “ab”)

2. Lex uses the **first applicable rule**

for the Input: *post*

Rule1: `"post"` `{printf ("Hello,"); }`

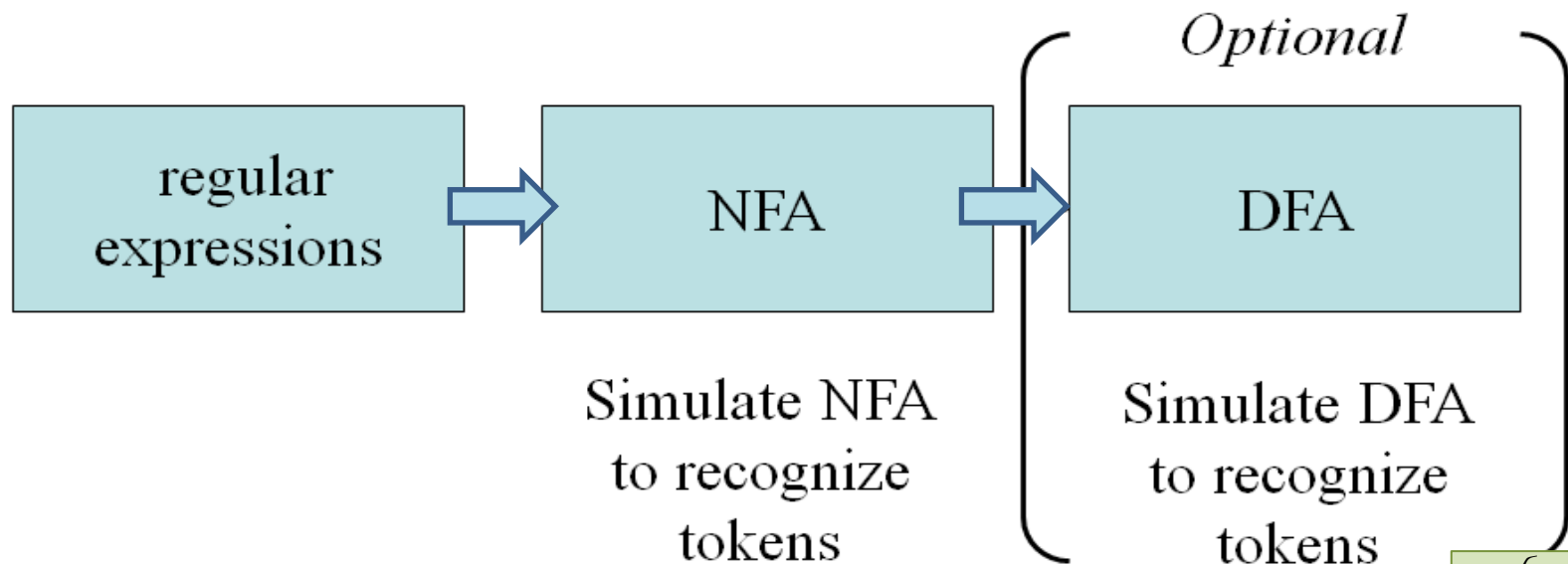
Rule2: `[a-zA-Z]+` `{printf ("World!"); }`

→ It will print Hello, (not “World!”)

LEX

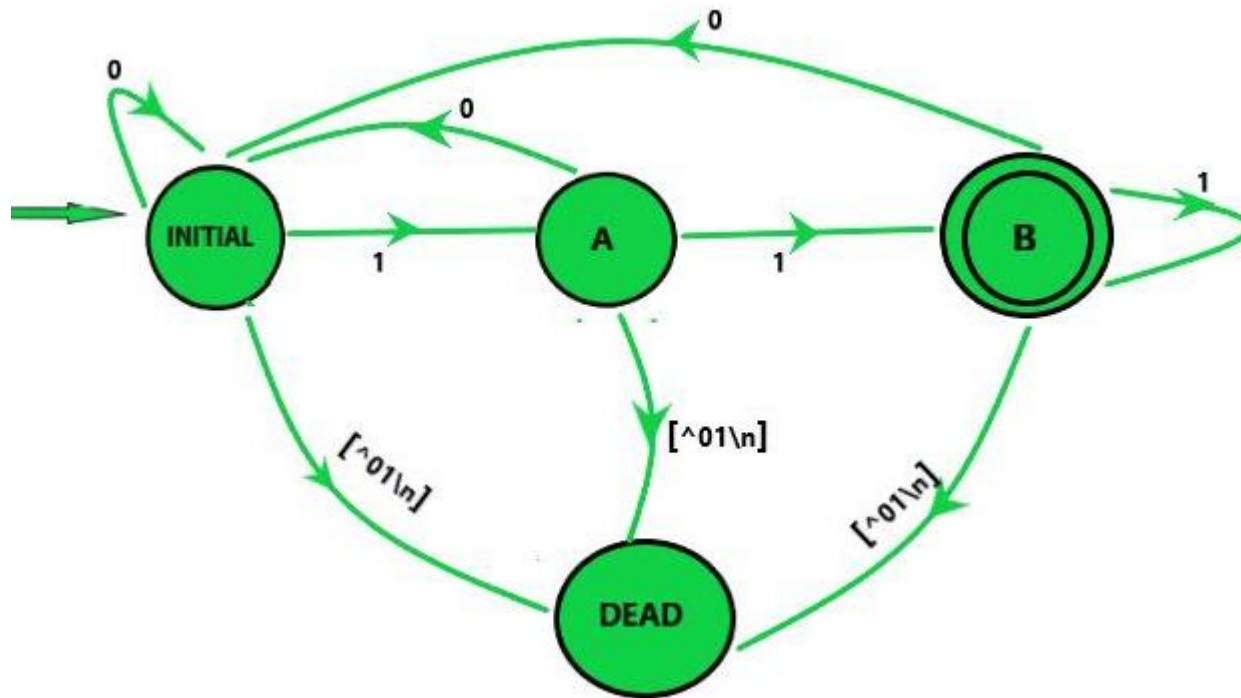
Design of a Lexical Analyzer Generator

- Translate regular expressions to NFA
- Translate NFA to an efficient DFA



LEX

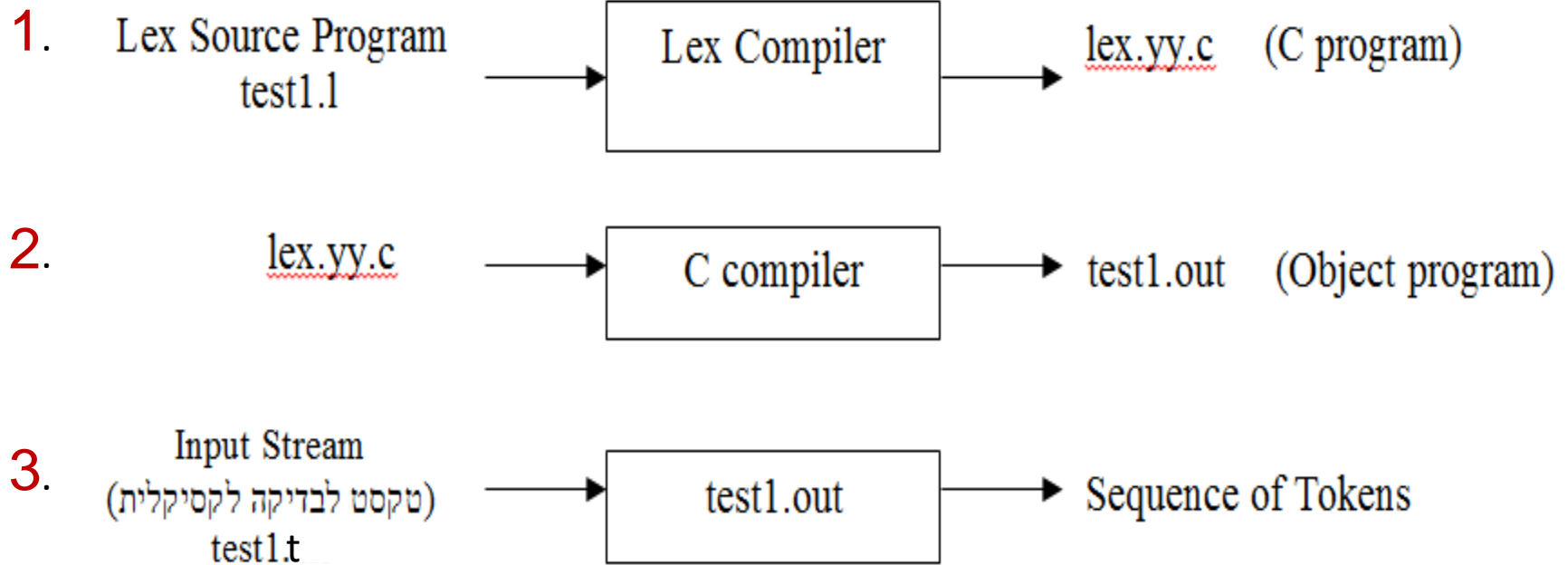
$\Sigma = \{ 0, 1 \}$



L ?

LEX

שלבי הרצת תכנית Lex



LEX

1. בעזרת עורך **nano** ניצור קובץ עם סיומת **.l** (או **lex**). נבצע לו קומפילציה של LEX על ידי הקשה של שם קובץ:

```
lex test1.l
```

2. כעת נבצע קומפילציה של C לקובץ **lex.yy.c** שנוצר מהשלב הקודם על ידי הקשה:

```
cc lex.yy.c -o test1 -ll
```

הקומפיילר של C ייצור קובץ בשם **test1** עם סיומת **.out**.

3. נבנה קובץ של טקסט לניתוח לקסיקלי **test1.t** (בעזרת עורך **nano**).
כעת נריץ את הקובץ (אפשר בלי סיומת **out**) שקיבלנו ונשלח אליו את קובץ הטקסט:

```
./test1 < test1.t
```

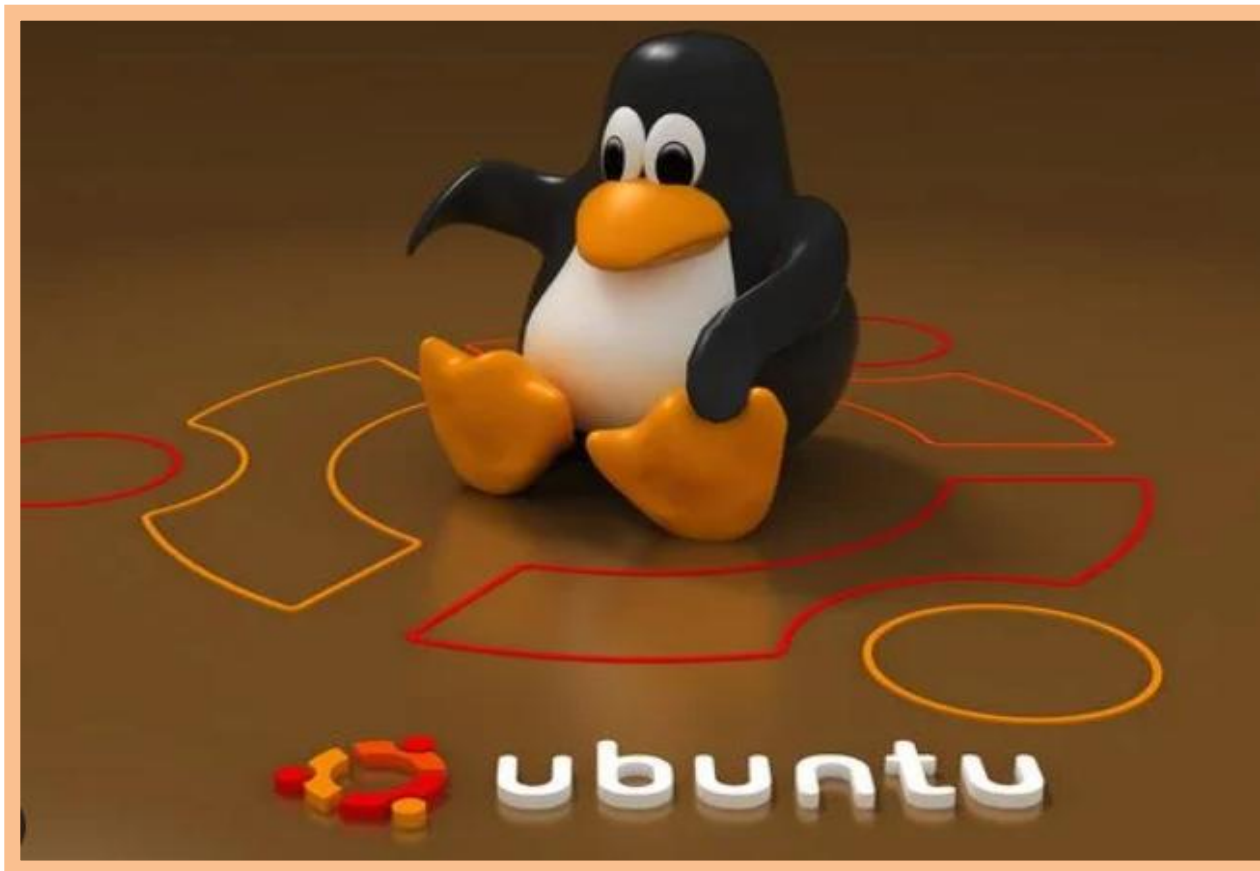
התקנת סביבה

- *VMware Workstation Player* is a desktop virtualization application that runs another operating system on the same computer *without rebooting*.
- Easily run multiple operating systems as virtual machines on your Windows PC.

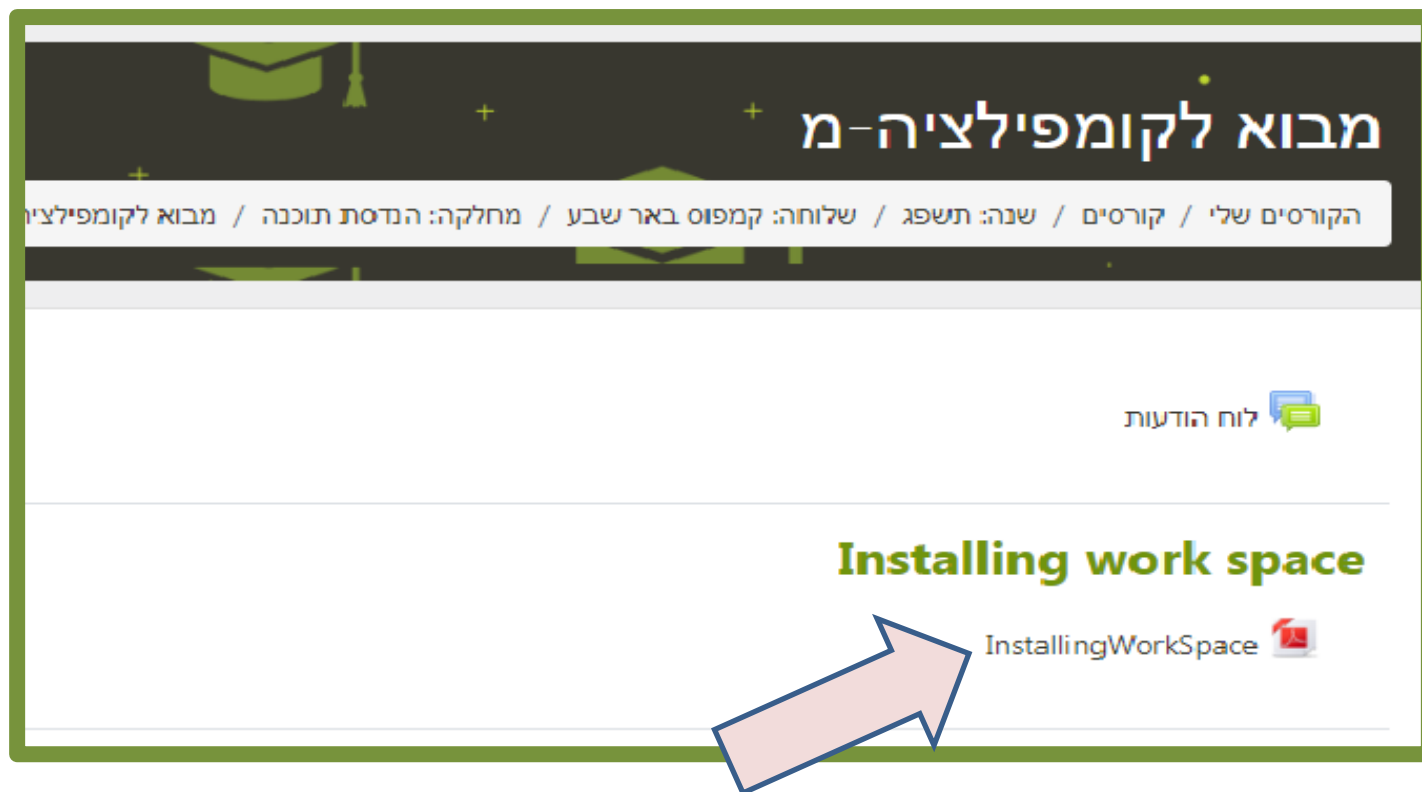


התקנת סביבה

- **Ubuntu** is a popular *Linux-based* operating system.
- According to Ubuntu's official website, it is the world's most widely used *Linux workstation* platform.



התקנת סביבה

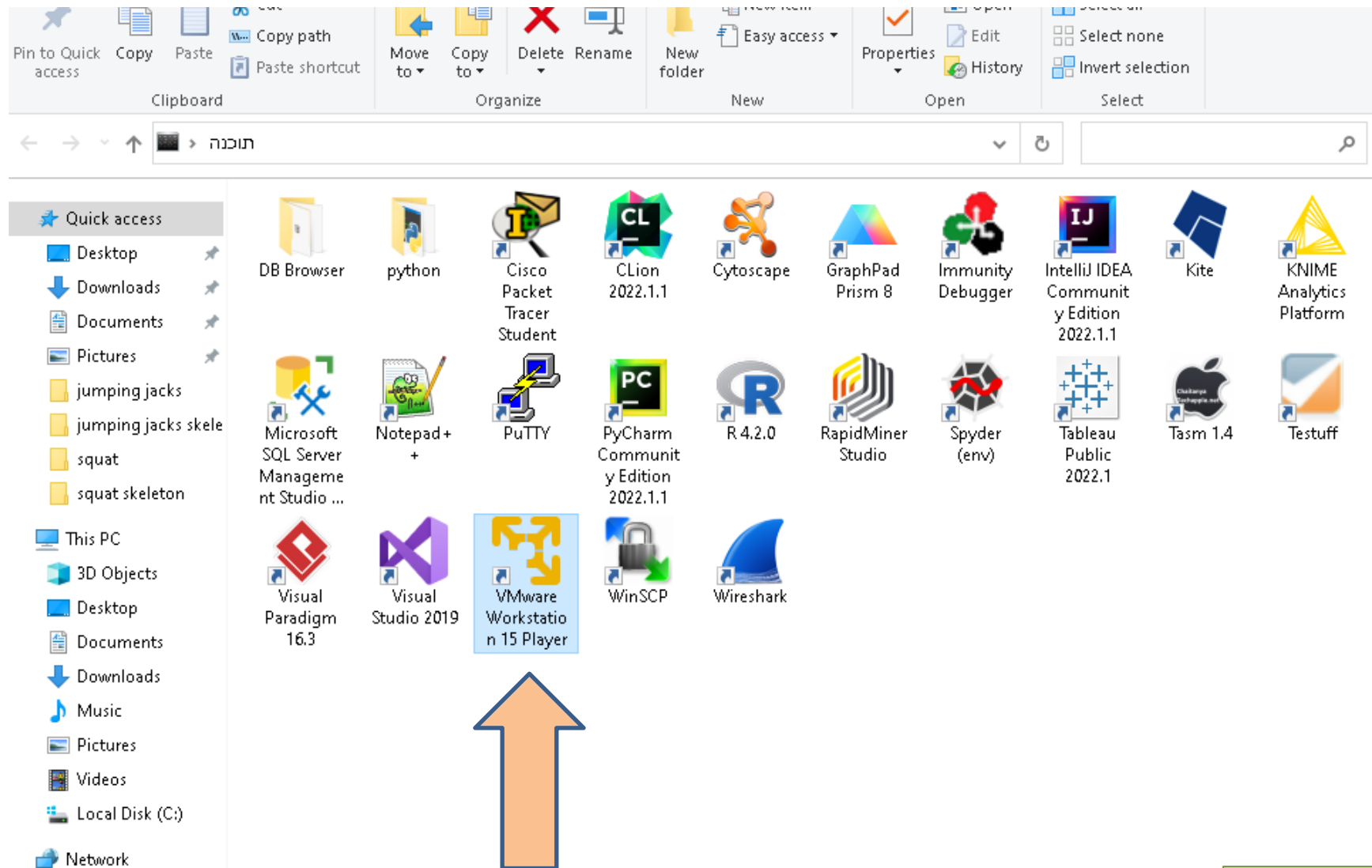


- מומלץ לעדכן במחשבים האישיים את סביבת העבודה המאפשרת את פיתוח הפרויקט בקורס מבוא לקומפילציה.

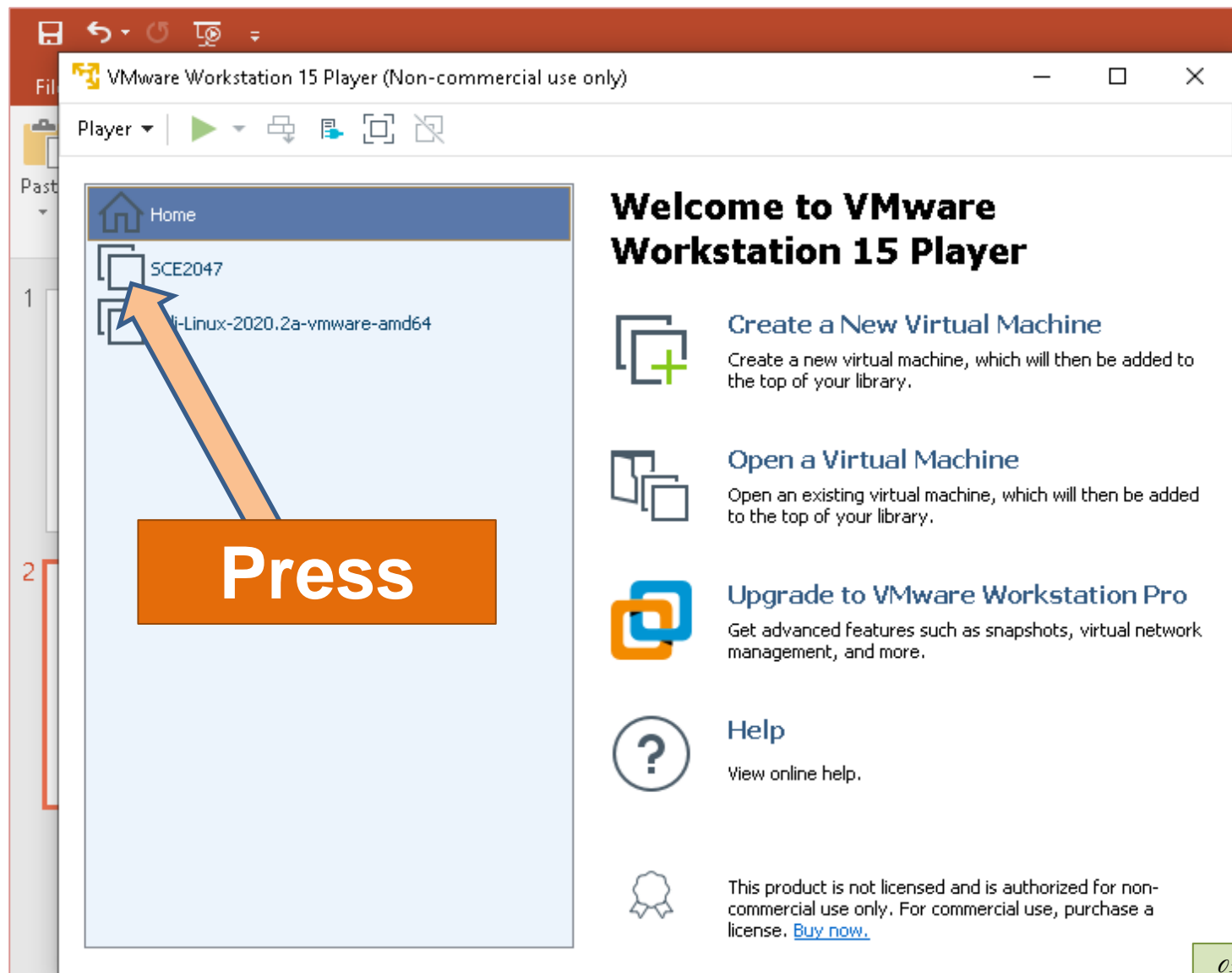
- במידה וישנה בעיה בהתקנה, יש לפנות לגנאדי קוגן במייל :

genadko@sce.ac.il

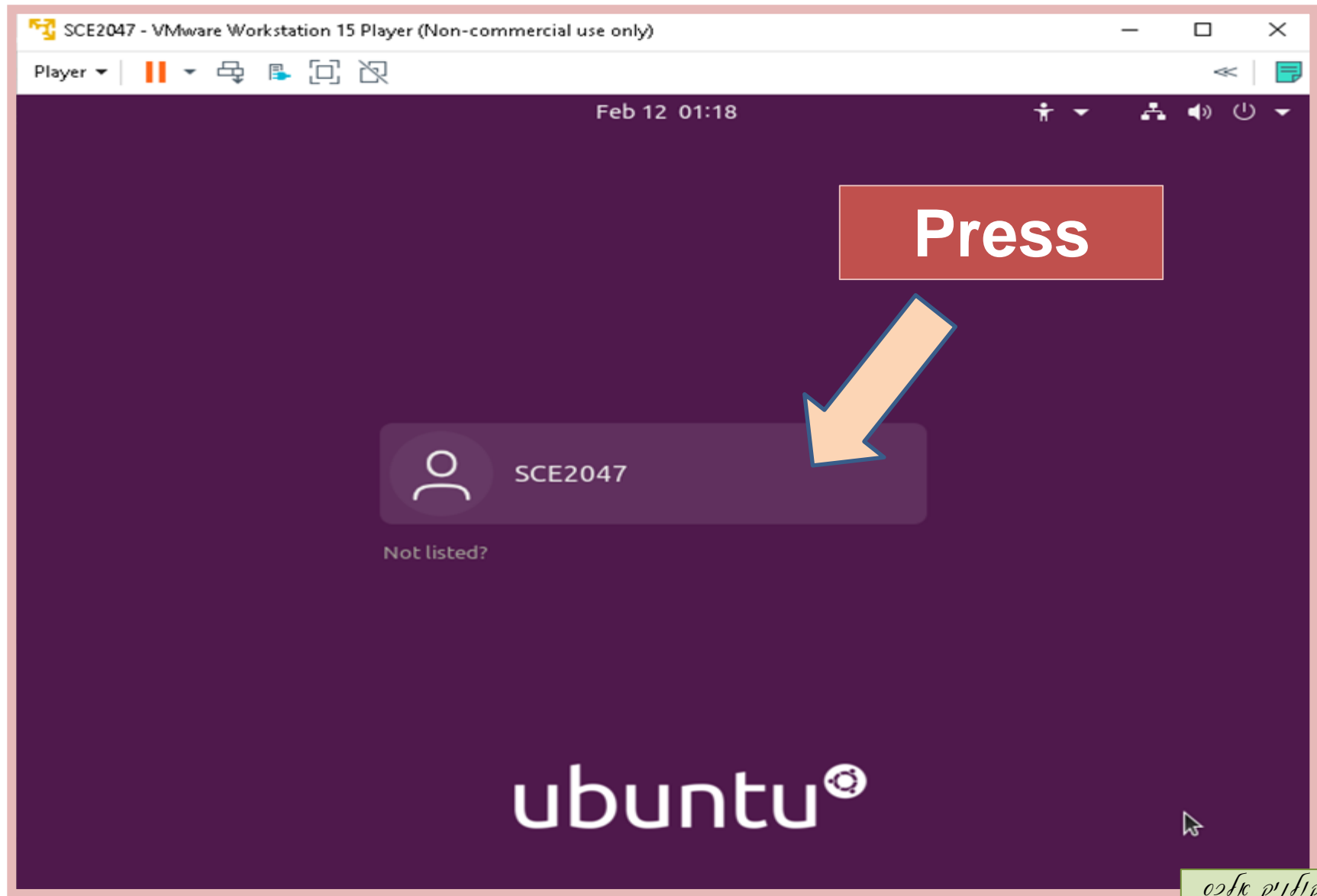
התקנת סביבה



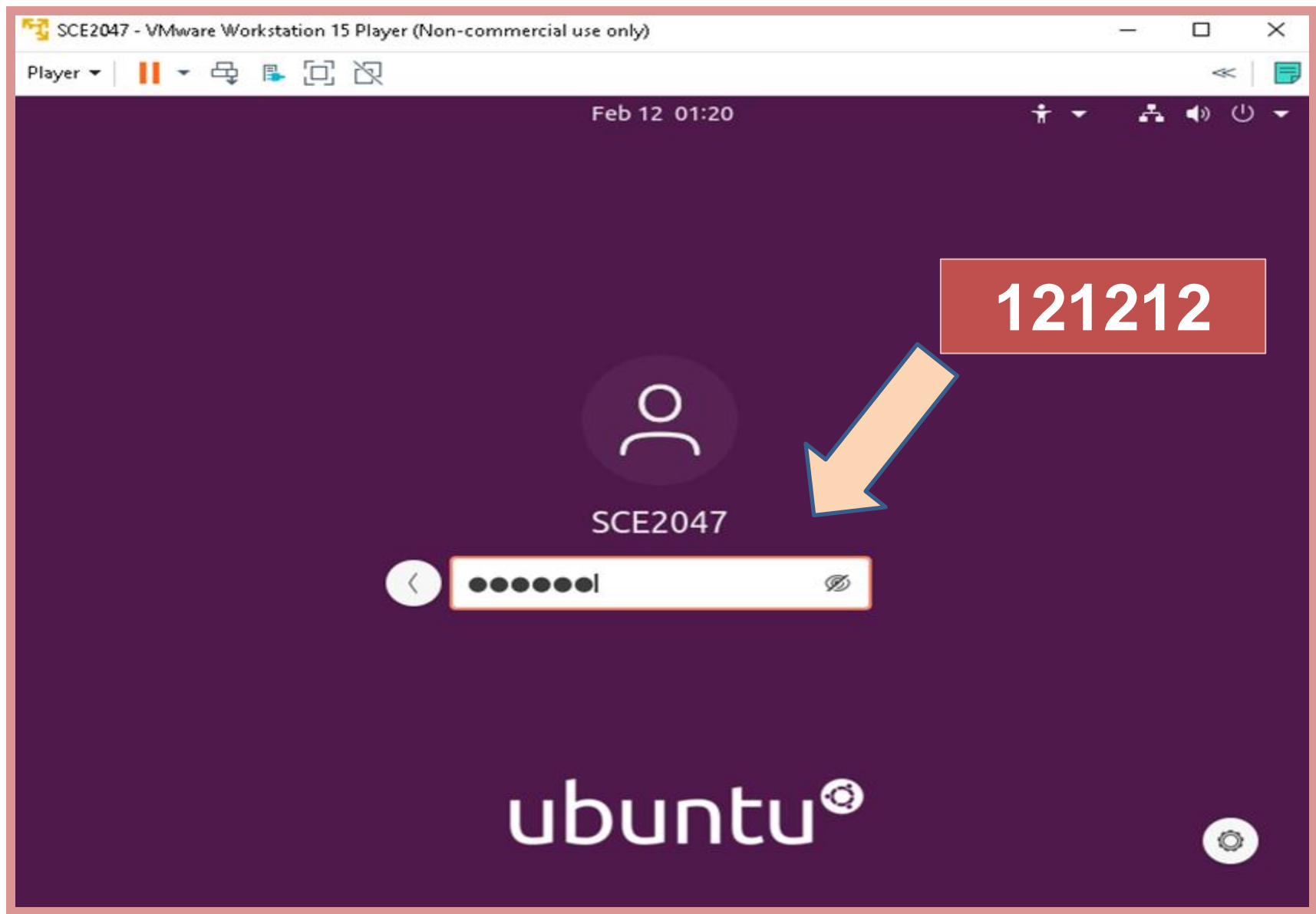
התקנת סביבה



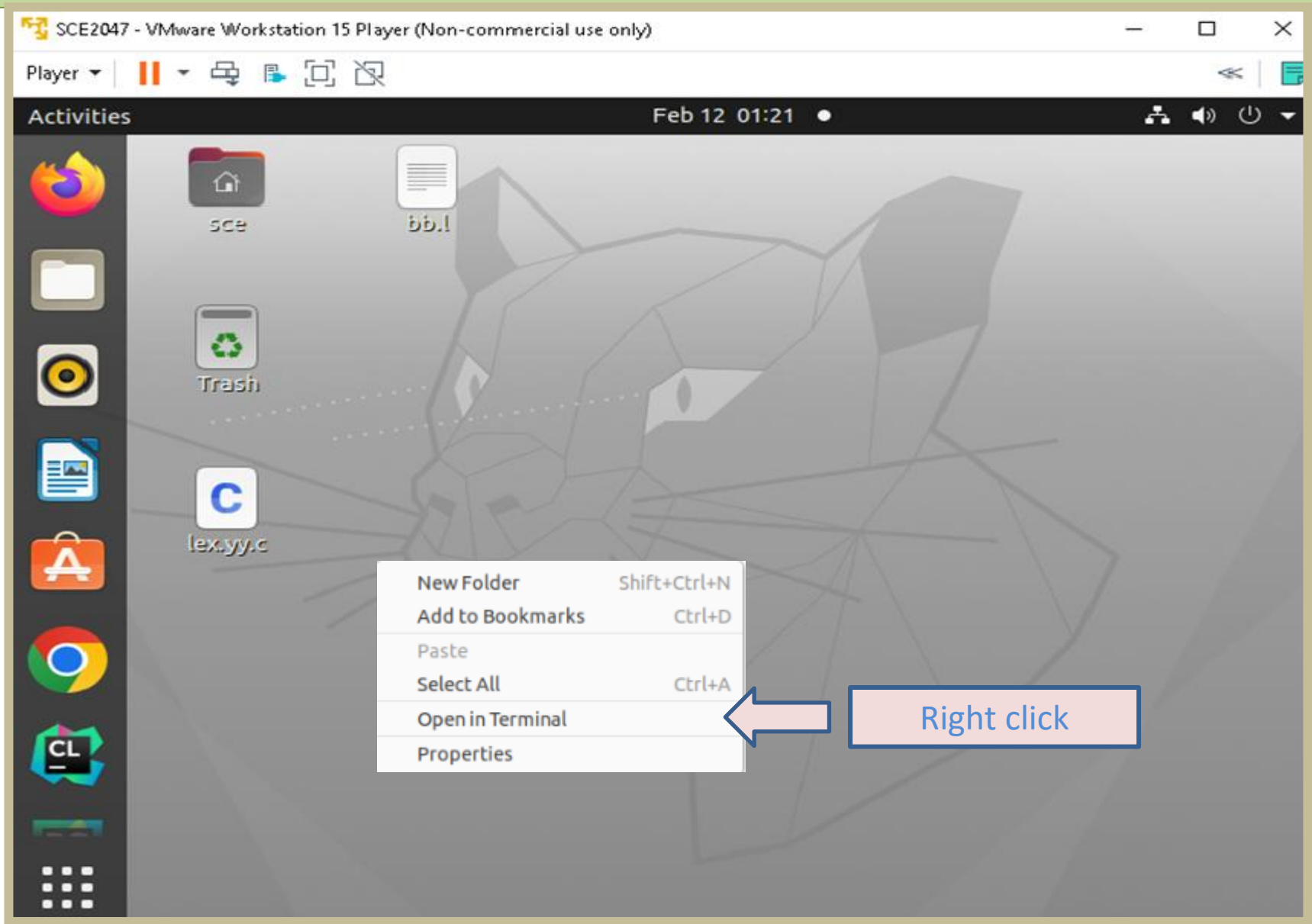
התקנת סביבה



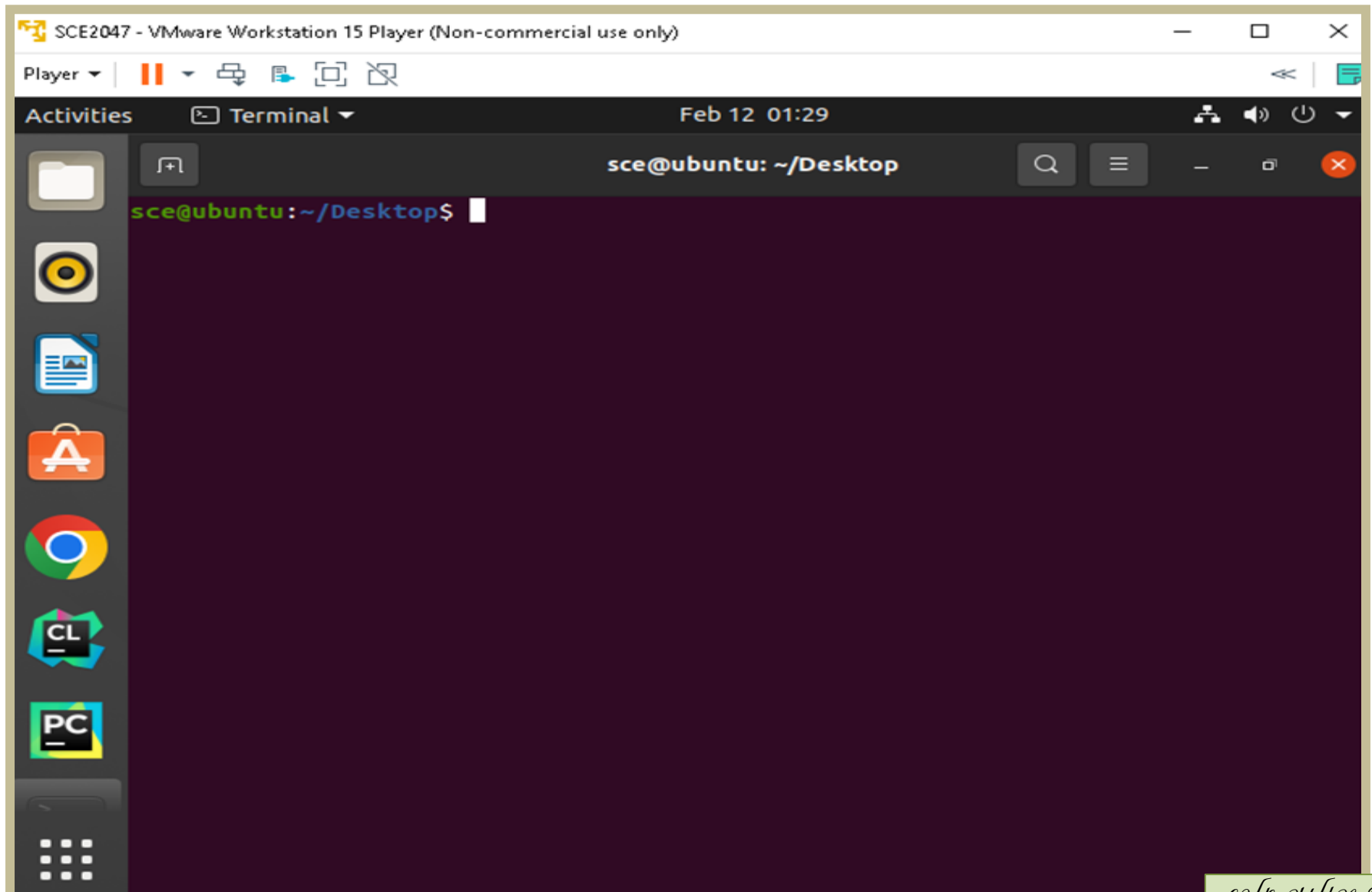
התקנת סביבה



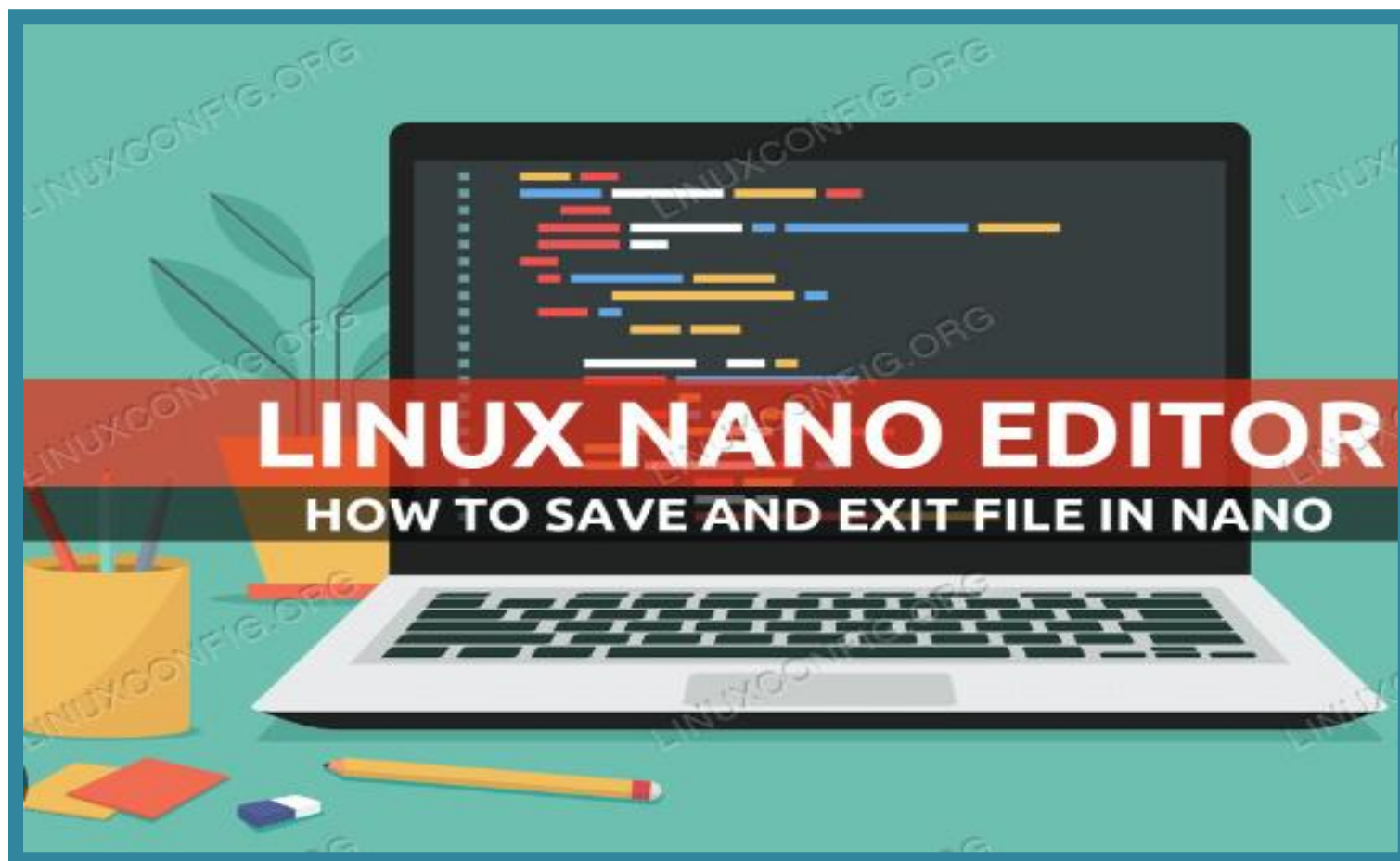
התקנת סביבה



התקנת סביבה

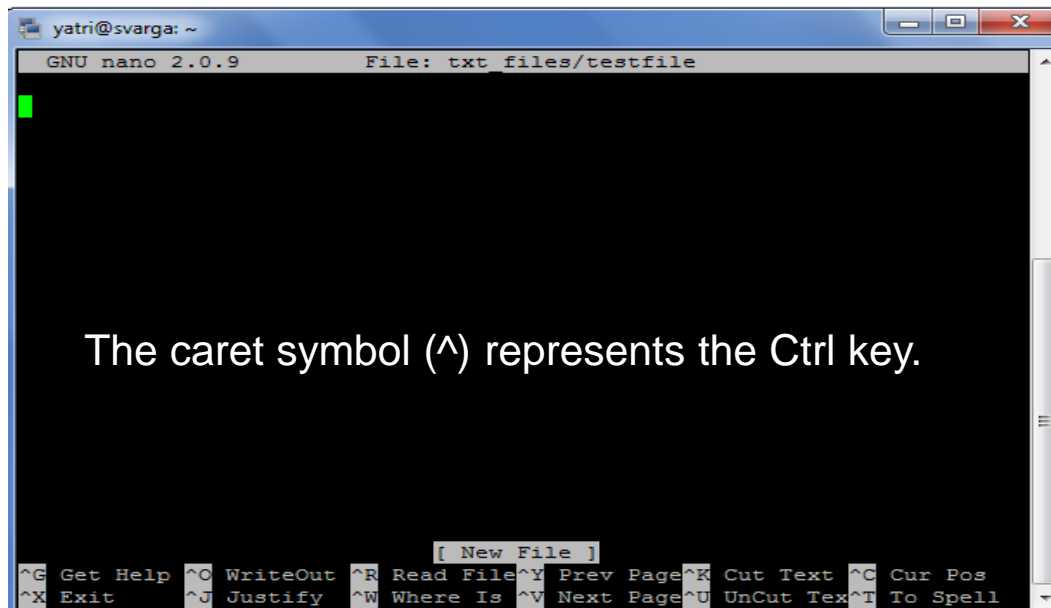


התקנת סביבה



תכנית ראשונה

- על מנת לכתוב ולשמור את הטקסטים השונים מומלץ להשתמש בתכנית עריכה **nano**.



למשל, כדי לשמור או לעדכן טקסט תכנית **test1.l** בשפת LEX פשוט מקלידים :
`nano test1.l`

בזמן עריכה, נוכל לשמור את הקובץ הנוכחי על ידי לחיצה על צירוף המקשים **Ctrl + X**.
כדי להציג את רשימת הקבצים בספרייה הנוכחית נשמש בפקודה: **ls**.

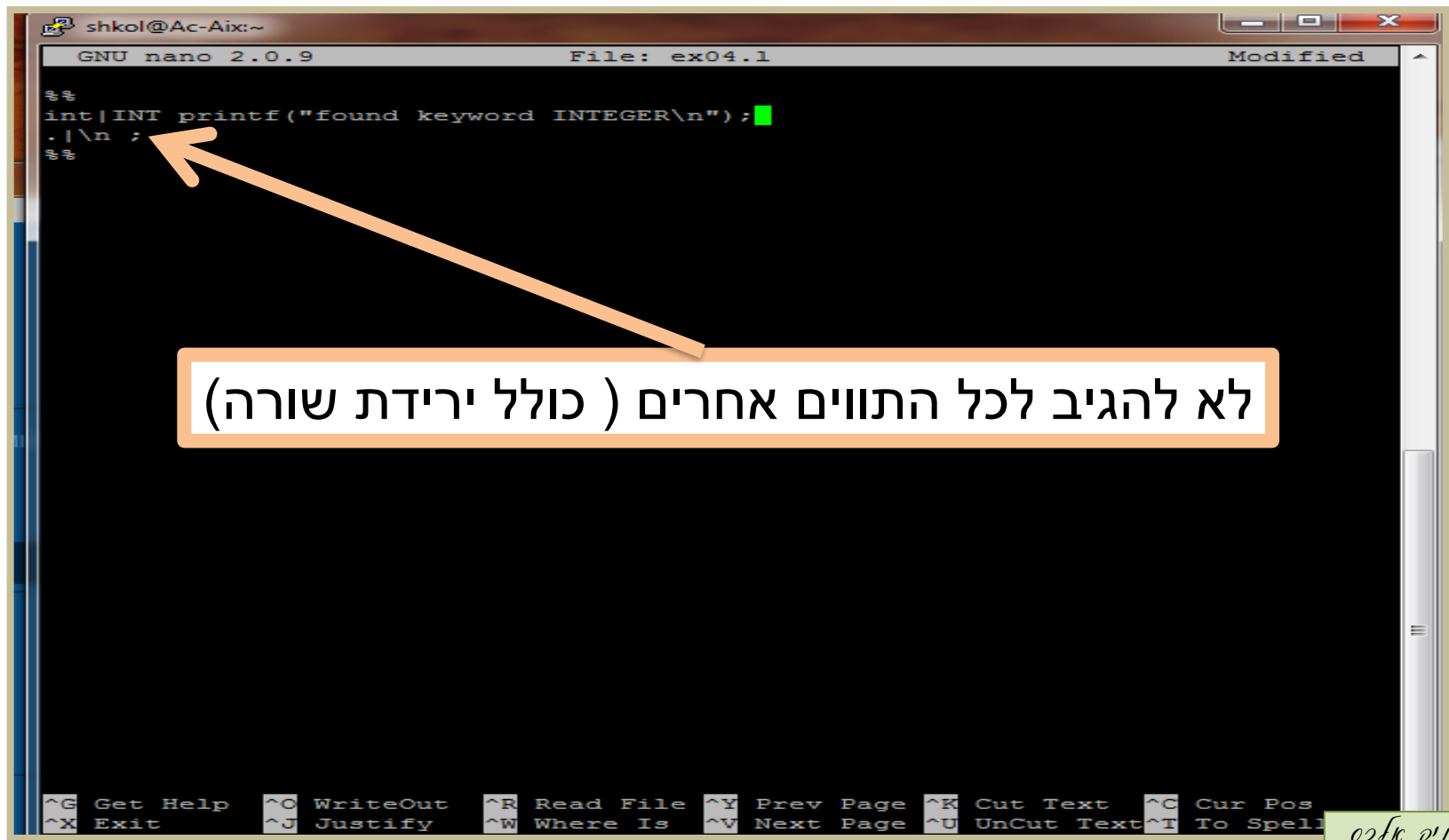
<https://linuxize.com/post/how-to-use-nano-text-editor/>

תכנית ראשונה

הרצת מנתח לקסיקלי LEX עבור **דוגמה 1**

1 - יצירת קובץ test1.l (בעזרת עורך nano) **nano test1.l**

2 - וביצוע LEX קומפילציה **lex test1.l**



```
shkol@Ac-Aix:~  
GNU nano 2.0.9 File: ex04.1 Modified  
%%  
int|INT printf("found keyword INTEGER\n");  
.|\n ;  
%%
```

לא להגיב לכל התווים אחרים (כולל ירידת שורה)

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

תכנית ראשונה

הרצת מנתח לקסיקלי LEX עבור **דוגמה 1**

3 - יצירת קובץ test1.t (בעזרת עורך nano) `nano test1.t`

4 - וביצוע C קומפילציה `cc lex.yy.c -o test1 -ll`

```
shkol@Ac-Aix:~  
GNU nano 2.0.9 File: ex04.t  
ccFX int 333 INT 666 #12%%* int abcd  
[ Read 1 line ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

3 הופעות של המילה
int/INT

© שקלאניק ארכיט

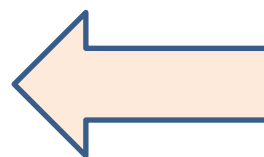
תכנית ראשונה

הרצת מנתח לקסיקלי LEX עבור **דוגמה 1**

5 - הרצת תכנית test1.out לביצוע ניתוח לקסיקלי עבור נתוני קובץ test1.t :

./test1<test1.t

```
[shkol@Ac-Aix ~]$ ./test1<test1.t  
found keyword INTEGER  
found keyword INTEGER  
found keyword INTEGER  
[shkol@Ac-Aix ~]$
```



בהתאם לנתוני קובץ טקסט test1.t
המשפט "found keyword INTEGER"
מודפס **3** פעמים !