ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΠΛΗ 402
Θεωρία Υπολογισμού

Εργασία Προγραμματισμού
Λεκτική και Συντακτική Ανάλυση της Γλώσσας Προγραμματισμού FC

Διδάσκων
Μιχαήλ Γ. Λαγουδάκης

Βοηθός
Γιώργος Ανέστης

Εαρινό Εξάμηνο 2017

Τελευταία ενημέρωση: 2017-03-16

1 Εισαγωγή

Η εργασία προγραμματισμού του μαθήματος **«ΠΛΗ 402 – Θεωρία Υπολογισμού»** έχει ως στόχο την 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 στην σχεδίαση και υλοποίηση των αρχικών σταδίων ενός μεταγλωττιστή (compiler) για την φανταστική γλώσσα προγραμματισμού **FC** (**F**ictional **C**), 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 **FC** 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 την γλώσσα **C**.

Η εργασία περιλαμβάνει δύο τμήματα:

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

Παρατηρήσεις

- Η εργασία θα εκπονηθεί **ατομικά**. Η τυφλή αντιγραφή (plagiarism), ακόμη και από παλαιότερες εργασίες, μπορεί να διαπιστωθεί πολύ εύκολα και οδηγεί σε μηδενισμό.
- 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, .tar, .rar) 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. Μη παράδοση της εργασίας οδηγεί αυτόματα σε αποτυχία στο μάθημα.
- Η αξιολόγηση της εργασίας θα περιλαμβάνει **εξέταση καλής λειτουργίας** του παραδοτέου προγράμματος σύμφωνα με τις προδιαγραφές καθώς και **προφορική εξέταση**. Η εξέταση θα γίνει στην Πολυτεχνειούπολη, σε ημέρες και ώρες που θα ανακοινωθούν.
- Υπενθύμιση: ο βαθμός της εργασίας θα πρέπει να είναι τουλάχιστον **50/100**.

2 Η γλώσσα προγραμματισμού FC

Η γλώσσα **FC** (**F**ictional **C**) βασίζεται στη γλώσσα **C**. Λόγω ομοιοτήτων της **FC** με τη **C**, η περιγραφή παρακάτω τονίζει κυρίως σημεία όπου οι δύο γλώσσες διαφέρουν. Σε περιπτώσεις πιθανής ασάφειας, μπορείτε να ανατρέξετε στην περιγραφή της C. Η περιγραφή της γλώσσας παρακάτω πιθανότατα περιέχει και στοιχεία τα οποία δεν εντάσσονται στην 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 **FC** 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 **FC** χωρίζονται στις παρακάτω κατηγορίες:

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

static	**boolean**	**integer**	**char**	**real**
true	**false**	**string**	**void**	**while**
do	**break**	**continue**	**return**	**begin**
if	**else**	**for**	**end**	**or**
not	**and**	**mod**		

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

- Τα ***αναγνωριστικά*** *(identifiers)* που χρησιμοποιούνται για ονόματα μεταβλητών, συναρτήσεων και κλάσεων 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` `my_value` `Distance_02`

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

  Παραδείγματα: `42.0` `4.2e1` `0.420E+2` `1234.12345678e-123`

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

- Οι ***σταθεροί χαρακτήρες*** *(constant characters)*, που αποτελούνται από ένα χαρακτήρα μέσα σε μονά εισαγωγικά. Ο χαρακτήρας αυτός μπορεί να είναι οποιοσδήποτε κοινός χαρακτήρας ή ακολουθία διαφυγής (escape sequence). Κοινοί χαρακτήρες είναι όλοι οι εκτυπώσιμοι χαρακτήρες πλην των

απλών και διπλών εισαγωγικών και του χαρακτήρα \ (backslash). Οι ακολουθίες διαφυγής ξεκινούν με το χαρακτήρα \ (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)

Παραδείγματα: `'a'` `'M'` `'1'` `'~'` `'@'` `'\n'` `'\''`

- Οι ***σταθερές συμβολοσειρές*** *(constant strings)*, που αποτελούνται από μια ακολουθία κοινών χαρακτήρων ή ακολουθιών διαφυγής μέσα σε διπλά εισαγωγικά. Οι σταθερές συμβολοσειρές δεν μπορούν να εκτείνονται σε περισσότερες από μία γραμμές του αρχείου εισόδου.

  Παραδείγματα: `"abc"` `"Route 66"` `"Hello world!\n"`

  `"Item:\t\"Laser Printer\"\nPrice:\t$142\n"`

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

  αριθμητικοί τελεστές: **+ - * / mod**
  σχεσιακοί τελεστές: **= >= <= > < !=**
  λογικοί τελεστές: **and or not && || !**
  τελεστής ανάθεσης: **:=**
  τελεστές προσήμου: **+ -**

Τελεστής	Περιγραφή
+ - * / mod	Πρόσθεση, Αφαίρεση, Πολλαπλασιασμός, Διαίρεση, Υπόλοιπο
= >= <=	Ίσο (=), Μεγαλύτερο ή ίσο (≥), Μικρότερο ή ίσο (≤)
> < !=	Μεγαλύτερο από (>), Μικρότερο από (<), Διάφορο από (≠)
and &&	Λογική σύζευξη
or ||	Λογική διάζευξη
not !	Λογική άρνηση

- Τους ***διαχωριστές***, οι οποίοι είναι οι παρακάτω:

  begin end ; ( ) , [ ]

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα **FC** 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)*, τα οποία ξεκινούν με την ακολουθία χαρακτήρων **//** και εκτείνονται ως το τέλος της τρέχουσας γραμμής.

2.2 Συντακτικοί κανόνες

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

2.2.1 Προγράμματα

Ένα πρόγραμμα **FC** μπορεί να βρίσκεται μέσα σε ένα αρχείο κειμένου με κατάληξη **.fc** και αποτελείται από τις παρακάτω δομικές μονάδες (αναλυτική περιγραφή τους δίνεται παρακάτω):

- Δηλώσεις μεταβλητών
- Δηλώσεις συναρτήσεων

Οι δηλώσεις αυτές πρέπει να εμφανίζονται ακριβώς με την παραπάνω σειρά και επίσης ισχύουν τα εξής:

1. Ένα πρόγραμμα **FC** μπορεί να έχει **μηδέν ή περισσότερες** δηλώσεις μεταβλητών.
2. Ένα πρόγραμμα **FC** πρέπει να έχει **μία ή περισσότερες** δηλώσεις συναρτήσεων. Πιο συγκεκριμένα, θα πρέπει να υπάρχει υποχρεωτικά τουλάχιστον ο ορισμός της συνάρτησης με επικεφαλίδα **integer main()** απ' όπου και ξεκινά η εκτέλεση του προγράμματος.

2.2.2 Τύποι δεδομένων

Η γλώσσα **FC** υποστηρίζει τους ακόλουθους βασικούς τύπους δεδομένων:

- **integer** : ακέραιοι αριθμοί
- **boolean** : λογικές τιμές
- **character** : χαρακτήρες
- **real** : πραγματικοί αριθμοί
- **string** : συμβολοσειρές (μεγέθους το πολύ 128 χαρακτήρων)

2.2.3 Μεταβλητές

Οι δηλώσεις μεταβλητών γίνονται με την αναγραφή του τύπου ακολουθούμενου από ένα ή περισσότερα ονόματα μεταβλητών (χωρισμένων με το διαχωριστικό **,**) και καταλήγουν με το διαχωριστικό **;**.

<type> <identifier1>, <identifier2>, ..., <identifierk>;

Ο τύπος κάθε μεταβλητής μπορεί να είναι ένας από τους βασικούς. Παραδείγματα δηλώσεων είναι:

```
integer i;    real a, b, c;    string phrase, word;
```

Η **FC** υποστηρίζει επίσης πίνακες δεδομένων, δηλαδή δήλωση πολυδιάστατων πινάκων ενός βασικού τύπου με την αναγραφή του μεγέθους του πίνακα σε κάθε διάσταση δίπλα στο αντίστοιχο αναγνωριστικό:

<type> <identifier>[n][m]…[k];

Τα n, m, …, k υποδηλώνουν το μέγεθος του πίνακα σε κάθε διάσταση και θα πρέπει να είναι θετικές ακέραιες σταθερές. Παραδείγματα δηλώσεων πινάκων:

```
real spik[45]; integer sp[23][8], spk[45]; character spak[2][2];
```

Συνδυασμοί απλών μεταβλητών και πινάκων του ιδίου τύπου στην ίδια δήλωση επιτρέπονται. Επιτρέπεται επίσης η αρχικοποίηση απλών μεταβλητών (όχι πινάκων) κατά τη δήλωση με σταθερές και προαιρετική χρήση του τελεστή προσήμου. Για παράδειγμα:

```
real x:=-100.50, matrix[40][100], y_24:=+30e-4;
```

Οι δείκτες θέσεων πινάκων μπορούν να είναι ακέραιες (θετικές) σταθερές ή εκφράσεις με θετική τιμή. Ακολουθούν παραδείγματα προσδιορισμού στοιχείων πινάκων:

```
matrix[1][5]                y[k][(k+2)*n]
```

Επιπλέον στη δήλωση μεταβλητών υποστηρίζεται και ο όρος **static**. Για παράδειγμα:

```
static integer I:=25, k[40][5];
```

2.2.4 Συναρτήσεις

Κάθε συνάρτηση είναι μια δομική μονάδα που αποτελείται από την επικεφαλίδα ακολουθούμενη από το σώμα της. Στην επικεφαλίδα αναφέρεται το όνομα της συνάρτησης, οι τυπικές της παράμετροι μέσα σε παρενθέσεις και ο τύπος του αποτελέσματος (ένας από τους βασικούς τύπους, όχι πίνακες). Οι παρενθέσεις είναι υποχρεωτικές ακόμα και αν μία συνάρτηση δεν έχει τυπικές παραμέτρους. Επίσης, αν η συνάρτηση δεν επιστρέφει τιμή, τότε ο τύπος της ορίζεται ως **void**. Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομά της και τον τύπο της (επιτρέπονται μόνο βασικοί τύποι, όχι πίνακες).

Το σώμα μιας συνάρτησης οριοθετείται από τους διαχωριστές **begin** και **end** και αποτελείται από δηλώσεις μεταβλητών και μια ακολουθία από εντολές (βλέπε παρακάτω) με αυτή τη σειρά εμφάνισης. Το σώμα μιας συνάρτησης δεν μπορεί να περιέχει δηλώσεις άλλων συναρτήσεων. Ακολουθεί παράδειγμα συνάρτησης με επικεφαλίδα και σώμα:

```
integer foo(integer k, real bound)
begin
     integer p, i:=0, z:=0;
     p:=34*k;
     for(i:=1; i<=k; i++)
          if (z<bound)
               z:=p*i;
     return z;
end
```

Η **FC** παρέχει και κάποιες προκαθορισμένες συναρτήσεις βιβλιοθήκης για είσοδο και έξοδο δεδομένων, οι οποίες βρίσκονται στη διάθεση του προγραμματιστή για χρήση χωρίς δήλωση. Παρακάτω, δίνονται οι επικεφαλίδες τους:

string readString()	**void writeString(string s)**
integer readInteger()	**void writeInteger(integer i)**
real readReal()	**void writeReal(real r)**

2.2.5 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 **FC** 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 **FC**. Προηγούνται οι τελεστές που εμφανίζονται πιο ψηλά στον πίνακα. Όσοι τελεστές βρίσκονται στο ίδιο κελί 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
+ -	Τελεστής προσήμου	1	prefix, δεξιά
*** / 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 υπολοίπου, 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
```

2.2.6 Εντολές

Οι εντολές που υποστηρίζει η γλώσσα **FC** είναι οι ακόλουθες (όλες οι εντολές, εκτός της σύνθετης, θεωρούνται απλές και, επίσης, κάθε απλή εντολή της γλώσσας **FC** τερματίζει με το ειδικό διαχωριστικό σύμβολο `;`, με εξαίρεση τις εντολές `if`, `for`, `while`, όπου η απαίτηση αυτή μεταφέρεται στις εσωτερικές εντολές, όπως αναγράφεται αναλυτικά παρακάτω):

1. Η *κενή εντολή* (`;`) που δεν κάνει καμία ενέργεια, όπως δηλώνει και το όνομά της.
2. Η *σύνθετη εντολή* που αποτελείται από μια (πιθανά κενή) ακολουθία απλών εντολών διαχωρισμένων μεταξύ τους με το `;` και οριοθετείται από τους διαχωριστές **`begin`** και **`end`**.
3. Η *εντολή ανάθεσης* **`v := e;`**, όπου `v` είναι μία μεταβλητή και `e` μια έκφραση.
4. Η *εντολή ελέγχου* **`if (e)`** s_1 **`else`** s_2. Το τμήμα `else` είναι προαιρετικό. Το `e` είναι μια έκφραση, ενώ τα s_1 και s_2 είναι απλές ή σύνθετες εντολές.

5. Η *εντολή επανάληψης* **for (s1;e;s2) s** όπου τα s1, s2 είναι απλές εντολές (κατ' εξαίρεση, χωρίς διαχωριστικό **;** στο τέλος) που εκτελούνται πριν την έναρξη και σε κάθε επανάληψη αντίστοιχα, η e είναι προαιρετική έκφραση που ελέγχεται/υπολογίζεται πριν από κάθε επανάληψη και το s είναι απλή ή σύνθετη εντολή που εκτελείται σε κάθε επανάληψη.
6. Οι *εντολές βρόχου* **while (e) s** και **do s while (e);**. Το e είναι μια έκφραση και το s μια απλή ή σύνθετη εντολή.
7. Η *εντολή διακοπής* **break;** που προκαλεί την άμεση έξοδο από τον πιο εσωτερικό βρόχο.
8. Η *εντολή συνέχισης* **continue;** που προκαλεί τη διακοπή της τρέχουσας επανάληψης και την έναρξη της επόμενης επανάληψης του βρόχου μέσα στον οποίο βρίσκεται.
9. Η *εντολή επιστροφής* **return e;** που τερματίζει (πιθανά, πρόωρα) την εκτέλεση μιας συνάρτησης και επιστρέφει μια τιμή που δίνεται από την έκφραση e.
10. Η *εντολή κλήσης* μιας συνάρτησης ή μεθόδου **f(e1,...,en);**, όπου f είναι το όνομα της συνάρτησης/μεθόδου και e1,...,en είναι εκφράσεις που αντιστοιχούν στα δηλωθέντα ορίσματα.

3 Αντιστοίχιση από την γλώσσα FC στη γλώσσα C

Ο τελικός στόχος της παρούσας εργασίας είναι ο παραγόμενος source-to-source compiler να παίρνει ως είσοδο τον πηγαίο κώδικα προγραμμάτων σε **FC** και να παράγει τον πηγαίο κώδικα ισοδύναμων προγραμμάτων στη γλώσσα **C**. Σε αυτό το πλαίσιο, οι κανόνες της γραμματικής του συντακτικού μας αναλυτή (για την ακρίβεια, οι αντίστοιχες ενέργειες των κανόνων που θα ενεργοποιούνται κάθε φορά) θα πρέπει να φροντίζουν και για την σωστή αντιστοίχιση του κώδικα της γλώσσας **FC** σε κατάλληλο – ισοδύναμο κώδικα της γλώσσας **C** λαμβάνοντας υπόψη τα ακόλουθα.

3.1 Αντιστοίχιση τύπων και σταθερών

Οι τύποι της **FC** αντιστοιχίζονται με τους τύπους της **C** με βάση των παρακάτω πίνακα:

Τύπος της FC	Αντιστοιχημένος τύπος της C
integer	**int**
boolean	**int**
char	**char**
real	**double**
T[n1]...[nk]	**T[n1]...[nk]**

Βάσει του παραπάνω πίνακα αντιστοιχίζονται και οι σταθερές της **FC** σε σταθερές της **C**. Για παράδειγμα, οι boolean σταθερές της **FC**, `true` και `false`, αντιστοιχίζονται στις ακέραιες τιμές `0` και `1` αντίστοιχα.

3.2 Αντιστοίχιση δομικών μονάδων

Κάθε δομική μονάδα της **FC** περιλαμβάνει δηλώσεις μεταβλητών (προαιρετικά) και συναρτήσεων. Η κύρια δομική μονάδα λοιπόν ενός προγράμματος σε **FC** αντιστοιχεί σε ένα αρχείο **.c** που περιλαμβάνει κάποιες αρχικές δηλώσεις απαραίτητες στη **C**, δηλώσεις (global) μεταβλητών (προαιρετικά) και συναρτήσεων, όπου περιλαμβάνεται και η αρχική συνάρτηση **int main()** της **C** ως αντίστοιχη της αρχικής συνάρτησης **integer main()** της **FC**.

Η αντιστοίχιση δοκιμών μονάδων μεταξύ των δύο γλωσσών γενικά θα έχει ως εξής:

- μια μεταβλητή `foo` τύπου `T` της **FC** αντιστοιχεί σε μια μεταβλητή με το ίδιο όνομα και με τον αντιστοιχημένο τύπο της **C**
- μια συνάρτηση `func` της **FC** αντιστοιχεί σε συνάρτηση με το ίδιο όνομα και αντιστοιχημένους τύπους παραμέτρων και τιμής επιστροφής της **C**
- Οι απλές και σύνθετες εντολές προγράμματος αντιστοιχίζονται με τον προφανή τρόπο
- Οι κλήσεις συναρτήσεων βιβλιοθήκης μπορούν να υλοποιηθούν ως εξής:

Κλήση της FC	**Συνάρτηση υλοποίησης σε C**
`string readString()`	`gets()`
`integer readInteger()`	`atoi(gets())`
`real readReal()`	`atof(gets())`
`void writeString(string s)`	`puts(s)`
`void writeInteger(integer i)`	`printf("%d", j)`
`void writeReal(real r)`	`printf("%g", r)`

4 Αναλυτική περιγραφή εργασίας

4.1 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 στις διανομές Linux Ubuntu και Mint, αλλά είναι πιθανόν να υπάρχουν κάποιες μικροδιαφορές σε άλλες διανομές.

4.2 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 **FC** 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 IDENTIFIER:       i
token ASSIGNMENT:       :=
token IDENTIFIER:       k
token OP_PLUS:          +
token CONST_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 IDENTIFIER:       i
token ASSIGNMENT:       :=
token IDENTIFIER:       k
```

Unrecognized token ^ in line 56: `i := k ^ 2;`

όπου 56 είναι ο αριθμός της γραμμής μέσα στο αρχείο εισόδου όπου βρίσκεται η συγκεκριμένη εντολή συμπεριλαμβανομένων των γραμμών σχολίων.

Για να φτιάξετε έναν λεκτικό αναλυτή θα χρησιμοποιήσετε το εργαλείο flex και τον compiler gcc. Δώστε `man flex` στην 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` στην 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.fc`, για είσοδο `example.fc`.

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.fc
```

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

- **2η φάση: Συντακτική ανάλυση**

Το τελικό προϊόν αυτής της φάσης θα είναι ένας συντακτικός αναλυτής και μεταφραστής της **FC** σε **C**, δηλαδή ένα πρόγραμμα το οποίο θα παίρνει ως είσοδο ένα αρχείο με κάποιο πρόγραμμα της γλώσσας **FC** και θα αναγνωρίζει αν αυτό το πρόγραμμα ακολουθεί τους συντακτικούς κανόνες της **FC**. Στην έξοδο θα παράγει το ισοδύναμο του προγράμματος που αναγνώρισε πλέον σε 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 56: `i := k + 2 * ;`
Syntax error in line 56: `i := k + 2 * ;(expression expected)`

όπου `56` είναι ο αριθμός της γραμμής μέσα στο αρχείο εισόδου όπου βρίσκεται η συγκεκριμένη εντολή συμπεριλαμβανομένων των γραμμών σχολίων.

Για να φτιάξετε έναν συντακτικό αναλυτή 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 τη 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.fc` γράφοντας:

```
./mycompiler < test.fc
```

<u>Προσοχή!</u> 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 τη διαδικασία. Είναι καλή ιδέα να φτιάξετε ένα script για όλα τα παραπάνω.

4.3 Παραδοτέα

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

- `mylexer.l`: Το αρχείο flex.
- `myanalyzer.y`: Το αρχείο bison.
- `mycompiler`: Το εκτελέσιμο αρχείο του αναλυτή σας.
- `report.pdf`: Συνοπτική (δακτυλογραφημένη, PDF μορφή) αναφορά που θα περιγράφει τις σχεδιαστικές σας επιλογές για τις κανονικές εκφράσεις και την γραμματική που επινοήσατε και θα εξηγεί τις τεχνικές που χρησιμοποιήσατε και τις ιδιαιτερότητες της εργασίας σας. Επίσης, θα πρέπει να αιτιολογεί τις όποιες συμβάσεις ή παραδοχές κάνατε και να αναφέρει τα στοιχεία της γλώσσας που θεωρήσατε ότι δεν εμπίπτουν στα πλαίσια της λεκτικής ή συντακτικής ανάλυσης.
- `correct1.fc`, `correct2.fc`: Δύο σωστά προγράμματα/παραδείγματα της **FC**.
- `correct1.c`, `correct2.c`: Τα ισοδύναμα προγράμματα των δύο παραπάνω σε γλώσσα **C**.
- `wrong1.fc`, `wrong2.fc`: Δύο λανθασμένα προγράμματα/παραδείγματα της **FC**.

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

4.4 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 σωστού 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 **FC***. Θα ελεγχθούν σίγουρα αυτά του Παραρτήματος και άλλα άγνωστα σ' εσάς παραδείγματα. Η καλή εκτέλεση τουλάχιστον των γνωστών παραδειγμάτων θεωρείται αυτονόητη.
- *Έλεγχος αναλυτή στα δικά σας παραδείγματα προγραμμάτων **FC***. Τέτοιοι έλεγχοι θα βοηθήσουν σε περίπτωση που θέλετε να αναδείξετε κάτι από τη δουλειά σας.
- *Ερωτήσεις σχετικά με την υλοποίηση και τα παραδοτέα κείμενα.* Θα πρέπει να είστε σε θέση να εξηγήσετε θέματα σχεδιασμού, επιλογών και τρόπων υλοποίησης.

5 Επίλογος

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

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

Καλή επιτυχία!

ΠΑΡΑΡΤΗΜΑ

6 Παραδείγματα προγραμμάτων της FC

6.1 Hello World!

```
/* My first FC program - myprog.fc*/

integer main()
begin
     writeString("Hello World!\n");
     return 0;
end
```

Ζητούμενο αποτέλεσμα λεκτικής – συντακτικής ανάλυσης:

```
Token KEYWORD_INTEGER:     integer
Token IDENTIFIER:          main
Token PARENTHESIS_LEFT:    (
Token PARENTHESIS_RIGHT:   )
Token KEYWORD_BEGIN:       begin
Token IDENTIFIER:          writeString
Token PARENTHESIS_LEFT:    (
Token CONST_STRING:        "Hello World!\n"
Token PARENTHESIS_RIGHT:   )
Token SEMICOLON:           ;
Token KEYWORD_RETURN:      return
Token CONST_INTEGER:       0
Token SEMICOLON:           ;
Token KEYWORD_END:         end

Your program is syntactically correct!
```

Το παραπάνω πρόγραμμα **myprog.fc** θα μπορούσε ***ενδεικτικά*** να μεταφραστεί σε **C** ως εξής:

```
#include <stdio.h>
/* FC library */
void writeString(char *str) { printf("%s",str); }

int main()
{
      writeString("Hello World!\n");
      return 0;
}
```

Το παραπάνω πρόγραμμα **myprog.c** μπορεί να μεταγλωττισθεί από τον compiler της **C** με την εντολή

gcc -Wall myprog.c

και στη συνέχεια να εκτελεσθεί.

6.2 Δομικές μονάδες FC

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

```
// A useless piece of FC code - useless.fc

integer i, k;

integer cube(integer i)
begin
     return i * i * i;
end

void add(integer n, integer k)
begin
     integer j;
     j := n + cube(k);
     writeInteger(j);
end

/* Here you can see some useless lines.
** Just for testing the multi-line comments ...
*/
integer main()
begin
     k := readInteger();
     i := readInteger();
     add(k,i); //Here you can see some dummy comments!
     return 0;
end
```

Το παραπάνω πρόγραμμα **useless.fc** θα μπορούσε ***ενδεικτικά*** να μεταφραστεί σε **C** ως εξής:

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

int i,k;
int cube(int i) {
      int result;
      result = i*i*i;
      return result;
}

void add(int n, int k)
{
      int j;

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

int main()
{
      k = readInteger();
      i = readInteger();
      add(k,i);
      return 0;
}
```

Το παραπάνω πρόγραμμα **useless.c** μπορεί να μεταγλωττισθεί από τον compiler της **C** με την εντολή

gcc -Wall useless.c

6.3 Πρώτοι αριθμοί

Το παρακάτω πρόγραμμα **FC** υπολογίζει τους πρώτους αριθμούς μεταξύ **1** και **n**, για το **n** που δίνεται.

```
boolean prime(integer n)
begin
      integer i; boolean isPrime, result;
      if (n < 0)
            result := prime(-n);
      else if (n < 2)
            result := false;
      else if (n = 2)
            result := true;
      eLse if (n mod 2 = 0)
            result := false;
      else
            begin
                  i := 3;
                  isPrime := true;
                  while ( isPrime && i <= n / 2 )
                  begin
                        isPrime := (n mod i != 0);
                        i := i+2;
                  end
                  result := isPrime;
            end
      return result;
end
integer main( )
begin
      integer limit, number, counter:=0;
      limit := readInteger();
      for (number:=1; number<=3; number:=number + 1)
            if (limit >= number)
                  begin
                        counter := counter + 1;
                        writeInteger(number);
                  end
      number := 6;
      while (number <= limit)
            begin
                  if (prime(number-1))
                        begin
                              counter := counter + 1;
                              writeInteger(number-1);
                        end
                  if ((number != limit) && prime(number+1))
                        begin
                              counter := counter + 1;
                              writeInteger(number+1);
                        end
                  number := number + 6;
            end
      writeInteger(counter);
      return 0;
end
```

6.4 Παράδειγμα με συντακτικό λάθος

```
/* My wrong FC program */

integer main()
begin
      writeString("Hello World!\n";
      return 0;
end
```

Ζητούμενο αποτέλεσμα λεκτικής – συντακτικής ανάλυσης:

```
Token KEYWORD_INTEGER:     integer
Token IDENTIFIER:          main
Token PARENTHESIS_LEFT:    (
Token PARENTHESIS_RIGHT:   )
Token KEYWORD_BEGIN:       begin
Token IDENTIFIER:          writeString
Token PARENTHESIS_LEFT:    (
Token CONST_STRING:        "Hello World!\n"
Token SEMICOLON:           ;


Syntax error in line 5:    writeString("Hello World!\n";
```

ή

```
Syntax error in line 5:    writeString("Hello World!\n";
(Missing parenthesis)
```