Μεταγλωττιστές 2023-24

Προσθήκη εντολών if και while στη γραμματική των αριθμητικών εκφράσεων

Στόχος

- Η προσθήκη εντολών if και while στον συντακτικό αναλυτή και AST interpreter
  - Εκτός από τα ASSIGN και PRINT
- `if E Bs` και `while E Bs`
  - E = expression, Bs = block statement
    - ή ένα μοναδικό Stmt
    - ή { Stmt_list }
  - Προς το παρόν θα παραλείψουμε τη δομή else

Η συνθήκη των if και while

- Η μορφή των εντολών if και while περιέχει μια συνθήκη E
  - `if E Bs` και `while E Bs`
  - Εάν/Όσο το E είναι αληθές εκτελείται το Bs
- Σχεδιαστική απόφαση: τι τύπο θα έχει το E;
  - Ξεχωριστές λογικές εκφράσεις;
  - Ενιαίος τύπος εκφράσεων; (λογικές – αριθμητικές)

Η συνθήκη των if και while (2)

- Ξεχωριστές λογικές εκφράσεις
  - Ακριβέστερη αλλά και συνθετότερη γραμματική
  - Ενδεχομένως δεν θα είναι LL(1)
    - Και δεν θα αναλύεται ούτε με custom κώδικα
- Ενιαίος τύπος εκφράσεων
  - Στη γραμματική ένα και μοναδικό Expr παντού
  - Πώς θα αναπαραστήσουμε το true/false στη γλώσσα;
    - Υπάρχει μόνο ο τύπος float

Χειρισμός του αληθούς/ψευδούς

- Αριθμητική αναπαράσταση
  - ίσο με 0 → ψευδές, διάφορο του 0 → αληθές
    - Όπως και σε αρκετές «πραγματικές» γλώσσες
  - Δεν χρειάζεται να προσθέσουμε νέο τύπο (boolean)
  - Ούτε να παρακολουθούμε κατά την εκτέλεση τον τύπο της υπολογιζόμενης έκφρασης
    - Κάθε έκφραση αποκτά τύπο δυναμικά (κατά την εκτέλεση)

Αρχική γραμματική

```
Stmt_list   → Stmt Stmt_list | ε
Stmt        → id = Expr | print Expr
Expr        → Term Term_tail
Term_tail    → Addop Term Term_tail | ε
Term        → Factor Factor_tail
Factor_tail → Multop Factor Factor_tail | ε
Factor      → (Expr) | id | number
Addop       → + | -
Multop      → * | /
```

Προτεινόμενη γραμματική

```
Stmt_list   → Stmt Stmt_list | ε
Stmt        → id = Expr | print Expr
            | if Expr Block_stmt | while Expr Block_stmt
Block_stmt  → Stmt | { Stmt_list }
Expr        → Term Term_tail
Term_tail   → Addop Term Term_tail | ε
Term        → Factor Factor_tail
Factor_tail → Multop Factor Factor_tail | ε
Factor      → ( Expr ) | id | number
Addop       → + | -
Multop      → * | /
```

TODO: Έλεγχος γραμματικής

- Βεβαιωθείτε ότι η γραμματική είναι LL(1)
- Βρείτε τα FIRST και FOLLOW sets
- Μπορείτε να συμβουλευτείτε κάποιο on-line εργαλείο, όπως το:

  http://smlweb.cpsc.ucalgary.ca/start.html

TODO: Τα νέα tokens

- Βρείτε τα νέα tokens (τερματικά σύμβολα) που πρέπει να προστεθούν στον λεκτικό αναλυτή
- Προσθέστε τα στα patterns του tokenizer

TODO: Προσθήκες στις μεθόδους των μη τερματικών (όχι ακόμα δημιουργία AST)

- Προσθέστε νέα μέθοδο για το Block_stmt
  - Κανόνες: `Block_stmt → Stmt | { Stmt_list }`
- Προσθέστε τους νέους κανόνες στο Stmt
  - `Stmt → if Expr Block_stmt`
  - `Stmt → while Expr Block_stmt`

TODO: Αλλαγές στις μεθόδους των μη τερματικών

- Ελέγξτε τις μεθόδους των μη τερματικών συμβόλων που ήδη υπάρχουν
  - Θα χρειαστεί να προσθέσετε tokens στους ελέγχους του next_symbol.token
    - Εάν τα FIRST sets έχουν αλλάξει
  - Και στην περίπτωση του `Stmt_list → ε`
    - Το FOLLOW set έχει επίσης αλλάξει

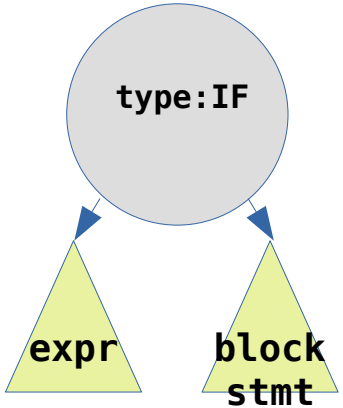
TODO: Έλεγχος λειτουργίας συντακτικού αναλυτή

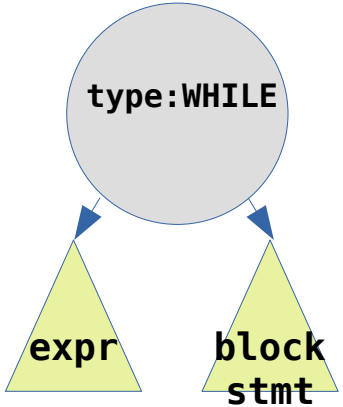
- Βάλτε σε σχόλια τη δημιουργία και εκτέλεση του interpreter
- Βάλτε σε σχόλιο την εκτύπωση του AST
- Ελέγξτε τη λειτουργία του συντακτικού αναλυτή με τη δοκιμαστική είσοδο της επόμενης διαφάνειας
  - Θα πρέπει να μην παράγεται κανένα σφάλμα λεκτικής ή συντακτικής ανάλυσης

Δοκιμαστική είσοδος

```
a = 2 + 7.55*44
print a
if a-7 {
  b = 3*(a-99.01)
  it = 5
  while it {
    print it+b*0.23
    it = it - 1
  }
}
c = 5-3-2
print c
```

TODO: Δημιουργία AST για νέες εντολές if και while





- Μέσα στη μέθοδο Stmt, στους κανόνες του if και του while
  - Δημιουργήστε κόμβους AST σύμφωνα με το σχήμα
  - Το υποδένδρο expr είναι αυτό που επιστρέφει η Expr() του if/while (AST αριθμητικής έκφρασης)
  - Το υποδένδρο block-stmt είναι αυτό που επιστρέφει η Block_stmt() του if/while (AST ακολουθίας εντολών)
- Στο τέλος εκτυπώστε το συνολικό AST και ελέγξτε το αποτέλεσμα

TODO: Προσθήκες στον AST interpreter

- Στη μέθοδο execute_statement προσθέστε τον χειρισμό των κόμβων τύπου IF και WHILE
  - Υπολογίστε την τιμή της έκφρασης E
    - Καλέστε την evaluate_expression για το αριστερό υποδένδρο
    - Εάν/Όσο η επιστρεφόμενη τιμή είναι διάφορη του 0, τότε καλείτε την execute_statement για το δεξιό υποδένδρο (εκτέλεση εντολών που περιέχονται στο block statement BS)
- Αφαιρέστε τα σχόλια από τον κώδικα για τη δημιουργία και εκτέλεση του interpreter και ελέγξτε τα αποτελέσματα που τυπώνονται

Προσθήκη του “else”

```
Stmt        → if Expr Block_stmt
            | if Expr Block_Stmt else Block_Stmt
            | ...
```

- Στόχος είναι να προστεθεί (προαιρετικά) το else στη δομή if
  - Μέχρι τώρα έχουμε υλοποιήσει μόνο τον κανόνα `if Expr Block_stmt`
- Είναι η γραμματική LL(1);
  - Αρχικά θα πρέπει να φύγει ο «κοινός παράγοντας» (κοινό πρόθεμα `if Expr`) από τους δύο κανόνες if

Προσθήκη του “else”

```
Stmt        → if Expr Block_stmt Rest_if
            | ...
Rest_if     → else Block_stmt | ε
```

- Μετά την απαλοιφή του κοινού προθέματος είναι η νέα γραμματική LL(1);
  - Ας βρούμε τα FIRST και FOLLOW sets για το `Rest_if`

FIRST/FOLLOW sets για το Rest_if

Σύνολα FOLLOW	Σύνολα FIRST	Κανόνες	
			→ Stmt #
#,else ...	if ...	Stmt	→ if Expr Block_Stmt Rest_if | ...
#,else ...	else ε	Rest_if	→ else Block_stmt | ε

- Υπάρχει σύγκρουση μεταξύ του FIRST και του FOLLOW set του μη τερματικού `Rest_if` (FIRST/FOLLOW conflict)
  - Η γραμματική δεν είναι LL(1)
- Τι σημαίνει αυτό για την μέθοδο υλοποίησης που ακολουθούμε;

Μια απόπειρα υλοποίησης...

FIRST/FOLLOW conflict

Η γραμματική δεν είναι LL(1)!

```
def Rest_if(self):
    if self.next_token =='else':
        # Rest_if → else Block_stmt
        self.match('else')
        self.Block_stmt()
    elif self.next_token in ('else',None):
        # Rest_if → ε
        return
    else:
        raise ParseError
```

Πώς θα διαλέξουμε κανόνα όταν εμφανιστεί ένα else;

```
if expr if expr stmt1 else stmt2
```

Σε ποιο if ανήκει το **else**;

Η ιδέα

```
Stmt -> if Expr Block_stmt (else Block_stmt)?
      | ...
```

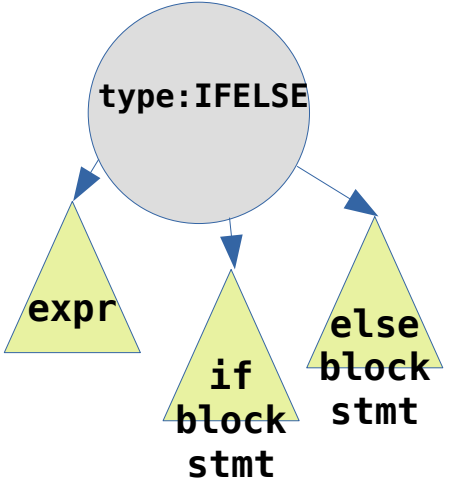
- **Εάν** μετά το if expr Block_stmt ακολουθεί else, τότε ταιριάζουμε κατευθείαν το else Block_stmt
  - Συνεπώς το else θα συνδυαστεί με το τελευταίο if που έχουμε ήδη επεξεργαστεί, δηλ. το **κοντινότερο if**
- Επεκταμένη σύνταξη γραμματικής, **(…)?** = «προαιρετικά»

Ο νέος κώδικας του if στο Stmt()

```
def Stmt(self):
    ...
    if self.next_token=='if':
        # Stmt -> if Expr Block_stmt (else Block_stmt)?
        self.match('if')
        self.Expr()
        self.Block_stmt()
        # test if optional part (else Block_stmt) follows
        if self.next_token=='else':
            self.match('else')
            self.Block_stmt()
    ...
```

TODO

- Προσθέστε ένα νέο token για το “else”
- Τροποποιήστε κατάλληλα τη μέθοδο Stmt()
  - Προσθέστε τον χειρισμό του else
  - Ο κόμβος του AST θα είναι τύπου IFELSE
- Προσθέστε τον κώδικα για την εκτέλεση του κόμβου IFELSE στον interpreter



Νέα δοκιμαστική είσοδος

```
a = 2 + 7.55*44
print a
if a-7 {
  b = 3*(a-99.01)
  it = 5
  while it {
    print it+b*0.23
    it = it - 1
  }
}
c = 5-3-2
if it+1 if c print c else print c+28
```