תורת הקומפילציה

תרגיל בית 1 – בנית מנתח לקסיקלי

מתרגל אחראי: נדב רובינשטיין- nadavru@campus.technion.ac.il

ההגשה בזוגות בלבד!

עבור כל שאלה על התרגיל, יש לעין ראשית **בפיאצה** ובמידה שלא פורסמה אותה השאלה, ניתן להוסיף אותה ולקבל מענה, אין לשלוח מיילים בנושא התרגיל בית כדי שנוכל לענות על השאלות שלכם ביעילות.

תיקונים לתרגיל יסומנו בצהוב, חובתכם להתעדכן בהם באמצעות קובץ התרגיל.

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, מבני בקרה ועוד.
- במנתח הלקסיקלי שתממשו נשתמש כדי ליצור את שתי התוכניות הבאות:
  - חלק א': תכנית הקוראת קלט מהמשתמש ומדפיסה מידע על האסימונים שהיא מצאה.
  - חלק ב': תכנית הממשת בדיקה ויישור של ביטוי המכיל סוגריים.
- התרגיל ייבדק אוטומטית. **הקפידו אחר ההוראות במדויק**. הבדיקה תתבצע על csComp אליו ניתן להתחבר דרך SSH לשרת csl3 בכתובת csl3.cs.technion.ac.il באמצעות שם המשתמש והסיסמא הטכנייונים שלכם. לאחר מכן תוכלו לבצע את הפקודה: ssh <user>@csComp שתחבר אותכם לשרת csComp.
- יש להשתמש ב- flex בלבד (ולא ב- lex) כפי שלמדנו בשיעורים.

הגדרות מושגים כלליים

- רווח לבן – אחד מבין: רווח (ספייס), טאב, CR (התו \r), LF (התו \n).
- תווים ניתנים להדפסה – התווים שערך ה- asci 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 אחד.
  - 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 אסימונים

שם האסימון	תיאור	ערכים אפשריים	דוגמאות	אנטי-דוגמאות
VOID	המילה השמורה void	void	void	
INT	המילה השמורה לטיפוס מסוג Integer	int	int	
BYTE	המילה השמורה לטיפוס מסוג Byte	byte	byte	
B	המילה השמורה לייצוג ליטרל מסוג Byte	b	b כאשר בפועל נשתמש בה בצמוד לליטרל. לדוגמא: 18b	
BOOL	המילה השמורה לטיפוס מסוג Boolean	bool	bool	
AND	המילה השמורה לאופרטור מסוג and (בשפת C: &&)	and	and	And
OR	המילה השמורה לאופרטור מסוג or (בשפת C: ||)	or	or	Or
NOT	המילה השמורה לאופרטור מסוג not (בשפת C: !)	not	not	Not
TRUE	המילה השמורה לליטרל "אמת"	true	true	True 1
FALSE	המילה השמורה לליטרל "שקר"	false	false	False 0
RETURN	המילה השמורה לחזרה מפונקציה	return	return	Return
IF	המילה השמורה ל- if עבור מבנה הבקרה של תנאי	if	if	If IF
ELSE	המילה השמורה ל- else עבור מבנה הבקרה של תנאי	else	else	Else ELSE
WHILE	המילה השמורה עבור מבנה	while	while	While

	הבקרה של לולאת while			
BREAK	המילה השמורה עבור עצירה ויציאה מלולאה	break	break	Break BREAK
CONTINUE	המילה השמורה עבור המשך ריצת הלולאה	continue	continue	Continue CONTINUE
SWITCH	מילה שמורה לפתיחת בלוק switch-case	switch	switch (expr) {…}	swtch Switch
CASE	מילה שמורה לתנאי בבלוק switch-case	case	case val1: …	Case
DEFAULT	מילה שמורה לברירת המחדל בבלוק switch-case	default	default: …	Default
COLON	נקודתיים	:	:	
SC	נקודה פסיק	;	;	
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, ואחרי שני הלוכסנים יכול לבוא כל תו מלבד ירידת שורה: LF, CR או CRLF	// my comment	//a comment Not anymore
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 <u>פרט</u> 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. \\ 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 <u>אינו מהווה קלט חוקי</u>. ניתן להניח שהאורך של מחרוזת בלי המרכאות לא עולה על 1024 תווים.	"simple" "also 'simple'" "new lines\n" "beast-mode" "hex \x10" "hex2 \x02" "hex2 \x3A" "hi\thow\tare\tyou"	'unmatching" "unclosed "multi Lined" "not-"-good"

<u>חלק א'</u>

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

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

המנתח יתעלם מכל הרווחים הלבנים, חוץ מבתוך מחרוזת.

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 <u>הערות ומחרוזות</u>, 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 (")
- \0 מוחלפת בתו Null-character שמיוצג ע"י \0 המציין את סוף המחרוזת.
  לכן, "abc\0abc" כקלט יודפס בפלט: 1 STRING abc.
  שימו לב! עליכם להמשיך לסרוק את המחרוזת לאחר שני התווים "\0" (ואם ישנה שגיאה לאחר מכן יש להתייחס אליה). המחרוזת תיחתך בהדפסה בלבד.
- 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 הקסדצימלי (not case-sensitive) של תו בטווח 0x00-0x7F יש להדפיס את התו המתאים במקום רצף הבריחה. אחרת, יש להדפיס שגיאה (ראה סעיף טיפול בשגיאות).
  - דוגמה – המחרוזת הבאה:

```
"Hello \x57orld!\r\nThis\tis\t\x63oo\x6C, as always."
```

תודפס בפורמט הנדרש באופן הבא:

```
1 STRING Hello World!
This  is   cool, as always.
```

הדפסת הלקסמה של הערות:

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

<u>קלט פלט לדוגמא</u>

עבור הקלט:

```
byte x = 15b;

print("Hello\nyou!");
```

פלט המתנח 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 ;
```

<u>טיפול בשגיאות</u>

הערה: אחרי הדפסת ההודעה המתאימה לשגיאה <u>הראשונה</u> בה נתקלתם, יש לסים 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 <u>שורה</u> מסתיימת באמצע מחרוזת (גם אם מכילה רצף בריחה לא חוקי), יש להדפיס:

```
Error unclosed string\n
```

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

```
Error undefined escape sequence <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'

בחלק זה תצטרכו לכתוב תכנית שתבדוק ביטוי המורכב מסוגריים (וייתכן מתווים נוספים), ותדפיס אותו מיושר.

התכנית תיכתב בpart_b.cpp, ותשתמש בפונקציה ()yylex שנוצרה ע"י flex ותבצע:

- קריאת קלט מהמשתמש כמו בסעיף א'.
- הדפסת הסוגריים המיושרים (כל עוד אין שגיאות).

סוגי הסוגריים אליהם תתייחסו הם ()[]{} שהמנתח הלקסיקלי שכתבתם כבר מזהה.

שימו לב! אלו שני סוגים שונים של סוגריים, לכן הם זרים אחד לשני.

כל עוד המנתח לא נתקל בשגיאות, בכל פתיחה\סגירה של סוגר (משני הסוגים), הנמצא בתוך x פתיחות (משני הסוגים) שלא נסגרו עדיין, יש להדפיס את התו עם x הזחות לפניו (tab או \t).

<u>טיפול בשגיאות:</u>

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 שאינו חוקי, כלומר שאינו אחד מארבעת הסוגרים, יש להדפיס:

```
Error: <token>\n
```

3. כאשר המנתח נתקל בסגירת סוגריים לא מתאימים, סגירת סוגריים כשלא ניתן לסגור, קלט שנגמר כשאמור להיות סגירת סוגריים, או כל אפשרות אחרת המציינת ביטוי סוגריים לא תקין, יש להדפיס (ללא הזחות):

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

לאחר כל סוג של שגיאה יש להדפיס את ההודעה המתאימה ולצאת מן התוכנית.

דוגמאות:

1. עבור הקלט: "{}()}{()" יודפס:

```
(
)
{
        (
        )
        {
        }
}
```

2. עבור הקלט "{a)" יודפס:

```
(
Error: ID
```

3. עבור הקלט "{()}" יודפס:

```
(
        {
Error: Bad Expression
```

4. עבור הקלט ")" יודפס:

```
)
Error: Bad Expression
```

הערות נוספות:

1. התעלמו מרווחים לבנים.
2. לא תהיינה בקלט שגיאה שלא מצויינת בטיפול בשגיאות. בפרט, כל string בקלט יהיה תקין.

הערות נוספות הנוגעות לשני החלקים

- בתרגיל זה תדרשו לכתוב קובץ lex. יחיד שישרת את שני חלקי התרגיל. שימרו עליו פשוט, וממשו את הלוגיקה הרצויה בקבצי ה- cpp.
- כברירת מחדל, 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 התכנית שתכתבו. כלומר, התכנית שתכתבו תדע להבין אילו אסימונים המנתח מחזיר.
לדוגמא, נניח כי יש לנו אסימון בשם FOR, לכן נוכל לכתוב בקובץ ה- lex. ב- rules section:

```
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
נקודותיים	:	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.

<u>הוראות הגשה</u>

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

<u>דרישות נוספות</u>

על המנתח להבנות 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 לדוגמא.

קבצי הta מתאימים לבדיקת החלק הראשון וקבצי הtb 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. לדוגמא, עבור הטסטים הראשונים 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!