תורת הקומפילציה

תרגיל בית 1 – שימוש ב-Flex

מתרגל אחראי לתרגיל: שקד ברודי

- תיקונים יהיו מסומנים בצהוב

ההגשה בזוגות.

יש להפנות שאלות על התרגיל במייל ל-shakedbr@cs בלבד.

לתרגיל ייפתח דף FAQ באתר הקורס. כל הבהרה שתופיע בו עד יומיים ממועד ההגשה תהווה הוראה מחייבת.

הנחיות כלליות

- בתרגיל זה תממשו מנתח לקסיקלי שיוכל לטפל בשפת FanC. שפה זו היא subset של שפת C שאתם מכירים, הכוללת פעולות אריתמטיות, פונקציות, המרות ועוד.
- במנתח הלקסיקלי שתממשו נשתמש כדי ליצור את שתי התוכניות הבאות:
  - חלק א: תוכנית הקוראת קלט מהמשתמש ומדפיסה מידע על האסימונים שהיא מצאה.
  - חלק ב: תוכנית הממשת מחשבון מסוג Prefix.
- התרגיל ייבדק אוטומטית. **הקפידו למלא אחר ההוראות במדויק.** הבדיקה תתבצע על csComp.
- יש להשתמש ב-flex בלבד (ולא ב-lex).
- קראו את כל התרגיל פני התחלתו שכן, נרצה שמימוש המנתח יתאים לשתי חלקי התרגיל.

הגדרת מושגים כלליים

- רווח לבן: אחד מבין: רווח (ספייס), טאב, CR (התו \r), LF (התו \n).
- תווים ניתנים להדפסה: התווים שערך ה-ascii שלהם בין 0x20 ל-0x7E, או רווחים לבנים: טאב (0x09), LF (0x0A), CR (0x0D) (רווח רגיל נכלל בתוך הטווח).
  - ניתן לקרוא על תווים ניתנים להדפסה בהרחבה בוויקיפדיה בערך הבא: https://en.wikipedia.org/wiki/ASCII#Printable_characters
- רצף בריחה (escape sequence): לוכסן אחורי (התו \) ואחריו תו או יותר שביחד מפורשים כתו אחר.
  - דוגמאות: \n – ירידת שורה, \t – טאב.
  - ניתן לקרוא על רצפי בריחה בהרחבה בוויקיפדיה בערך הבא: https://en.wikipedia.org/wiki/Escape_sequences_in_C

הגדרת האסימונים

שם האסימון	תיאור	ערכים אפשריים	דוגמאות (לקסמות המתאימות לאסימון זה)	אנטי-דוגמאות (לקסמות שאינן מתאימות לאסימון זה)
VOID	המילה השמורה void	void	void	
INT	המילה השמורה לטיפוס מסוג Integer	int	int	
BYTE	המילה השמורה לטיפוס מסוג Byte	byte	byte	

	b כאשר בפועל נשתמש בה בצמוד לליטרל. לדוגמא: 18b	b	המילה השמורה לייצוג ליטרל מסוג Byte	B
	bool	bool	המילה השמורה לטיפוס מסוג Boolean	BOOL
And	and	and	המילה השמורה לאופרטור מסוג and (בשפת C: &&)	AND
Or	or	or	המילה השמורה לאופרטור מסוג or, (בשפת C: ||)	OR
Not	not	not	המילה השמורה לאופרטור מסוג not, (בשפת C: !)	NOT
True 1	true	true	המילה השמורה לליטרל "אמת"	TRUE
False 0	false	false	המילה השמורה לליטרל "שקר"	FALSE
Return	return	return	המילה השמורה לחזרה מפונקציה	RETURN
If IF	if	if	המילה השמורה ל if עבור מבנה הבקרה של תנאי	IF

Else
 	
ELSE
	else	else	המילה השמורה ל else עבור מבנה הבקרה של תנאי	ELSE

While WHILE	while	while	המילה השמורה עבור מבנה הבקרה של לולאת while	WHILE
Break BREAK	break	break	המילה השמורה עבור עצירה ויציאת מלולאה	BREAK
Continue CONTINUE	continue	continue	המילה השמורה עבור המשך ריצת הלולאה	CONTINUE
	;	;	נקודה פסיק	SC

COMMA	פסיק	,	,	
LPAREN	סוגר שמאלי	(	(	
RPAREN	סוגר ימני	)	)	
LBRACE	סוגר מסולסל שמאלי	{	{	
RBRACE	סוגר מסולסל ימני	}	}	
ASSIGN	אופרטור השמה	=	=	==
RELOP	אופרטור רלציוני	== != <= >=	== != <= >=	
BINOP	אופרטור בינארי	+ - * /	+ - * /	
COMMENT	הערת שורה	מתחילה ב // שמופיע מחוץ למחרוזת, ומסתיימת בירידת שורה: LF, CR או CRLF, ביניהם יכול לבוא כל תו.	// my comment	
ID	מזהה (Identifier)	צריך לעמוד בכללים הבאים: - יכול להכיל אותיות אנגליות קטנות וגדולות ומספרים בלבד. - על המזהה להתחיל עם אות אנגלית (קטנה או גדולה). - על המזהה להכיל תו אחד לפחות.	x max OO7	12AB 42 big_x
NUM	מספר שלם	צריך לעמוד בכללים הבאים: - אפסים מובילים אסורים (ראה דוגמא אסורה). - על המספר להכיל תו אחד לפחות.	0 102	050 5.6
STRING	מחרוזת	אוסף תווים בתוך מרכאות כפולות. הערות: 1. אורך המחרוזת יכול להיות בגודל אפס או יותר. 2. ניתן לכלול כל תו ASCII הניתן להדפסה (ניתן לקרוא על כך כאן) פרט לתווים הבאים: a. לוכסן אחורי: \ b. מרכאות כפולות: " c. תו LF: \n (כאשר הוא מגיע כתו בודד) d. תו CR: \r (כאשר הוא מגיע כתו בודד) אלא אם כן הם מגיעים כחלק מ escape sequence תקין. 3. רשימת escape sequence תקינים: a. \\	"simple" "also 'simple'" "escape new lines\n" "hex \x10" "hex2 \xa2" "hex2 \xA2" "hello,\thow\tare\tyou"	'unmatching" "unclosed "multi-lined string" "incep- " -tion" "bad escape \" here"

		b. \" c. \n d. \r e. \t f. \0 g. \xdd כאשר dd מייצג ספרה הקסדצימלית אופן הטיפול ב escape sequence יוסבר בהמשך, בחלק של הדפסת האסימונים. שימו לב: כל רצף בריחה שאינו ברשימה הנ"ל אינו מהווה קלט חוקי. ניתן להניח שהאורך של מחרוזת בלי המרכאות לא עולה על 1024 תווים.	"	

חלק א

בחלק זה עליכם לכתוב תוכנית שתממש מנתח ותכתב בקובץ בשם part_a.cpp.

בתוכנית זו תשתמשו בפונקציה yylex() שנוצרת ע"י flex ועליה לעמוד בדרישות הבאות:

המנתח יתעלם מכל הרווחים הלבנים, חוץ מבתוך מחרוזות.

ניתן להניח שכל הערכים המספריים בתרגיל ניתנים לאחסון על ידי הטיפוס int.

כאשר המנתח מזהה אסימון, יש לפלוט שורה בפורמט הבא (יש לדאוג לרווח יחיד בין כל רכיב שורה ולירידת שורה ע"י LF (\n) בלבד לאחר הרכיב האחרון):

<line number> <token name> <value>

כאשר:

- line number: מספר השורה בה האסימון **מסתיים**.
- token: שם האסימון שזוהה (לפי השמות בחלק "הגדרת אסימונים" למעלה).
- value: ערך האסימון שזוהה, כלומר הלקסמה, פרט למקרה של הערות ומחרוזות, כמוסבר להלן.

הדפסת הלקסמה של מחרוזות:

מחרוזות יודפסו ללא המרכאות הכפולות המקיפות אותן.

נטפל ברצפי הבריחה באופן הבא:

- \n,\r,\t מוחלפים בסוג המתאים של רווח לבן (טאב, CR, LF).
- \\ מוחלפת בלוכסן אחורי יחיד (\).
- \" מוחלפת במרכאות כפולות (").
- רצף בריחה של תו ascii (\xdd): יודפס התו בעל ערך ה-ascii אשר מייצג את הרצף ההקסדצימלי. כך למשל, עבור הרצף \x41 יודפס התו A.
- אם הרצף מהווה ייצוג הקסדסימלי של תו בטווח 0x00-0xFF יש להדפיס את התו המתאים במקום רצף הבריחה. אחרת, יש להדפיס שגיאה (ראה סעיף טיפול בשגיאות).

דוגמה – המחרוזת הבאה:

"Hello **\x57**orld!\r\nThis\tis\t**\x63**oo**\x6C**, as always."

תודפס בפורמט הנדרש באופן הבא:

1 STRING Hello **W**orld!
This is **c**oo**l**, as always.

הדפסת הלקסמה של הערות:

במקום תוכן הערה, יש להדפיס שני לוכסנים קדמיים - //

<u>דוגמה</u>

עבור הקלט:

byte x = 15b;

print("Hello\nyou!");

פלט המנתח יהיה:

1 BYTE byte

1 ID x

1 ASSIGN =

1 NUM 15

1 B b

1 SC ;

2 ID print

2 LPAREN (

2 STRING Hello

you!

2 RPAREN)

2 SC ;

טיפול בשגיאות

הערה: אחרי הדפסת ההודעה המתאימה לשגיאה הראשונה בה נתקלתם, יש לסיים את התכנית (היעזרו בפקודה exit(0)).

1. כאשר המנתח נתקל בתו לא חוקי יש להדפיס:

```
Error <char>\n
```

כך שעבור הקלט הבא:

```
@
```

הודעת השגיאה תהיה:

```
Error @\n
```

(`\n` מסמל תו ירידת שורה)

2. כאשר שורה מסתיימת באמצע מחרוזת, יש להדפיס:

```
Error unclosed string\n
```

3. כאשר מחרוזת מכילה רצף escaping שלא מופיע בהגדרת התרגיל, יש להדפיס:

```
Error undefined escape <sequence>\n
```

כך שעבור מחרוזת המכילה את הרצף **`\q`** , הודעת השגיאה תהיה:

```
Error undefined escape sequence q\n
```

עבור מקרה בו הרצף **`\x`** מלווה בתווים שאינם מייצגים ערך הקסדצימלי או שהמחרוזת נגמרת לפני שניתן לקרוא 2 תווים לאחר ה- **`x`** (למשל עבור המחרוזת "hey \xF"), הודעת השגיאה תכיל את ה escape sequence המלא. לדוגמא עבור מחרוזת המכילה את הרצף **`\xFT`** , הודעת השגיאה תהיה:

```
Error undefined escape sequence xFT\n
```

עבור מקרה בו התו האחרון במחרוזת הוא **** יש להדפיס:

```
Error unclosed string\n
```

חלק ב

הקדמה

בניגוד לייצוג אליו אנחנו רגילים: Infix, ייצוג prefix של ביטוי חשבוני בנוי תחילה מהאופטור ולאחר מכן משני האופרנדים. למשל

+ 5 2 = 10

צורה זו למעשה מייתרת את הצורך לסמן את סדר הפעולות בביטוי ע"י סוגריים. למשל הביטוי הבא בצורת Infix:

7 * (5 + 3)

שקול לביטוי הבא בצורת prefix:

* 7 + 5 3

בנוסף, נוכל לכתוב ביטויים מורכבים יותר, כגון

+/ 100 * 2 25 5 = 7 // equivalent to: (100 / (2*25)) + 5

דוגמא לקריאת ביטוי

כדי להבין כיצד יש לקרוא ביטוי בצורת prefix נפעל כך: נניח כי נתון לנו הביטוי הבא:

+/ 100 * 2 25 5

נסתכל על הביטוי מימן לשמאל, ונסתכל על האופרטור הראשון שנתקל בו. במקרה הזה, זהו אופרטור הכפל (*). מכיוון שהאופרטור פועל על שני אופרנדים שמימנו, נוכל להסתכל על הביטוי בצורה הבאה:

+/ 100 (* 2 25) 5

כעת, נמשיך לאופרטור הבא – חילוק (/), ושוב נסתכל על שני האופרנדים שמימנו. נשים לב שמה שסימנו בסוגריים מהווה אופרנד, לכן נקבל:

+[/ 100 (* 2 25)]5

בדרך זו ניתן לקרוא כל ביטוי מורכב.

כעת, חשבו כיצד ניתן לממש אלגוריתם שבהינתן ביטוי, יתן לנו את ערכו.

בתרגיל זה נעבוד אך ורק עם מספרים שלמים ועם ארבעת הפעולות הבינאריות הבסיסיות (/,*,-,+). שימו לב כי פעולת החילוק תהיה בשלמים. כמו כן, שימו לב כי עבור פעולות החיסור והחילוק, סדר האופרנדים מוגדר כדלקמן:

/ a b ⇔ a / b

- a b ⇔ a - b

קלט ופלט

בחלק זה עליכם לכתוב תוכנית שתממש מחשבון prefix ותכתב בקובץ בשם part_b.cpp.

גם בתוכנית זו תשתמשו בפונקציה ()yylex שנוצרת ע"י flex ועליה לעמוד בדרישות הבאות:

התוכנית תקרא קלט מהמשתמש כפי שביצעתם בסעיף א'.

לאחר קריאת הקלט, התוכנית תדפיס את תוצאת החישוב עם ירידת שורה (\n).

- ניתן להניח כי קבצי הקלט יכילו שורה בודדת.
- לא יבדקו מקרים בהם ישנה מחרוזת לא סגורה.

טיפול בשגיאות

הערה: אחרי הדפסת ההודעה המתאימה לשגיאה הראשונה בה נתקלתם, יש לסיים את התכנית (היעזרו בפקודה exit(0)).

1. כאשר המנתח נתקל בתו לא חוקי יש להדפיס:

```
Error <char>\n
```

כך שעבור הקלט הבא:

```
@
```

הודעת השגיאה תהיה:

```
Error @\n
```

(\n מסמל תו ירידת שורה)

2. כאשר המנתח נתקל באסימון לא חוקי, כלומר כל אסימון שהוא לא מספר שלם או אופרטור בינארי:

```
Error: <TOKEN>\n
```

כך שעבור הקלט "+ 3 void" , הודעת השגיאה תהיה:

```
Error: VOID\n
```

שימו לב, על ההדפסה להכיל את האסימון הלא חוקי הראשון שבו המנתח נתקל.

3. כאשר המנתח נתקל בביטוי שאינו תקין, למשל עבור הדוגמאות הבאות:

++ 5 2

* 2

יודפס:

```
Error: Bad Expression\n
```

הערות נוספות הנוגעות לשני החלקים

- בתרגיל זה תדרשו לכתוב קובץ lex. יחיד שישרת את שני חלקי התרגיל. לכן, שימרו עליו פשוט, וממשו את הלוגיקה הרצויה בקבצי ה cpp.
- באופן דיפולטי, הפונקציה yylex() מחזירה טיפוס int, וחוזרת למשתמש כאשר קיימת פקודת return ב action של האסימון. (ראו שקף 23, בתרגול על המנתח הלקסיקלי)
- לתרגיל מצורף קובץ בשם tokens.hpp המכיל משתנה enum הכולל בתוכו את כל האסימונים.
ביצוע Include לקובץ זה הן בקובץ ה lex. והן בקבצי ה cpp. מאפשר "תקשורת" בין המנתח ש flex יוצר לבין התוכנית שתכתבו. כלומר, התוכנית שתכתבו תדע להבין אילו אסימונים המנתח מחזיר.
לדוגמא, נניח כי יש לנו אסימון בשם FOR, לכן נוכל לכתוב בקובץ ה lex. ב- rule sections:

```
for        return FOR;
```

ואילו בקובץ ה cpp.

```
If (yylex() == FOR) {...}
```

- בנוסף, קובץ ה tokens.hpp מכיל הגדרות שיאפשרו לכם להשתמש בפונקציה yylex() ובמשתנים yylineno, yytext, yyleng
- לתרגילים מצורפים קבצי טמפלייט part_a.cpp, part_b.cpp, המכילים את לולאת הקריאה ל yylex().
העזרו בהם.
- מומלץ להיוועץ ב-manual של flex לצורך ביצוע התרגיל. קל יותר לבצע אותו על ידי שימוש ביכולות מתקדמות של flex שלא נלמדו בתרגולים, כגון **start conditions**, regex patterns מתקדמים, ו- debug mode.
- **טיפ:** השתמשו במבני הנתונים הזמינים בשפת C++ (STL) כגון vector, stack.
- **טיפ:** תוכלו להשתמש באתר http://regexp.com/ שעוזר בהבנה ובבנייה של תבניות regex מורכבות

הערות נוספות על תווים בקובץ

ניתן להניח כי קבצי הדוגמאות הם קבצי ascii בלבד (כלומר: אינם UTF-8 או UTF-16). בהכינכם קבצי בדיקה, וודאו כי אתם מכוונים את ה-Encoding של הקובץ ל-ASCII או ANSI, או מבצעים save as כ-ASCII.
לנוחותכם, וכדי למנוע בעיות בהעתקה בין קבצים, להלן מפתח של התווים המוזכרים בתרגיל וערכי ה-ASCII שלהם:

שם	סימן	ערך ASCII (hex)
סוגר מרובע שמאלי	[	5B
סוגר מרובע ימני	]	5D
סוגר מסולסל שמאלי	{	7B
סוגר מסולסל ימני	}	7D
נקודתיים	:	3A
שווה	=	3D
סימן קריאה	!	21
לוכסן אחורי	\	5C
סולמית	#	23
נקודה פסיק	;	3B
מינוס / מקף	-	2D
פלוס	+	2B

פסיק	,	2C
קו תחתון	_	5F
נקודה	.	2E
גרש	'	27
מרכאות כפולות	"	22
Carriage return	CR	0D
Line feed	LF	0A
רווח		20
טאב		09
שטרודל	@	40
סוגר משולש ימני	>	3E
טילדה	~	7E
כוכבית	*	2A
לוכסן (סלש)	/	2F

קבצי הטסט זמינים בקובץ zip ומומלץ תמיד להוריד ולהעביר אותם כ-zip על מנת למנוע שינוי אוטומטי של ירידות השורה על ידי תוכנות להעברת קבצים.

הוראות הגשה

עליכם להגיש קובץ zip המכיל את **כל** הקבצים שבהם השתמשתם (כולל tokens.hpp אם החלטתם להשתמש בו) ובפרט את הקבצים הבאים (הקפידו על שמות הקבצים):

scanner.lex

part_a.cpp

part_b.cpp

דרישות נוספות:

על המנתח להיבנות על השרת csComp בעזרת הפקודות הבאות:

```
flex scanner.lex
g++ -std=c++11 lex.yy.c part_a.cpp -o part_a.out

g++ -std=c++11 lex.yy.c part_b.cpp -o part_b.out
```

מנתח שלא יבנה בהצלחה בעזרת הפקודות הללו **יקבל 0 אוטומטית**.

בתרגיל זה (כמו בתרגילים אחרים בקורס) ייבדקו העתקות. אנא כתבו את הקוד שלכם בעצמכם.

<u>בדיקת המנתח</u>

באתר הקורס מופיע קובץ zip המכיל קבצי בדיקה לדוגמה. קבצי ה- ta מתאימים לבדיקת החלק הראשון ואילו קבצי ה- tb מתאימים לבדיקת החלק השני.

ניתן ואף רצוי לבדוק את עצמכם באופן הבא:

בנו את המנתח על ידי הפקודות לעיל על השרת csComp. העבירו את קובץ ה-zip של הקבצים לדוגמא לשרת ובצעו unzip. יש להריץ:

```
./part_a.out < ta1.in >& ta1.res
diff ta1.res ta1.out

./part_b.out < tb1.in >& tb1.res
diff tb1.res tb1.out
```

ולבדוק שמתקבל diff ריק. שימו לב כי במידה והמנתח שלכם לא עובר את כל קבצי הבדיקה שסופקו מראש, לא תתאפשר הגשה חוזרת של התרגיל.

שימו לב כי באתר מופיע script לבדיקה עצמית לפני ההגשה בשם selfcheck. תוכלו להשתמש בו על מנת לוודא כי ההגשה שלכם תקינה.

בהצלחה!