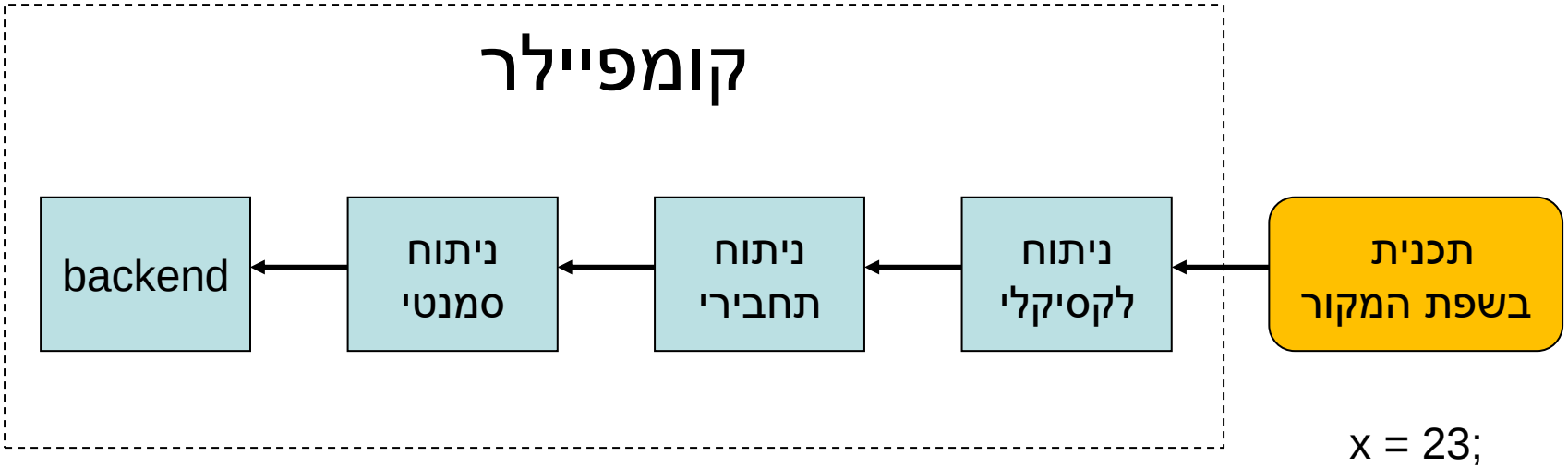
ניתוח לקסיקלי וכלי Lex

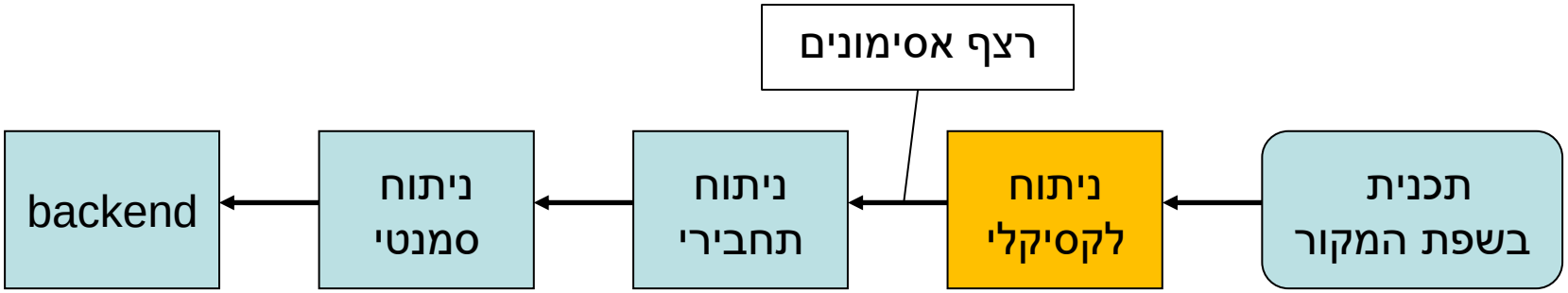
עודכן סמסטר חורף 2021/22

מבנה הקומפיילר



(רצף של תווים)

מבנה הקומפיילר

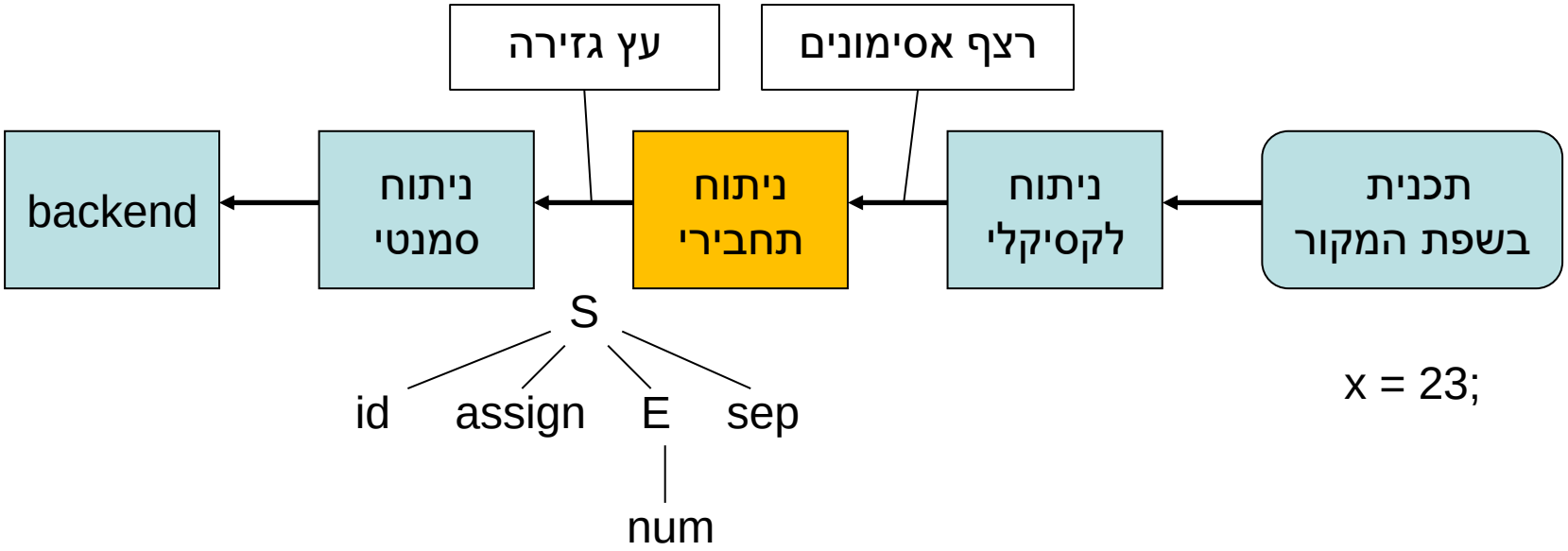


x = 23;

id assign num sep

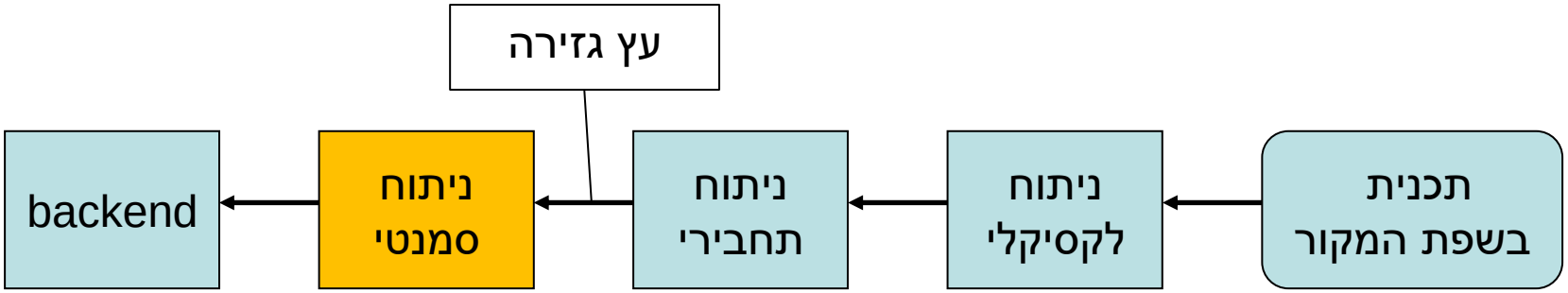
x = 23 ;

מבנה הקומפיילר



(העלים של עץ הגזירה הם האסימונים
שהתקבלו מהניתוח הלקסיקלי)

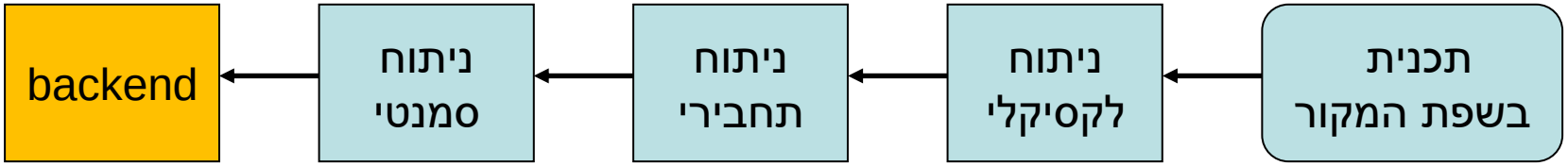
מבנה הקומפיילר



האם קיים משתנה ששמו x?
האם x מטיפוס int?

x = 23;

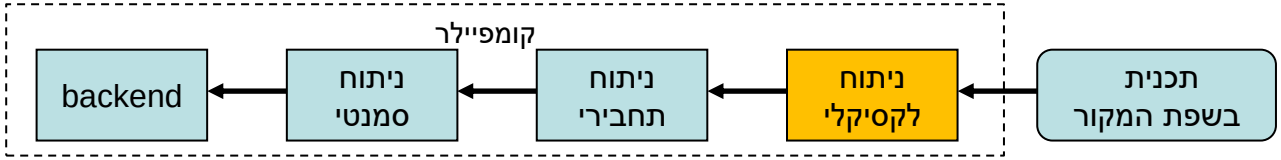
מבנה הקומפיילר



x = 23;

ייצור קוד בשפת היעד

תפקיד המנתח הלקסיקלי



- מחלק את קוד המקור לאסימונים ("מילים")
- מסנן חלקים שאינם דרושים להמשך הניתוח
  - למשל: רווחים, ירידות שורה, הערות.
- מזהה שגיאות לקסיקליות - מחרוזות שאינן יכולות להיות אף אסימון
  - למשל: "@" בשפת C

הגדרות

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

המנתח הלקסיקלי - דוגמה

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

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

המנתח לא יחזיר אסימון

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

דוגמה: ניתן להגדיר את האסימון num ע"י התבנית (0+1+…+9)+

המנתח הלקסיקלי ותכונות סמנטיות

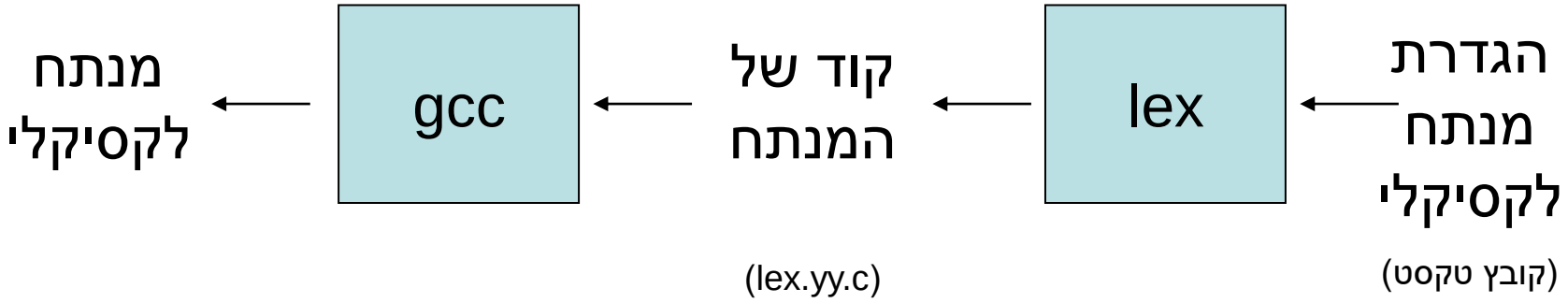
- הלקסמות אינן עוברות הלאה לשלבים הבאים בניתוח, אלא רק האסימונים שנוצרו מהן
  - המנתח התחבירי אינו צריך את הלקסמות.
  - המנתח הסמנטי כן צריך מידע עליהן:
    - האם קיים משתנה ששמו x?
    - מה הטיפוס שלו?

- לטובת העברת המידע על הלקסמות, המנתח הלקסיקלי מחשב לכל אסימון תכונות סמנטיות:
  - id {name = “x”}
  - op {type = “+”}

- התכונות הסמנטיות ידונו בהרחבה בהמשך.

כלי Lex

- כלי <u>לייצור</u> מנתחים לקסיקליים.
- צורת הפעלה:



- בקורס נעבוד עם כלי תואם הנקרא Flex

בניית מנתח לקסיקלי בעזרת Lex

1)עריכת קובץ הגדרת המנתח (source.lex) בקובץ טקסט.

2)הרצת הפקודה `flex source.lex`

– הפלט המתקבל: קובץ C בשם lex.yy.c

3)הרצת הפקודה `gcc -ll lex.yy.c`

לטובת ייצור קובץ ההרצה.

מבנה קובץ הגדרות המנתח

- קובץ הגדרות הוא קובץ טקסט המורכב משלושה חלקים המופרדים בעזרת שורות המכילות "%%" בלבד:

```
Definition section
%%
Rules section
%%
C code section
```

דוגמה למבנה קובץ הגדרות המנתח

```
1   %{
2   /* Declarations section */
3   #include <stdio.h>
4   void showToken(char *);
5   %}
6
7   %option yylineno
8   %option noyywrap
9   digit            ([0-9])
10  letter           ([a-zA-Z])
11  whitespace       ([\t\n ])
12
13  %%
14  {digit}+                      showToken("number");
15  {letter}+@{letter}+\.com      showToken("email address");
16  {whitespace}                  ;
17  .        printf("Lex doesn't know what that is!\n");
18  %%
19
20  void showToken(char * name) {
21          printf("Lex found token %s , name);
22          printf("the lexeme is %s , yytext);
23          printf("its length is %d\n", yyleng);
24  }
```

דוגמה למבנה קובץ הגדרות המנתח

Definitions section

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit            ([0-9])
10 letter           ([a-zA-Z])
11 whitespace       ([\t\n ])
12
13 %%
14 {digit}+                    showToken("number");
15 {letter}+@{letter}+\.com    showToken("email address");
16 {whitespace}                ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

דוגמה למבנה קובץ הגדרות המנתח

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit           ([0-9])
10 letter          ([a-zA-Z])
11 whitespace      ([\t\n ])
12
13 %%
14 {digit}+                      showToken("number");
15 {letter}+@{letter}+\.com      showToken("email address");
16 {whitespace}                  ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

Rules section

דוגמה למבנה קובץ הגדרות המנתח

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit            ([0-9])
10 letter           ([a-zA-Z])
11 whitespace       ([\t\n ])
12
13 %%
14 {digit}+                     showToken("number");
15 {letter}+@{letter}+\.com     showToken("email address");
16 {whitespace}                 ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

C code section

Definitions section

הגדרות של שפת C - קוד זה מועתק כפי שהוא לתחילת קובץ ה-C ש-flex מייצר.

אופציות השולטות על צורת העבודה של Flex

הגדרת מקרואים (macros) בעזרת ביטויים רגולריים (לשימוש בחלק הבא)

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit            ([0-9])
10 letter           ([a-zA-Z])
11 whitespace       ([\t\n ])
12
13 %%
14 {digit}+                        showToken("number");
15 {letter}+@{letter}+\.com        showToken("email address");
16 {whitespace}                    ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

ביטויים רגולריים של Lex

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

דוגמאות נוספות ניתן למצוא באתר הקורס

Rules section

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit            ([0-9])
10 letter           ([a-zA-Z])
11 whitespace       ([\t\n ])
12
13 %%
14 {digit}+                      showToken("number");
15 {letter}+@{letter}+\.com      showToken("email address");
16 {whitespace}                  ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

Rules section

הגדרות של
אסימונים ופעולות
ש-Lex צריך לבצע
בעת זיהוי שלהם

משתנים גלובליים של Lex

שם	טיפוס	משמעות
yytext	char *	הטקסט של הלקסמה האחרונה שזוהתה
yyleng	int	אורך הלקסמה האחרונה שזוהתה
yylineno	int	השורה הנוכחית בקלט
yylval	user defined	משמש לתקשורת עם המנתח התחבירי (פרטים בתרגול מספר 4)

C code section

```
1  %{
2  /* Declarations section */
3  #include <stdio.h>
4  void showToken(char *);
5  %}
6
7  %option yylineno
8  %option noyywrap
9  digit            ([0-9])
10 letter           ([a-zA-Z])
11 whitespace       ([\t\n ])
12
13 %%
14 {digit}+                    showToken("number");
15 {letter}+@{letter}+\.com    showToken("email address");
16 {whitespace}                ;
17 .        printf("Lex doesn't know what that is!\n");
18 %%
19
20 void showToken(char * name) {
21         printf("Lex found token %s , name);
22         printf("the lexeme is %s , yytext);
23         printf("its length is %d\n", yyleng);
24 }
```

C code section

הגדרות פונקציות
שהוכרזו בחלק ראשון

משתנים גלובליים
של Lex

תכונות המנתח הלקסיקלי ש-Lex בונה

- קורא קלט מ-stdin וכותב פלט ל-stdout.
- רוב העבודה נעשית בפונקציה yylex (נמצאת בקובץ lex.yy.c שנוצר ע"י flex) שתפקידה:
  - לקרוא את הקלט.
  - לזהות אסימונים.
  - לבצע את הפעולה המתאימה לאסימון (ע"פ קובץ ההגדרות).
- yylex חוזרת רק כאשר:
  - המשתמש כתב return בפעולה של אסימון.

  או
  - מגיעה לסוף הקלט. (מהמקלדת: "ctrl + d")
- אם רוצים סיום ריצה לא מתוכנן: exit()

טיפול בשגיאות ניתוח

- שגיאת ניתוח נוצרת כאשר בנקודה מסוימת yylex לא מצליחה לזהות אף אסימון.
- במקרה כזה, פעולת ברירת המחדל היא העתקת התו הנוכחי לפלט. הניסיון להתאמת אסימונים ימשיך מהתו הבא.
- דוגמה:

abc$178

מזוהה כ-id | מזוהה כ-num

לא מזוהה ולכן מודפס לפלט

טיפול בשגיאות ניתוח (המשך)

- כדי לשנות את ברירת המחדל לטיפול בשגיאה, ניתן להשתמש באסימון הנקודה (.) בתור אסימון אחרון ב `Rules section`.
  - כפי שנעשה בדוגמה:

```
13  %%
14  {digit}+                        showToken("number");
15  {letter}+@{letter}+\.com        showToken("email address");
16  {whitespace}                    ;
17  .        printf("Lex doesn't know what that is!\n");
18  %%
```

פתרון קונפליקטים

- קונפליקט נוצר כאשר אותו קלט יכול להתאים למספר אסימונים.
- דוגמה: נניח שמוגדרים האסימונים הבאים:

for “for”

id [a-z]+

מחרוזות שיגרמו לקונפליקט	ניתוחים אפשריים
abc	
ford	
for	

פתרון קונפליקטים

- קונפליקט נוצר כאשר אותו קלט יכול להתאים למספר אסימונים.
- דוגמה: נניח שמוגדרים האסימונים הבאים:

for “for”

id [a-z]+

מחרוזות שיגרמו לקונפליקט	ניתוחים אפשריים
abc	a, ab, abc
ford	
for	

פתרון קונפליקטים

- קונפליקט נוצר כאשר אותו קלט יכול להתאים למספר אסימונים.
- דוגמה: נניח שמוגדרים האסימונים הבאים:

for “for”

id [a-z]+

מחרוזות שיגרמו לקונפליקט	ניתוחים אפשריים
abc	a, ab, abc
ford	for, ford, f, fo
for	

פתרון קונפליקטים

- קונפליקט נוצר כאשר אותו קלט יכול להתאים למספר אסימונים.
- דוגמה: נניח שמוגדרים האסימונים הבאים:

for “for”

id [a-z]+

ניתוחים אפשריים	מחרוזות שיגרמו לקונפליקט
a, ab, abc	abc
for, ford, f, fo	ford
for (id) / for (for)	for

הכללים לפתרון הקונפליקט

1. המנתח חמדן – תמיד מעדיף את הלקסמה <u>הארוכה ביותר</u> שניתן לבחור.
2. אם כלל (1) לא פתר את הקונפליקט, בוחרים באסימון בעל עדיפות גבוהה יותר:

- האסימון הראשון שמופיע בקובץ ההגדרות (העליון ביותר המתאים ללקסמה).

פתרון הדוגמה

- נניח שמוגדרים האסימונים הבאים:

for “for”

id [a-z]+

<u>הכללים לפתרון קונפליקט</u>

1. בחירת הלקסמה הארוכה ביותר.
2. אם כלל (1) לא פתר - בחירת אסימון בעל עדיפות גבוהה יותר.

לפי כלל	הניתוח שייבחר	ניתוחים אפשריים	מחרוזות שיגרמו לקונפליקט
1	abc	a, ab, abc	abc
1	ford	for, ford, f, fo	ford
2	for (for)	for (id) / for (for)	for

Start Conditions

- מאפשר הפעלת חוקים סלקטיבית
- נגדיר "מצבים" למנתח כך שנוכל לבחור איזה חוקים רלוונטיים לכל מצב
  - כחלק מהטיפול בחוקים נצטרך גם להגדיר את המעברים בין המצבים
- דוגמה: קלוט רצפים של אותיות, אבל בין סולמיות קלוט רק אותיות קטנות

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+                showToken("outside");
#                        BEGIN(HASHTAG);
<HASHTAG>{lower}+        showToken("inside");
<HASHTAG>#               BEGIN(INITIAL);
<HASHTAG>.               showToken("illegal");
%%
```

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+                showToken("outside");
#                        BEGIN(HASHTAG);
<HASHTAG>{lower}+        showToken("inside");
<HASHTAG>#               BEGIN(INITIAL);
<HASHTAG>.               showToken("illegal");
%%
```

הגדרת מצב חדש

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+        showToken("outside");
#                BEGIN(HASHTAG);
<HASHTAG>{lower}+  showToken("inside");
<HASHTAG>#       BEGIN(INITIAL);
<HASHTAG>.       showToken("illegal");
%%
```

כלל שיופעל רק במצב ההתחלתי
(ניתן לרשום גם כ: *<INITIAL>{letter}+*)

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+                showToken("outside");
#                        BEGIN(HASHTAG);
<HASHTAG>{lower}+        showToken("inside");
<HASHTAG>#               BEGIN(INITIAL);
<HASHTAG>.               showToken("illegal");
%%
```

מעבר למצב HASHTAG

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+          showToken("outside");
#                  BEGIN(HASHTAG);
<HASHTAG>{lower}+  showToken("inside");
<HASHTAG>#         BEGIN(INITIAL);
<HASHTAG>.         showToken("illegal");
%%
```

כלל שיופעל רק במצב HASHTAG.

ניתן להגדיר כלל עבור מספר מצבים באופן הבא:
<HASHTAG,OTHER_STATE>{lower}+

דוגמה

```
%{
void showToken(char * name) {
        printf("%s, %s, %d\n", name, yytext, yyleng);
}
%}

%option noyywrap
letter [a-zA-Z]
lower [a-z]

%x HASHTAG

%%
{letter}+                showToken("outside");
#                        BEGIN(HASHTAG);
<HASHTAG>{lower}+        showToken("inside");
<HASHTAG>#               BEGIN(INITIAL);
<HASHTAG>.               showToken("illegal");
%%
```

חזרה למצב התחלתי

מקורות נוספים

- דוגמאות לשימוש ב-Lex ניתן למצוא באתר הקורס
- אינפורמציה נוספת על Lex ועל ביטויים רגולריים:
  - man lex
  - man –s 5 regexp
- אתר הבית של כלי Flex:
  http://flex.sourceforge.net

שאלה ממבחן

1. חברכם לעבודה רשם את קטע ה-flex הבא :

```
c          {printf "1"}
ac+b*      {printf "2"}
cc         {printf "3"}
ab*        {printf "4"}
a          {printf "5"}
ac*b+      {printf "6"}
```

אתם, לעומתו, כבוגרי הקורס בקומפילציה, שמתם לב מיד שאחד או יותר מהחוקים לא יבוצע. מדוע?

```
c        {printf "1"}
ac+b*    {printf "2"}
cc       {printf "3"}
ab*      {printf "4"}
a        {printf "5"}
ac*b+    {printf "6"}
```

פתרון שאלה

- הכללים a ו+ac*b המתאימים להדפסות 5 ו 6 בהתאמה לעולם לא יופעלו.
- הכלל a לעולם לא יופעל, מכיוון שייתפס ע"י כלל *ab שנמצא בעדיפות גבוהה ממנו.
- הכלל +ac*b לעולם לא יופעל מכיוון:
  - עבור מילים מהצורה +ab המילים יתפסו ע"י כלל *ab.
  - עבור מילים מהצורה +ac+b המילים יתפסו ע"י כלל *ac+b.