ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΠΛΗ 402
Θεωρία Υπολογισμού

Εργασία Προγραμματισμού
Λεκτική και Συντακτική Ανάλυση της Γλώσσας Προγραμματισμού FL

Διδάσκων
Βασίλης Σαμολαδάς

Βοηθός
Γιώργος Ανέστης

Παράδοση: 20 Μαΐου 2018

Εαρινό Εξάμηνο 2018

Τελευταία ενημέρωση: 2018-03-20

1 Εισαγωγή

Η εργασία προγραμματισμού του μαθήματος **«ΠΛΗ 402 – Θεωρία Υπολογισμού»** έχει ως στόχο τη βαθύτερη κατανόηση της χρήσης και εφαρμογής θεωρητικών εργαλείων, όπως οι κανονικές εκφράσεις και οι γραμματικές χωρίς συμφραζόμενα, στο πρόβλημα της μεταγλώττισης (compilation) γλωσσών προγραμματισμού. Συγκεκριμένα, η εργασία αφορά στη σχεδίαση και υλοποίηση των αρχικών σταδίων ενός μεταγλωττιστή (compiler) για τη φανταστική γλώσσα προγραμματισμού **FL (F**ictional **L**anguage**)**, η οποία περιγράφεται αναλυτικά παρακάτω.

Πιο συγκεκριμένα, θα δημιουργηθεί ένας **source-to-source compiler** (trans-compiler ή transpiler), δηλαδή ένας τύπος μεταγλωττιστή ο οποίος παίρνει ως είσοδο τον πηγαίο κώδικα ένας προγράμματος σε μια γλώσσα προγραμματισμού και παράγει τον ισοδύναμο πηγαίο κώδικα σε μια άλλη γλώσσα προγραμματισμού. Στην περίπτωσή μας ο πηγαίος κώδικας εισόδου θα είναι γραμμένος στη φανταστική γλώσσα προγραμματισμού **FL** και ο παραγόμενος κώδικας θα είναι στη γλώσσα προγραμματισμού **C**.

Για την υλοποίηση της εργασίας θα χρησιμοποιήσετε τα εργαλεία **flex** και **bison** τα οποία είναι διαθέσιμα ως ελεύθερο λογισμικό και τη γλώσσα **C**.

Η εργασία περιλαμβάνει δύο τμήματα:

- Υλοποίηση **λεκτικού αναλυτή** για τη γλώσσα **FL** με χρήση **flex**
- Υλοποίηση **συντακτικού αναλυτή** για τη γλώσσα **FL** με χρήση **bison**
  - Μετατροπή του κώδικα της **FL** σε κώδικα **C** με χρήση ενεργειών του **bison**

Παρατηρήσεις

- Η εργασία θα εκπονηθεί **σε ομάδα των δύο ατόμων**.
- Για την εκπόνηση της εργασίας μπορούν να χρησιμοποιηθούν υπολογιστές του Μηχανογραφικού Κέντρου και προσωπικοί υπολογιστές. Τα εργαλεία flex και bison είναι διαθέσιμα σε οποιαδήποτε διανομή Linux.
- Η παράδοση της εργασίας θα γίνει **ηλεκτρονικά** μέσα από την ιστοσελίδα του μαθήματος στο courses. Το παραδοτέο αρχείο τύπου archive (.zip ή .tar) θα πρέπει να εμπεριέχει όλα τα απαραίτητα αρχεία σύμφωνα με τις προδιαγραφές της εργασίας.
- Η εργασία πρέπει να παραδοθεί εντός της προθεσμίας. Εκπρόθεσμες εργασίες δεν γίνονται δεκτές.
- Η αξιολόγηση της εργασίας θα περιλαμβάνει **εξέταση καλής λειτουργίας** του παραδοτέου προγράμματος σύμφωνα με τις προδιαγραφές καθώς και **προφορική εξέταση**. Η εξέταση θα γίνει στο Πολυτεχνείο, σε ημέρες και ώρες που θα ανακοινωθούν.

2 Η γλώσσα προγραμματισμού FL

Η περιγραφή της γλώσσας **FL** παρακάτω πιθανότατα περιέχει και στοιχεία τα οποία δεν εντάσσονται στη λεκτική ή συντακτική ανάλυση. Είναι ευθύνη σας να αναγνωρίσετε αυτά τα στοιχεία και να τα αγνοήσετε κατά την ανάπτυξη του αναλυτή σας.

Κάθε πρόγραμμα σε γλώσσα **FL** είναι ένα σύνολο από *λεκτικές μονάδες,* οι οποίες είναι διατεταγμένες βάσει *συντακτικών κανόνων*, όπως περιγράφονται παρακάτω.

2.1 Λεκτικές μονάδες

Οι λεκτικές μονάδες της γλώσσας **FL** χωρίζονται στις παρακάτω κατηγορίες:

- Τις ***λέξεις κλειδιά*** *(keywords)*, οι οποίες είναι οι παρακάτω:

and	**array**	**boolean**	**char**	**begin**
div	**do**	**else**	**for**	**end**
function	**goto**	**if**	**integer**	**var**
mod	**not**	**of**	**or**	**while**
procedure	**program**	**real**	**repeat**	**to**
result	**return**	**then**	**until**	**downto**

Οι λέξεις κλειδιά είναι case-sensitive, δηλαδή. δεν μπορείτε να χρησιμοποιήσετε κεφαλαία γράμματα γι αυτές.

- Τα ***αναγνωριστικά*** *(identifiers)* που χρησιμοποιούνται για ονόματα μεταβλητών και συναρτήσεων και αποτελούνται από ένα πεζό ή κεφαλαίο γράμμα του λατινικού αλφαβήτου, ακολουθούμενο από μια σειρά μηδέν ή περισσοτέρων πεζών ή κεφαλαίων γραμμάτων, ψηφίων του δεκαδικού συστήματος ή χαρακτήρων υπογράμμισης (underscore). Τα αναγνωριστικά δεν πρέπει να συμπίπτουν με τις λέξεις κλειδιά που αναφέρθηκαν παραπάνω.

  Παραδείγματα: `x` `y1` `angle` `myValue` `Distance_02`

- Οι ***ακέραιες (θετικές) σταθερές*** *(integer positive constants)*, που αποτελούνται από ένα ή περισσότερα ψηφία του δεκαδικού συστήματος χωρίς περιττά μηδενικά στην αρχή.

  Παραδείγματα: `0` `42` `1284200` `3` `100001`

- Οι ***πραγματικές (θετικές) σταθερές*** *(real positive constants)*, που αποτελούνται από ένα ακέραιο μέρος, ένα κλασματικό μέρος και ένα προαιρετικό εκθετικό μέρος. Το ακέραιο μέρος αποτελείται από ένα ή περισσότερα ψηφία (του δεκαδικού συστήματος) χωρίς περιττά μηδενικά στην αρχή. Το κλασματικό μέρος αποτελείται από το χαρακτήρα της υποδιαστολής (`.`) ακολουθούμενο από ένα ή περισσότερα ψηφία του δεκαδικού συστήματος. Τέλος, το εκθετικό μέρος αποτελείται από το πεζό ή κεφαλαίο γράμμα E, ένα προαιρετικό πρόσημο + ή - και ένα ή περισσότερα ψηφία του δεκαδικού συστήματος χωρίς περιττά μηδενικά στην αρχή.

  Παραδείγματα: `42.0` `4.2e1` `0.420E+2` `42000.0e-3`

- Οι ***λογικές σταθερές*** *(boolean constants)*, που είναι οι λέξεις-τιμές **`true`** και **`false`**.

- Οι ***σταθερές συμβολοσειρές*** *(constant strings)*, που αποτελούνται από μια ακολουθία κοινών χαρακτήρων ή χαρακτήρων διαφυγής (escape characters) μέσα σε μονά ή διπλά εισαγωγικά. Κοινοί χαρακτήρες είναι όλοι οι εκτυπώσιμοι χαρακτήρες πλην των απλών και διπλών εισαγωγικών και του χαρακτήρα `\` (backslash). Οι χαρακτήρες διαφυγής ξεκινούν με το `\` (backslash) και περιγράφονται στον παρακάτω πίνακα.

Χαρακτήρας Διαφυγής	Περιγραφή
\n	χαρακτήρας αλλαγής γραμμής (line feed)
\t	χαρακτήρας στηλοθέτησης (tab)
\r	χαρακτήρας επιστροφής στην αρχή της γραμμής
****	χαρακτήρας \ (backslash)
\'	χαρακτήρας ' (απλό εισαγωγικό)
\"	χαρακτήρας " (διπλό εισαγωγικό)

Μια σταθερή συμβολοσειρά δεν μπορεί να εκτείνεται σε περισσότερες από μία γραμμές του αρχείου εισόδου. Ακολουθούν παραδείγματα έγκυρων συμβολοσειρών:

```
'a'   "M"   "\n"   '\''   "abc"   'Route 66'
"Hello world!\n"   "Item:\t\"Laser Printer\"\nPrice:\t$142\n"
```

- Τους ***τελεστές*** *(operators)*, οι οποίοι είναι οι παρακάτω:

  αριθμητικοί τελεστές: **+ - * / div mod**

  σχεσιακοί τελεστές: = <> < <= > >=

  λογικοί τελεστές: **and or not && || !**

  τελεστές προσήμου: **+ -**

  τελεστής ανάθεσης: **:=**

  τελεστές casting: **(<data-type>)**

Τελεστής	Περιγραφή
div	ακέραιο μέρος πηλίκου διαίρεσης
mod	υπόλοιπο διαίρεσης
=	ίσο με (=)
>	μεγαλύτερο από (>)
<	μικρότερο από (<)
<>	διάφορο από (≠)
>=	μεγαλύτερο ή ίσο με (≥)
<=	μικρότερο ή ίσο με (≤)
and &&	λογική σύζευξη
or ||	λογική διάζευξη
not !	λογική άρνηση
(<data-type>)	casting έκφρασης σε κάποιο βασικό τύπο

- Τους ***διαχωριστές*** *(delimiters)*, οι οποίοι είναι οι παρακάτω:

  begin end ; ( ) , [ ] := : .

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα **FL** μπορεί επίσης να περιέχει και στοιχεία που αγνοούνται (δηλαδή αναγνωρίζονται, αλλά δεν γίνεται κάποια ανάλυση):

- ***Κενούς χαρακτήρες*** *(white space)*, δηλαδή ακολουθίες αποτελούμενες από κενά διαστήματα (space), χαρακτήρες στηλοθέτησης (tab), χαρακτήρες αλλαγής γραμμής (line feed) ή χαρακτήρες επιστροφής στην αρχή της γραμμής (carriage return).
- ***Σχόλια*** *(comments)*, τα οποία ξεκινούν με την ακολουθία χαρακτήρων **(*** και τερματίζονται με την πρώτη μετέπειτα εμφάνιση της ακολουθίας χαρακτήρων ***)**. Τα σχόλια δεν επιτρέπεται να είναι φωλιασμένα. Στο εσωτερικό τους επιτρέπεται η εμφάνιση οποιουδήποτε χαρακτήρα.
- ***Σχόλια γραμμής*** *(line comments),* το οποία ξεκινούν με την ακολουθία χαρακτήρων // και εκτείνονται ως το τέλος της τρέχουσας γραμμής.

Το αρχείο εισόδου μπορεί να περιέχει προαιρετικά δηλώσεις ορισμού της μορφής

@defmacro <identifier> <string>

οι οποίες καταλαμβάνουν μία ξεχωριστή γραμμή αποκλειστικά η κάθε μία και μπορούν να βρίσκονται οπουδήποτε μέσα στην είσοδο. Για παράδειγμα, μια δήλωση ορισμού θα ήταν η

`@defmacro my2pi (2*3.14159265)`

Τέτοιες δηλώσεις ορισμού δεν μεταφέρονται ως λεκτικές μονάδες στη συντακτική ανάλυση, αλλά πρέπει να διεκπεραιωθούν πλήρως κατά τη λεκτική ανάλυση. Αυτό σημαίνει ότι αν στην πορεία αναγνωρισθεί κάποια λεκτική μονάδα που ταιριάζει με τον identifier κάποιου define που έχει προηγηθεί, τότε αυτή θα πρέπει να αντικατασταθεί από το string που ορίζεται στα δεξιά του **defmacro** για το συγκεκριμένο identifier, να συνεχιστεί η λεκτική ανάλυση μ' αυτό το νέο string και μετά να επανέλθει στο αρχείο εισόδου. Εάν υπάρχουν πολλαπλές δηλώσεις ορισμού με τον ίδιο identifier, τότε σε κάθε σημείο του προγράμματος ισχύει η πιο πρόσφατη δήλωση σε σχέση με το εν λόγω σημείο.

2.2 Συντακτικοί κανόνες

Οι συντακτικοί κανόνες της γλώσσας **FL** ορίζουν την ορθή σύνταξη των λεκτικών μονάδων της.

i) Προγράμματα

Ένα πρόγραμμα **FL** μπορεί να βρίσκεται μέσα σε ένα αρχείο με κατάληξη **`.fl`** και αποτελείται από την κύρια δομική μονάδα η οποία μπορεί να περιέχει εσωτερικά περισσότερες δομικές μονάδες. Η κύρια δομική μονάδα αποτελείται από τα παρακάτω συστατικά τα οποία παρατίθενται μ' αυτή τη σειρά και διαχωρίζονται μεταξύ τους με το διαχωριστικό "**`;`**":

- Επικεφαλίδα κύριας δομικής μονάδας
- Δηλώσεις μεταβλητών (προαιρετικά)
- Ορισμοί υποπρογραμμάτων (προαιρετικά)
- Σύνθετη εντολή κύριας δομικής μονάδας

Η επικεφαλίδα της κύριας δομικής μονάδας είναι της μορφής

program <identifier>

όπου `<identifier>` είναι το όνομα του προγράμματος. Στη συνέχεια, η κύρια δομική μονάδα περιλαμβάνει προαιρετικά δηλώσεις μεταβλητών και ορισμούς υποπρογραμμάτων. Τελευταίο και απαραίτητο συστατικό της κύριας δομικής μονάδας είναι μια σύνθετη εντολή, η οποία καθορίζει τη λειτουργία του προγράμματος και μπορεί να κάνει χρήση των μεταβλητών και των υποπρογραμμάτων που έχουν δηλωθεί στην κύρια δομική μονάδα. Η εκτέλεση του προγράμματος ξεκινάει από τη σύνθετη αυτή εντολή. Η κύρια δομική μονάδα τελειώνει υποχρεωτικά με το διαχωριστή "**`.`**".

Ένα απλό παράδειγμα έγκυρου αρχείου **`.fl`** είναι το παρακάτω:

```
program test;
var x: integer;
```

```
begin
    x := 1+2+3+4+5;
    writeInteger(x)
end.
```

ii) Τύποι δεδομένων

Η γλώσσα **FL** υποστηρίζει τέσσερις βασικούς τύπους δεδομένων (data types):

- `integer` : ακέραιοι αριθμοί
- `boolean` : λογικές τιμές
- `char` : χαρακτήρες
- `real` : πραγματικοί αριθμοί

Η **FL** υποστηρίζει επίσης τους παρακάτω σύνθετους τύπους δεδομένων (πίνακες):

- **`array [n] of`** ***`T`***: μονοδιάστατοι πίνακες αποτελούμενοι από `n` στοιχεία τύπου `t`. Το `n` θα πρέπει να είναι ακέραια σταθερά με θετική τιμή και το `t` έγκυρος βασικός τύπος. Παράδειγμα: `array [10] of integer`
- **`array [n]…[k] of`** ***`T`***: πολυδιάστατοι πίνακες αποτελούμενοι από στοιχεία τύπου `t`. Τα `n`, …, `k` θα πρέπει να είναι ακέραιες σταθερές με θετική τιμή και το `t` έγκυρος βασικός τύπος. Ένα παράδειγμα τρισδιάστατου πίνακα τύπου `char` είναι: `array [10][4][20] of char`
- **`array of`** ***`T`***: πίνακες αποτελούμενοι από άγνωστο αριθμό στοιχείων τύπου `t`, όπου το `t` είναι έγκυρος βασικός τύπος. Παράδειγμα: `array of real`
- **`function (`*`ParamList`*`) : `*`Treturn`**: Τύπος συνάρτησης, με συγκεκριμένη "υπογραφή", δηλαδή, τύπου ορισμάτων και επιστροφής.

Κατά τη χρήση, οι δείκτες των πινάκων πρέπει να είναι ακέραιες σταθερές ή εκφράσεις με θετική τιμή.

Τέλος, μπορούμε να δηλώσουμε ονόματα για συγκεκριμένους τύπους, π.χ.,

```
type
   string = array of char;
   vector = array [3] of real;
   string_array = array [10] of string;
   transformation = function(x : vector): vector;
```

τα οποία μπορούν να χρησιμοποιηθούν στον κώδικα ως συντομογραφίες.

iii) Μεταβλητές

Οι δηλώσεις μεταβλητών ξεκινούν με τη λέξη κλειδί **`var`**. Ακολουθούν ένα ή περισσότερα αναγνωριστικά μεταβλητών χωρισμένα με κόμμα και, τέλος, ο διαχωριστής `:` με έναν βασικό ή σύνθετο τύπο δεδομένων. Πολλαπλές συνεχόμενες δηλώσεις μεταβλητών διαχωρίζονται μεταξύ τους με το διαχωριστικό "**`;`**" και εντάσσονται υποχρεωτικά στο ίδιο `var`. Ακολουθεί ένα παράδειγμα δηλώσεων μεταβλητών:

```
var i: integer;
    x, y: real;
    s: array [80] of char;
```

iv) Υποπρογράμματα

Τα υποπρογράμματα διακρίνονται σε *διαδικασίες* (**procedure**) και *συναρτήσεις* (**function**). Κάθε υποπρόγραμμα είναι μια δομική μονάδα και αποτελείται από τα παρακάτω συστατικά τα οποία παρατίθενται μ' αυτή τη σειρά και διαχωρίζονται μεταξύ τους με το διαχωριστικό "**;**":

- Επικεφαλίδα υποπρογράμματος
- Δηλώσεις τύπων (προαιρετικά)
- Δηλώσεις μεταβλητών (προαιρετικά)
- Ορισμοί υποπρογραμμάτων (προαιρετικά)
- Σύνθετη εντολή υποπρογράμματος

Στην επικεφαλίδα αναφέρεται κατ' αρχήν αν το υποπρόγραμμα είναι διαδικασία ή συνάρτηση, το όνομά του, οι παράμετροί του μέσα σε παρενθέσεις και, αν πρόκειται για συνάρτηση, ο τύπος του αποτελέσματος, ο οποίος μπορεί να είναι βασικός ή σύνθετος. Οι παρενθέσεις είναι υποχρεωτικές ακόμη κι αν ένα υποπρόγραμμα δεν έχει παραμέτρους. Ακολουθούν παραδείγματα έγκυρων επικεφαλίδων συναρτήσεων (το "**;**" είναι το διαχωριστικό από το επόμενο στοιχείο της δομικής μονάδας):

```
procedure p1 ();
procedure p2 (n: integer);
procedure p3 (a, b: integer; b: boolean);
function  f1 (x: real): real;
function  f2 (s: array of char): integer;
function  f3 (x: real): array [10] of real;
```

Στη συνέχεια, ένα υποπρόγραμμα μπορεί να περιέχει δηλώσεις μεταβλητών (προαιρετικά) και ορισμούς άλλων υποπρογραμμάτων (προαιρετικά) που διαχωρίζονται μεταξύ τους με το "**;**". Οι δηλώσεις αυτές είναι τοπικές και ισχύουν μόνο για τη δομική μονάδα όπου ανήκουν. Τέλος, ένα υποπρόγραμμα καταλήγει υποχρεωτικά με μια σύνθετη εντολή, η οποία καθορίζει τη λειτουργία του υποπρογράμματος, εκτελείται όταν καλείται το υποπρόγραμμα και μπορεί να κάνει χρήση των μεταβλητών και των άλλων υποπρογραμμάτων που έχουν δηλωθεί μέσα στην ίδια δομική μονάδα του υποπρογράμματος.

Η **FL** υποστηρίζει ένα σύνολο προκαθορισμένων διαδικασιών και συναρτήσεων, οι οποίες βρίσκονται στη διάθεση του προγραμματιστή για χρήση οπουδήποτε μέσα στο πρόγραμμα. Παρακάτω, δίνονται οι επικεφαλίδες τους:

```
readString(): array of char;

readInteger(): integer;

readReal(): real;

writeString(s: array of char);

writeInteger(j: integer);

writeReal(j: real);
```

v) Εκφράσεις

Οι εκφράσεις (expressions) είναι ίσως το πιο σημαντικό κομμάτι μιας γλώσσας προγραμματισμού. Οι βασικές μορφές εκφράσεων είναι οι σταθερές, οι μεταβλητές οποιουδήποτε τύπου και οι κλήσεις συναρτήσεων. Σύνθετες μορφές εκφράσεων προκύπτουν με τη χρήση τελεστών και παρενθέσεων.

Οι τελεστές της **FL** διακρίνονται σε τελεστές με ένα όρισμα και τελεστές με δύο ορίσματα. Από τους πρώτους, ορισμένοι γράφονται πριν το όρισμα (prefix) και ορισμένοι μετά (postfix), ενώ οι δεύτεροι γράφονται πάντα μεταξύ των ορισμάτων (infix). Η αποτίμηση των ορισμάτων των τελεστών με δυο ορίσματα γίνεται από αριστερά προς τα δεξιά. Στον παρακάτω πίνακα

ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της **FL**. Προηγούνται οι τελεστές που εμφανίζονται πιο ψηλά στον πίνακα. Όσοι τελεστές βρίσκονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα. Σημειώστε ότι μπορούν να χρησιμοποιηθούν παρενθέσεις σε μια έκφραση για να δηλωθεί η επιθυμητή προτεραιότητα.

Τελεστές	Περιγραφή	Ορίσματα	Θέση Προσεταιριστικότητα
**not ! **	Τελεστής λογικής άρνησης	1	prefix, δεξιά
+ -	Τελεστές προσήμου	1	prefix, δεξιά
(<data-type>)	Τελεστές casting	1	prefix, δεξιά
*** / div mod**	Τελεστές με παράγοντες	2	infix, αριστερή
+ -	Τελεστές με όρους	2	infix, αριστερή
= <> < > <= >=	Σχεσιακοί τελεστές	2	infix, αριστερή
and &&	Λογική σύζευξη	2	infix, αριστερή
or ||	Λογική διάζευξη	2	infix, αριστερή

Ακολουθούν παραδείγματα σωστών εκφράσεων:

```
-a                    // αντίθετος της μεταβλητής a
a + b * (b / a)       // αριθμητική έκφραση
4 + 50.0*x / 2.45     // αριθμητική έκφραση
(a+1) mod cube(b+3)   // τελεστής αύξησης, κλήση συνάρτησης
(a <= b) and (c >= d) // τελεστές λογικοί με σχεσιακούς
a + (c <> d)          // τελεστές αριθμητικοί με σχεσιακούς
a + b[1][k][(k+1)*2]  // αριθμητική έκφραση με πίνακα
15.7e-2 + (real) 45   // αριθμητική έκφραση με casting
```

vi) Εντολές

Οι εντολές που υποστηρίζει η γλώσσα **FL** είναι οι ακόλουθες (όλες οι εντολές, εκτός της σύνθετης, θεωρούνται απλές):

- Η *σύνθετη εντολή*, που αποτελείται από μια (πιθανά κενή) ακολουθία απλών εντολών διαχωρισμένων μεταξύ τους με το "**;**" που οριοθετείται από τους διαχωριστές "**begin**" και "**end**".
- Η *εντολή ανάθεσης* **v := e**, όπου v είναι μία μεταβλητή και e μια έκφραση.
- Η *ειδική εντολή ανάθεσης* **result := e** που εμφανίζεται μόνο μέσα σε συναρτήσεις. Η λέξη-κλειδί result υποδηλώνει τη μεταβλητή που περιέχει την τιμή που επιστρέφει η συνάρτηση.
- Η *εντολή ελέγχου* **if e then s_1 else s_2**. Το τμήμα του else είναι προαιρετικό. Το e είναι μια έκφραση, ενώ τα s_1 και s_2 είναι απλές ή σύνθετες εντολές.
- Οι *εντολές επανάληψης* **for v := e_1 to e_2 do s**, **for v := e_1 downto e_2 do s** όπου v είναι μία μεταβλητή, e_1 και e_2 είναι εκφράσεις και το s είναι απλή ή σύνθετη εντολή.
- Οι *εντολές βρόχου* **while e do s** και **repeat s until e**. Το e είναι μια έκφραση και το s μια απλή ή σύνθετη εντολή.
- Η *εντολή με ετικέτα* **l:s**, όπου l μια ετικέτα και s μια απλή ή σύνθετη εντολή.
- Η *εντολή άλματος* **goto l**, όπου l μια ετικέτα στην ίδια δομική μονάδα.

- Η *εντολή επιστροφής* **`return`**, που τερματίζει (πιθανά, πρόωρα) την εκτέλεση της δομικής μονάδας (υποπρόγραμμα ή κύρια δομική μονάδα) στην οποία βρίσκεται και επιστρέφει.
- Η *εντολή κλήσης* μιας διαδικασίας ή συνάρτησης **`f(`**e_1**`,...,`**e_n**`)`**, όπου `f` είναι το όνομα της συνάρτησης και $e_1, \ldots, e_n$ είναι εκφράσεις που αντιστοιχούν στα δηλωθέντα ορίσματα.

2.3 Αντιστοίχιση από την FL στη C99

Η C99 είναι η αναθεώρηση του standard της γλώσσας C που έγινε το 1999. Στην αναθεώρηση αυτή προστέθηκαν διάφορες χρησιμότατες επεκτάσεις στην κάπως παλιά C89. Δείτε το αντίστοιχο άρθρο της Wikipedia για παραπάνω λεπτομέρειες. Να σημειωθεί ότι η τελευταία αναθεώρηση της C έγινε το 2011 (και έχουμε τη C11). Καθώς η C99 υποστηρίζει εμφωλευμεένες συναρτήσεις (nested functions), είναι ιδιαίτερα εύκολο να αντιστοιχίσει κανείς προγράμματα της **FL** σε προγράμματα της C99. Τις λεπτομέρειες της απεικόνισης αυτής θα περιγράψουμε τώρα:

2.3.1 Αντιστοίχιση τύπων και σταθερών

Οι τύποι της Ptuc αντιστοιχίζονται με τους τύπους της C99 με βάση τον παρακάτω πίνακα:

Τύπος της FL	**Αντιστοιχημένος τύπος της C99**
integer	int
boolean	int
char	char
real	double
array [n1]...[nk] of T	T[n1]...[nk]
array of T	T*
function (a1: *T1*; ... ak: *Tk*) : *Treturn*	Treturn (*)(T1 a1, ... Tk ak)

Στη βάση του παραπάνω πίνακα αντιστοιχίζονται και οι σταθερές της **FL** σε σταθερές της C99. Για παράδειγμα, οι boolean σταθερές της **FL**, `true` και `false`, αντιστοιχίζονται σε ακέραιες τιμές.

2.3.2 Αντιστοίχιση δομικών μονάδων

Μια δομική μονάδα περιλαμβάνει προαιρετικά δηλώσεις μεταβλητών, functions, procedures και υποχρεωτικά το τμήμα του κυρίως κώδικα.

Η κύρια δομική μονάδα λοιπόν, αντιστοιχεί σε ένα αρχείο .c που περιλαμβάνει, δηλώσεις global μεταβλητών, συναρτήσεων και την αρχική ρουτίνα `main()`.

Η αντιστοίχιση είναι ως εξής:

- Ένας ονοματισμένος τύπος,

```
type
        foo = T;
```

αντιστοιχεί σε ονοματισμένο τύπο, πχ. κάτι σαν

```
typedef  T foo;
```

- Μια **FL** μεταβλητή foo με τύπο T αντιστοιχεί σε μια μεταβλητή με το ίδιο όνομα και με τον αντιστοιχημένο τύπο

```
var
   foo, bar: T;
```

αντιστοιχεί σε κάτι σαν

```
T foo, bar;
```

- Μια υπορρουτίνα αντιστοιχεί σε συνάρτηση της C99 με το ίδιο όνομα και αντιστοιχημένους τύπους παραμέτρων.

Υποπρόγραμμα	Συνάρτηση της C99
`function foo(x1, x2: T1;... ; xn: Tn): Tr`	`Tr foo(T1 x1, T1 x2,...,Tn xn)`
`procedure foo(x1: T1;... xn: Tn)`	`void foo(T1 x1, ... ,Tn xn)`

- Οι εντολές προγράμματος αντιστοιχούνται με προφανή τρόπο.
- Οι κλήσεις βιβλιοθήκης μπορούν να υλοποιηθούν ως εξής:

Κλήση FL	**Συνάρτηση υλοποίησης σε C99**
`readString(): array of char`	`gets()`
`readInteger(): integer`	`atoi(gets())`
`readReal(): real`	`atof(gets())`
`writeString(s: array of char)`	`puts(s)`
`writeInteger(j: integer)`	`printf("%d", j)`
`writeReal(j: real)`	`printf("%g", j)`

3 Αναλυτική περιγραφή εργασίας

3.1 Τα εργαλεία

Για να ολοκληρώσετε επιτυχώς την εργασία χρειάζεται να γνωρίζετε καλά προγραμματισμό σε **C**, **flex** και **bison**. Τα εργαλεία flex και bison έχουν αναπτυχθεί στα πλαίσια του προγράμματος GNU και μπορείτε να τα βρείτε σε όλους τους κόμβους του διαδικτύου που διαθέτουν λογισμικό GNU (π.χ. www.gnu.org). Περισσότερες πληροφορίες, εγχειρίδια και συνδέσμους για τα δύο αυτά εργαλεία θα βρείτε στην ιστοσελίδα του μαθήματος.

Στο λειτουργικό σύστημα Linux (οποιαδήποτε διανομή) τα εργαλεία αυτά είναι συνήθως ενσωματωμένα. Αν δεν είναι, μπορούν να εγκατασταθούν τα αντίστοιχα πακέτα πολύ εύκολα. Οι οδηγίες χρήσης που δίνονται παρακάτω για τα δύο εργαλεία έχουν δοκιμαστεί στη διανομή Linux Ubuntu και Mint, αλλά είναι πιθανόν να υπάρχουν μικροδιαφορές σε άλλες διανομές.

3.2 Προσέγγιση της εργασίας

Για τη δική σας διευκόλυνση στην κατανόηση των εργαλείων που θα χρησιμοποιήσετε καθώς και του τρόπου με τον οποίο τα εργαλεία αυτά συνεργάζονται, προτείνεται η υλοποίηση της εργασίας σε δύο φάσεις.

- **1η φάση: Λεκτική ανάλυση**

Το τελικό προϊόν αυτής της φάσης θα είναι ένας λεκτικός αναλυτής, δηλαδή ένα πρόγραμμα το οποίο θα παίρνει ως είσοδο ένα αρχείο με κάποιο πρόγραμμα της γλώσσας **FL** και θα αναγνωρίζει τις λεκτικές μονάδες (tokens) στο αρχείο αυτό. Η έξοδός του θα είναι μία λίστα από τα tokens που διάβασε και ο χαρακτηρισμός τους. Για παράδειγμα, για είσοδο

```
i := k + 2;
```

η έξοδος του προγράμματός σας θα πρέπει να είναι

```
token VAR_IDENTIFIER:   i
token ASSIGNMENT:       :=
token VAR_IDENTIFIER:   k
token OP_PLUS:              +
token INTEGER:          2
token SEMICOLON:        ;
```

Σε περίπτωση μη αναγνωρίσιμης λεκτικής μονάδας θα πρέπει να τυπώνεται κάποιο κατάλληλο μήνυμα λάθους στην οθόνη και να τερματίζεται η λεκτική ανάλυση. Για παράδειγμα, για τη λανθασμένη είσοδο

```
i := k ^ 2;
```

η έξοδος του προγράμματός σας θα πρέπει να είναι

```
token VAR_IDENTIFIER:   i
token ASSIGNMENT:           :=
token VAR_IDENTIFIER:   k

Unrecognized token ^ in line 46: i := k ^ 2;
```

όπου 46 είναι ο αριθμός της γραμμής μέσα στο αρχείο εισόδου όπου βρίσκεται η συγκεκριμένη εντολή συμπεριλαμβανομένων των γραμμών σχολίων.

Για να φτιάξετε ένα λεκτικό αναλυτή θα χρησιμοποιήσετε το εργαλείο flex και τον compiler gcc. Δώστε `man flex` στη γραμμή εντολών για να δείτε το manual του flex ή ανατρέξτε στο PDF αρχείο που βρίσκεται στο courses. Τα αρχεία με κώδικα flex έχουν προέκταση `.l`. Για να μεταγλωττίσετε και να τρέξετε τον κώδικά σας ακολουθήστε τις οδηγίες που δίνονται παρακάτω.

1. Γράψτε τον κώδικα flex σε ένα αρχείο με προέκταση `.l`, π.χ. `mylexer.l`.

2. Μεταγλωττίστε, γράφοντας `flex mylexer.l` στη γραμμή εντολών.
3. Δώστε `ls` για να δείτε το αρχείο `lex.yy.c` που παράγεται από τον flex.
4. Δημιουργήστε το εκτελέσιμο με `gcc -o mylexer lex.yy.c –lfl`
5. Αν δεν έχετε λάθη στο `mylexer.l`, παράγεται το εκτελέσιμο `mylexer`.
6. Εκτελέστε με `./mylexer < example.fl`, για είσοδο `example.fl`.

Κάθε φορά που αλλάζετε το `mylexer.l` θα πρέπει να κάνετε όλη την διαδικασία:

```
flex mylexer.l
gcc -o mylexer lex.yy.c -lfl
./mylexer < example.fl
```

Επομένως, είναι καλή ιδέα να φτιάξετε ένα script ή ένα makefile για να κάνει όλα τα παραπάνω αυτόματα.

- ## 2η φάση: Συντακτική ανάλυση και μετάφραση

Το τελικό προϊόν αυτής της φάσης θα είναι ένας συντακτικός αναλυτής και μεταφραστής της **FL** σε **C**, δηλαδή ένα πρόγραμμα το οποίο θα παίρνει ως είσοδο ένα αρχείο με κάποιο πρόγραμμα της γλώσσας **FL** και θα αναγνωρίζει αν αυτό το πρόγραμμα ακολουθεί τους συντακτικούς κανόνες της **FL**. Στην έξοδο θα παράγει το πρόγραμμα που αναγνώρισε, στη γλώσσα C, εφόσον το πρόγραμμα που δόθηκε είναι συντακτικά σωστό ή διαφορετικά θα εμφανίζεται ο αριθμός γραμμής όπου διαγνώσθηκε το πρώτο λάθος, το περιεχόμενο της γραμμής με το λάθος και *προαιρετικά* ένα κατατοπιστικό μήνυμα διάγνωσης. Για παράδειγμα, για τη λανθασμένη είσοδο

```
...
i := k + 2 * ;
...
```

το πρόγραμμά σας θα πρέπει να τερματίζει με ένα από τα παρακάτω μηνύματα λάθους

Syntax error in line 46: `i := k + 2 * ;`

Syntax error in line 46: `i := k + 2 * ; (expression expected)`

όπου 46 είναι ο αριθμός της γραμμής μέσα στο αρχείο εισόδου όπου βρίσκεται η συγκεκριμένη εντολή συμπεριλαμβανομένων των γραμμών σχολίων.

Για να φτιάξετε ένα συντακτικό αναλυτή και μεταφραστή θα χρησιμοποιήσετε το εργαλείο bison και τον compiler gcc. Δώστε `man bison` για να δείτε το manual του bison. Τα αρχεία με κώδικα bison έχουν προέκταση `.y`. Για να μεταγλωττίσετε και να τρέξετε τον κώδικά σας ακολουθήστε τις οδηγίες που δίνονται παρακάτω.

1. Υποθέτουμε ότι έχετε ήδη έτοιμο το λεκτικό αναλυτή στο `mylexer.l`.
2. Γράψτε τον κώδικα bison σε αρχείο με προέκταση `.y`, π.χ. `myanalyzer.y`.
3. Για να ενώσετε το flex με το bison πρέπει να κάνετε τα εξής:
   a. Βάλτε τα αρχεία `mylexer.l` και `myanalyzer.y` στο ίδιο directory.
   b. Βγάλτε την συνάρτηση `main` από το flex αρχείο και φτιάξτε μια `main` στο bison αρχείο. Για αρχή το μόνο που χρειάζεται να κάνει η καινούρια `main` είναι να καλεί μια φορά την μακροεντολή του bison `yyparse()`. Η `yyparse()` τρέχει επανειλημμένα την `yylex()` και προσπαθεί να αντιστοιχίσει κάθε token που επιστρέφει ο λεκτικός αναλυτής στη γραμματική που έχετε γράψει στο συντακτικό αναλυτή. Επιστρέφει 0 για επιτυχή τερματισμό και 1 για τερματισμό με συντακτικό σφάλμα.
   c. Αφαιρέστε τα defines που είχατε κάνει για τα tokens στο flex ή σε κάποιο άλλο `.h` αρχείο. Αυτά θα δηλωθούν τώρα στο bison αρχείο ένα σε κάθε γραμμή με την

εντολή `%token`. Όταν κάνετε compile το `myanalyzer.y` δημιουργείται αυτόματα και ένα αρχείο με όνομα `myanalyzer.tab.h`. Το αρχείο αυτό θα πρέπει να το κάνετε include στο αρχείο `mylexer.l` και έτσι ο flex θα καταλαβαίνει τα ίδια tokens με τον bison.

4. Μεταγλωττίστε τον κώδικά σας με τις παρακάτω εντολές:

```
bison -d –v –r all myanalyzer.y
flex mylexer.l
gcc -o mycompiler lex.yy.c myanalyzer.tab.c -lfl
```

5. Καλέστε τον εκτελέσιμο `mycompiler` για είσοδο `test.fl` γράφοντας: **`./mycompiler < test.fl`**

Προσοχή! Πρέπει πρώτα να κάνετε compile το `myanalyzer.y` και μετά το `mylexer.l` γιατί το `myanalyzer.tab.h` γίνεται include στο `mylexer.l`.

Το αρχείο κειμένου `myanalyzer.output` που παράγεται με το flag `–r all` θα σας βοηθήσει να εντοπίσετε πιθανά προβλήματα shift/reduce και reduce/reduce.

Κάθε φορά που αλλάζετε το `mylexer.l` και `myanalyzer.y` θα πρέπει να κάνετε όλη την διαδικασία. Είναι καλή ιδέα να φτιάξετε ένα script ή ένα makefile για όλα τα παραπάνω.

3.3 Παραδοτέα

Το παραδοτέο για την εργασία του μαθήματος θα περιέχει τα παρακάτω αρχεία (από τη 2η φάση):

- `mylexer.l`: Το αρχείο flex.
- `myanalyzer.y`: Το αρχείο bison.
- `mycompiler`: Το εκτελέσιμο αρχείο του αναλυτή σας.
- `correct1.fl`,`correct2.fl`: Δύο σωστά προγράμματα/παραδείγματα της **FL**.
- `correct1.c`,`correct2.c`: Τα ισοδύναμα προγράμματα των δύο παραπάνω σε γλώσσα **C**.
- `wrong1.fl`,`wrong2.fl`: Δύο λανθασμένα προγράμματα/παραδείγματα της **FL**.

Είναι δική σας ευθύνη να αναδείξετε τη δουλειά σας μέσα από αντιπροσωπευτικά προγράμματα.

3.4 Εξέταση

Κατά την εξέταση της εργασίας σας θα ελεγχθούν τα εξής:

- *Μεταγλώττιση των παραδοτέων προγραμμάτων και δημιουργία του εκτελέσιμου αναλυτή*. Ανεπιτυχής μεταγλώττιση σημαίνει ότι η εργασία σας δεν μπορεί να εξετασθεί περαιτέρω.
- *Επιτυχής δημιουργία του αναλυτή*. Ο βαθμός σας θα εξαρτηθεί από τον αριθμό των shift-reduce και reduce-reduce conflicts που εμφανίζονται κατά τη δημιουργία του αναλυτή σας.
- *Έλεγχος αναλυτή σε σωστά και λανθασμένα παραδείγματα προγραμμάτων **FL***. Θα ελεγχθούν σίγουρα αυτά του Παραρτήματος, αλλά και άλλα άγνωστα σ' εσάς παραδείγματα. Η καλή εκτέλεση τουλάχιστον των γνωστών παραδειγμάτων θεωρείται αυτονόητη.
- *Έλεγχος αναλυτή στα δικά σας παραδείγματα προγραμμάτων **FL***. Τέτοιοι έλεγχοι θα βοηθήσουν σε περίπτωση που θέλετε να αναδείξετε κάτι από τη δουλειά σας.

- *Ερωτήσεις σχετικά με την υλοποίηση.* Θα πρέπει να είστε σε θέση να εξηγήσετε θέματα σχεδιασμού, επιλογών και τρόπων υλοποίησης.

4 Επίλογος

Κλείνοντας θα θέλαμε να τονίσουμε ότι είναι σημαντικό να ακολουθείται πιστά τις οδηγίες και να παραδίδετε αποτελέσματα σύμφωνα με τις προδιαγραφές που έχουν τεθεί. Αυτό είναι κάτι που πρέπει να τηρείται ως μηχανικοί για να μπορέσετε στο μέλλον να εργάζεσθε συλλογικά σε μεγάλες ομάδες εργασίας, όπου η συνέπεια είναι το κλειδί για τη συνοχή και την επιτυχία του κάθε έργου.

Στη διάρκεια του εξαμήνου θα δοθούν διευκρινίσεις όπου χρειάζεται. Για ερωτήσεις μπορείτε να απευθύνεστε στους διδάσκοντες και στους βοηθούς του μαθήματος. Γενικές απορίες καλό είναι να συζητώνται στο χώρο συζητήσεων του μαθήματος για να τις βλέπουν και οι συνάδελφοί σας.

Καλή επιτυχία!

ΠΑΡΑΡΤΗΜΑ

5 Παραδείγματα προγραμμάτων της FL

5.1 Hello World!

```
(* My first Ptuc program *)
program hello;

@defmacro message "Hello World!\n"

begin
    writeString(message)
end.
```

Ζητούμενο αποτέλεσμα λεκτικής – συντακτικής ανάλυσης:

```
Token KEYWORD_PROGRAM:          program
Token IDENTIFIER:               hello
Token SEMICOLON:                ;
Token KEYWORD_BEGIN:            begin
Token IDENTIFIER:               writeString
Token LEFT_PARENTHESIS:   (
Token STRING:             "Hello World!\n"
Token RIGHT_PARENTHESIS:  )
Token KEYWORD_END:        end
Token DOT:                      .

Your program is syntactically correct!
```

5.2 Δομικές μονάδες FL

Παράδειγμα για την κατανόηση της σύνταξης δομικών μονάδων στη γλώσσα **FL**.

```
# A useless piece of FL code

@defmacro N 100

program useless;

    var i, k: integer;

    procedure add(n: integer; k:integer);
        var j: integer;

        function cube(i: integer): integer;

        begin
            result := i * i * i;
            return
        end;

    begin
        j := (integer) (N - n) + cube(k);
        writeInteger(j)
    end;

(* Here you can see some useless lines.
** Just for testing the multi-line comments ...
*)

begin
    k := readInteger();
    i := readInteger();
    add(k,i) #Here you can see some dummy comments!
end.
```

Το παραπάνω πρόγραμμα θα μπορούσε να ενδεικτικά να μεταφραστεί ως εξής:

```
#include <stdio.h>
/* FL library */
int readInteger() { int ret; scanf("%d", &ret); return ret; }
void writeInteger(int n) { printf("%d",n); }

int i,k;

void add(int n, int k)
{
        int j;
        int cube(int i) {
                int result;
                result = i*i*i;
                return result;
        }

        j = (int)(100-n) + cube(k);
        writeInteger(j);
}

void ptuc_main()
{
        k = readInteger();
        i  = readInteger();
        add(k,i);
}

int main()
{
        ptuc_main();
        return 0;
}
```

Μπορεί να μεταφραστεί από τον compiler με την εντολή

```
gcc -std=c99 -Wall  myprog.c
```

5.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα **FL** είναι ένα πρόγραμμα που υπολογίζει τους πρώτους αριθμούς μεταξύ **1** και **n**, όπου το **n** δίνεται από το χρήστη.

```
program calculate;

    var limit, number, counter: integer;

    function prime(n: integer): boolean;

        var
           i             : integer;
           isPrime  : boolean;

    begin

        if n < 0 then
            result := prime(-n)
        else if n < 2 then
            result := false
        else if n = 2 then
            result := true
        else if n mod 2 = 0 then
            result := false
        else begin
            i := 3;
            isPrime := true;
            while isPrime and (i < n div 2) do begin
                isPrime := n mod i <> 0;
                i := i+2
            end;
            result := isPrime
        end;
        return

    end;

begin

    limit := readInteger();
    counter := 0;
    number := 2;

    while number <= limit do begin

        if prime(number) then begin
            counter := counter + 1;
            writeInteger(number);
            writeString(" ")
        end;

        number := number + 1
    end;
    writeString("\n");
    writeInteger(counter)

end.
```

5.4 Παράδειγμα με συντακτικό λάθος

```
1 (* My first Ptuc program *)
2 program hello;
3
4 @defmacro message "Hello World!\n"
5
6 begin
7     writeString(message
8 end.
```

Ζητούμενο αποτέλεσμα λεκτικής – συντακτικής ανάλυσης:

```
Token KEYWORD_PROGRAM:          program
Token IDENTIFIER:               hello
Token SEMICOLON:                ;
Token KEYWORD_BEGIN:            begin
Token IDENTIFIER:               writeString
Token LEFT_PARENTHESIS:   (
Token STRING:             "Hello World!\n"
Token RIGHT_PARENTHESIS:  )
Token KEYWORD_END:        end
```

```
Syntax error in line 8:      writeString(message
```

ή

```
Syntax error in line 8:      writeString(message
(Missing parenthesis)
```