Μεταγλωττιστές

Λεκτική Ανάλυση

Δημήτρης Μιχαήλ

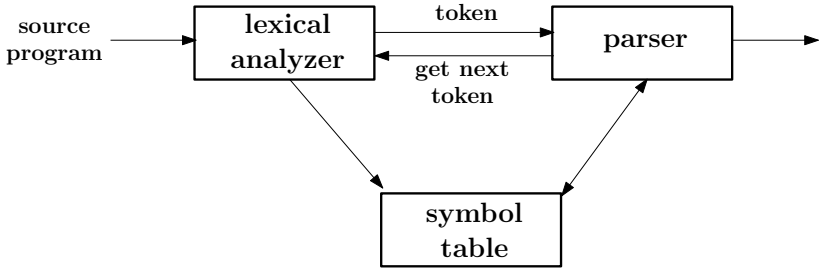
Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Λεκτικός Αναλυτής

Ο λεκτικός αναλυτής είναι η πρώτη φάση της μεταγλώττισης.

Σκοπός του είναι να διαβάσει μία είσοδο από χαρακτήρες και να παράξει μία σειρά από **λεκτικές μονάδες** (tokens) που θα χρησιμοποιηθούν στην συνέχεια για την συντακτική ανάλυση.

Λεκτικός Αναλυτής και Συντακτικός Αναλυτής



Ο λεκτικός αναλυτής υλοποιείται συνήθως ως υπορουτίνα του συντακτικού αναλυτή.

Ο συντακτικός αναλυτής ζητάει από τον λεκτικό αναλυτή μία καινούρια λεκτική μονάδα και ο λεκτικός αναλυτής διαβάζει χαρακτήρες μέχρι να μπορεί να αναγνωρίσει κάποια λεκτική μονάδα.

Επιπλέον Λειτουργίες του Λεκτικού Αναλυτή

Συνήθως ο λεκτικός αναλυτής κάνει και μερικές έξτρα λειτουργίες που αφορούν την είσοδο όπως:

- αφαίρεση σχολίων
- αφαίρεση κενών, νέων γραμμών και tabs
- συσχέτιση λεκτικών μονάδων και λαθών με αριθμό γραμμής
- υλοποίηση preprocessing macros

Διαχωρισμός Λεκτικής και Συντακτικής Ανάλυσης

Για πιο λόγο διαχωρίζουμε την λεκτική και την συντακτική ανάλυση;

1. απλοποίηση
2. πιο αποδοτικός μεταγλωττιστής
3. μεταφερτότητα (portability), ειδικά σύμβολα και παραξενιές γλώσσας παραμένουν μόνο στον λεξικό αναλυτή

Λεκτικές Μονάδες, Περιγραφές και Λεξήματα

- Ο λεκτικός αναλυτής επιστρέφει **λεκτικές μονάδες** (tokens).
- Κάθε λεκτική μονάδα αντιστοιχεί σε ένα σύνολο από συμβολοσειρές. Το σύνολο αυτό περιγράφεται από έναν κανόνα που ονομάζεται *περιγραφή* (pattern).
- Ένα *λέξημα* (lexeme) είναι μια συμβολοσειρά για την οποία ισχύει μια περιγραφή για μια λεκτική μονάδα.

Λεκτικές Μονάδες, Περιγραφές και Λεξήματα

Token	Sample Lexeme	Informal Pattern Description
const	const	const
if	if	if
relation	<, <=, −, <>, >, >=	< ή <= ή = ή <> ή >= ή >
id	pi, count, d2	γράμμα ακολουθούμενο από γράμμα ή ψηφία
num	3.14, 0, 6.02e23	οποιαδήποτε αριθμητική σταθερά
literal	”core dumped”	οποιοσδήποτε χαρακτήρας μεταξύ ” και ” εκτός από ”

Οι λεκτικές μονάδες είναι τα τερματικά σύμβολα της γραμματικής του προγράμματος.

Λεκτικές Μονάδες και Γλώσσες Προγραμματισμού

Στις περισσότερες γλώσσες προγραμματισμού τα παρακάτω είναι λεκτικές μονάδες:

- λέξεις κλειδιά (keywords)
- τελεστές (operators)
- αναγνωριστικά (identifiers)
- σταθερές (constants)
- συμβολοσειρές (string literals)
- παρενθέσεις, κόμμα, κ.τ.λ. (punctuation symbols)

Λεξήματα και Χαρακτηριστικά Γνωρίσματα

Επειδή μια λεκτική μονάδα μπορεί να είναι οποιοδήποτε λέξημα που ακολουθεί την περιγραφή της λεκτικής μονάδας, χρειαζόμαστε έναν τρόπο να ξέρουμε σε τι πραγματικά αντιστοιχεί.

π.χ

η περιγραφή αριθμός **num** ισχύει και για την συμβολοσειρά 111 και για την 222.

Για αυτό τον λόγο αντιστοιχούμε συνήθως την λεκτική μονάδα και με μια τιμή χαρακτηριστικού γνωρίσματος (attribute value).

Λεξήματα και Χαρακτηριστικά Γνωρίσματα

Καθώς αναγνωρίζουμε λεκτικές μονάδες προσθέτουμε στοιχεία σε μια δομή που ονομάζεται πίνακας συμβόλων και επιστρέφουμε ζευγάρια λεξημάτων και δεικτών σε στοιχεία του πίνακα συμβόλων.

```
E = M * C ** 2
```

η παραπάνω πρόταση σε Fortran θα γινόταν μετά τον λεκτικό αναλυτή

(`id`, pointer to symbol-table for `E`)
(`assign_op`,)
(`id`, pointer to symbol-table for `M`)
(`mult_op`,)
(`id`, pointer to symbol-table for `C`)
(`exp_op`,)
(`num`, integer value 2)

Επιπλέον πληροφορία μπορεί να είναι η γραμμή εμφάνισης ώστε να βοηθήσει στην εκτύπωση φιλικών μηνυμάτων λαθών.

Συμβολοσειρές

Αλφάβητο

Ένα *αλφάβητο* (alphabet) είναι ένα πεπερασμένο σύνολο συμβόλων.
π.χ το {0, 1} είναι το δυαδικό αλφάβητο, ενώ το ASCII είναι ένα κλασσικό αλφάβητο στους υπολογιστές.

Συμβολοσειρά

Μια *συμβολοσειρά* (string) πάνω σε ένα αλφάβητο είναι μια πεπερασμένη ακολουθία συμβόλων από το αλφάβητο.

Συμβολίζουμε με ϵ την κενή συμβολοσειρά.

Για μια συμβολοσειρά *s* συμβολίζουμε το μήκος της με $|s|$. π.χ $|banana| = 6$

Πράξεις σε Συμβολοσειρές

παράθεση (concatenation)

Εαν *x* και *y* είναι συμβολοσειρές, τότε η *παράθεση* τους *xy* είναι η συμβολοσειρά που προκύπτει προσθέτοντας την *y* στο τέλος της *x*.

π.χ αν x=foo και y=bar τότε xy=foobar

Πράξεις σε Συμβολοσειρές

παράθεση (concatenation)

Εαν x και y είναι συμβολοσειρές, τότε η *παράθεση* τους xy είναι η συμβολοσειρά που προκύπτει προσθέτοντας την y στο τέλος της x.

π.χ αν x=foo και y=bar τότε xy=foobar

Εάν φανταστούμε την παράθεση ως πολλαπλασιασμό, μπορούμε να ορίσουμε την ύψωση σε δύναμη.

Δυνάμεις

$$s^i = \begin{cases} \epsilon & \text{αν } i = 0, \\ s^{i-1}s & \text{αν } i > 0. \end{cases}$$

και άρα $s^0 = \epsilon$, $s^1 = s$, $s^2 = ss$, $s^3 = sss$, ...

Ορισμοί

πρόθεμα (prefix)

Ένα **πρόθεμα** μιας συμβολοσειράς s προκύπτει αφαιρώντας μηδέν ή περισσότερα σύμβολα από το τέλος της s.

π.χ η συμβολοσειρά ban είναι πρόθεμα της banana

επίθεμα (suffix)

Ένα **επίθεμα** μιας συμβολοσειράς s προκύπτει αφαιρώντας μηδέν ή περισσότερα σύμβολα από την αρχή της s.

π.χ η συμβολοσειρά nana είναι επίθεμα της banana

Ορισμοί

υποσυμβολοσειρά (substring)

Μια **υποσυμβολοσειρά** μιας συμβολοσειράς s προκύπτει αφαιρώντας ένα πρόθεμα και ένα επίθεμα από την s.

γνήσιο πρόθεμα, επίθεμα ή υποσυμβολοσειρά

Έστω x ένα πρόθεμα, επίθεμα ή υποσυμβολοσειρά μιας συμβολοσειράς s. Ονομάζετε γνήσιο(α) σε περίπτωση που

- $x \neq \epsilon$ και
- $x \neq s$.

Ορισμοί

υποακολουθία (subsequence)

Οποιαδήποτε συμβολοσειρά προκύπτει σβήνοντας μηδέν ή περισσότερα σύμβολα, όχι υποχρεωτικά συνεχόμενα, από την συμβολοσειρά *s*.

π.χ η συμβολοσειρά baaa είναι υποακολουθία της banana

Γλώσσες

Γλώσσα

Οποιοδήποτε αριθμήσιμο σύνολο συμβολοσειρών ενός αλφαβήτου.

παραδείγματα

- $\{\}$
- $\{\epsilon\}$
- $\{foo, bar\}$
- Όλα τα συντακτικά σωστά προγράμματα Java
- Όλες οι γραμματικά σωστές προτάσεις αγγλικών

Γλώσσες

- ένωση γλωσσών (union)

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$$

Γλώσσες

- ένωση γλωσσών (union)

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$$

- παράθεση γλωσσών (concatenation)

$$L_1 L_2 = \{st \mid s \in L_1 \wedge t \in L_2\}$$

Γλώσσες

- ένωση γλωσσών (union)

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$$

- παράθεση γλωσσών (concatenation)

$$L_1 L_2 = \{st \mid s \in L_1 \wedge t \in L_2\}$$

- ”δυνάμεις”

$$L^i = \begin{cases} \{\epsilon\} & \text{αν } i = 0, \\ L^{i-1} L & \text{αν } i > 0. \end{cases}$$

Γλώσσες

- ένωση γλωσσών (union)

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$$

- παράθεση γλωσσών (concatenation)

$$L_1 L_2 = \{st \mid s \in L_1 \wedge t \in L_2\}$$

- ”δυνάμεις”

$$L^i = \begin{cases} \{\epsilon\} & \text{αν } i = 0, \\ L^{i-1} L & \text{αν } i > 0. \end{cases}$$

- Κλειστότητα ή άστρο του Kleene (Kleene closure)

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Μηδέν ή περισσότερες παραθέσεις της γλώσσας L.

Γλώσσες

- ένωση γλωσσών (union)

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$$

- παράθεση γλωσσών (concatenation)

$$L_1 L_2 = \{st \mid s \in L_1 \wedge t \in L_2\}$$

- ”δυνάμεις”

$$L^i = \begin{cases} \{\epsilon\} & \text{αν } i = 0, \\ L^{i-1} L & \text{αν } i > 0. \end{cases}$$

- Κλειστότητα ή άστρο του Kleene (Kleene closure)

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Μηδέν ή περισσότερες παραθέσεις της γλώσσας L.

- Θετική κλειστότητα (positive closure)

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Μία ή περισσότερες παραθέσεις της γλώσσας L, δηλαδή $L^+ = LL^*$.

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων
- LD είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από ένα γράμμα ακολουθούμενο από ένα ψηφίο

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων
- LD είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από ένα γράμμα ακολουθούμενο από ένα ψηφίο
- L^4 είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από 4 γράμματα

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων
- LD είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από ένα γράμμα ακολουθούμενο από ένα ψηφίο
- L^4 είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από 4 γράμματα
- L^* είναι το σύνολο όλων των συμβολοσειρών από γράμματα, συμπεριλαμβανομένου και της κενής συμβολοσειράς ϵ

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων
- LD είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από ένα γράμμα ακολουθούμενο από ένα ψηφίο
- L^4 είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από 4 γράμματα
- L^* είναι το σύνολο όλων των συμβολοσειρών από γράμματα, συμπεριλαμβανομένου και της κενής συμβολοσειράς ϵ
- $L(L \cup D)^*$ είναι το σύνολο όλων των συμβολοσειρών από γράμματα και ψηφία που ξεκινούν από γράμμα

Παραδείγματα με Γλώσσες

Έστω $L = \{A, B, \ldots, Z, a, b, \ldots, z\}$ και $D = \{0, 1, \ldots, 9\}$

- $L \cup D$ είναι το σύνολο όλων των γραμμάτων και ψηφίων
- LD είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από ένα γράμμα ακολουθούμενο από ένα ψηφίο
- L^4 είναι το σύνολο όλων των συμβολοσειρών που αποτελούνται από 4 γράμματα
- L^* είναι το σύνολο όλων των συμβολοσειρών από γράμματα, συμπεριλαμβανομένου και της κενής συμβολοσειράς ϵ
- $L(L \cup D)^*$ είναι το σύνολο όλων των συμβολοσειρών από γράμματα και ψηφία που ξεκινούν από γράμμα
- D^+ είναι το σύνολο όλων των συμβολοσειρών από ένα ή περισσότερα ψηφία

Κανονικές Εκφράσεις και Γλώσσες

Οι κανονικές εκφράσεις είναι ένας τρόπος αναπαράστασης των γλωσσών που ονομάζονται *κανονικές γλώσσες*.

Έστω ένα αλφάβητο Σ.

Οι παρακάτω κανόνες ορίζουν κανονικές εκφράσεις επάνω στο Σ.

1. ϵ είναι κανονική έκφραση για το σύνολο $\{\epsilon\}$
2. εάν $a \in \Sigma$, τότε a είναι η κανονική έκφραση για το σύνολο $\{a\}$.
3. αν r και s είναι κανονικές εκφράσεις για τις γλώσσες R και S τότε
   1. $(r)|(s)$ είναι κανονική έκφραση της γλώσσας $R \cup S$
   2. $(r)(s)$ είναι κανονική έκφραση της γλώσσας RS
   3. $(r)^*$ είναι κανονική έκφραση της γλώσσας R^*
   4. (r) είναι κανονική έκφραση της γλώσσας R

Κανονικές Εκφράσεις και Γλώσσες

Ο ορισμός των κανονικών εκφράσεων είναι αναδρομικός.

Για να μην χρησιμοποιούμε συνέχεια παρενθέσεις μπορούμε να υποθέσουμε τις εξής προτεραιότητες:

1. ο μοναδιαίος τελεστής * έχει την μεγαλύτερη προτεραιότητα και είναι αριστερά προσεταιριστικός
2. η παράθεση έχει την δεύτερη υψηλότερη προτεραιότητα και είναι αριστερά προσεταιριστική
3. η πράξη $|$ έχει την χαμηλότερη προτεραίοτητα και είναι αριστερά προσεταιριστική

π.χ αντί να γράψουμε $(a)|((b)^*(c))$ μπορούμε πλέον να γράψουμε $a|b^*c$, δηλαδή το σύνολο των συμβολοσειρών που είναι είτε ένα a είτε μηδέν ή περισσότερα b ακολουθούμενα από ένα c.

Παραδείγματα

Έστω $\Sigma = \{a, b\}$.

1. η κανονική έκφραση $a|b$ αναπαριστά το σύνολο $\{a, b\}$
2. η κανονική έκφραση $(a|b)(a|b)$ αναπαριστά το σύνολο $\{aa, ab, ba, bb\}$
3. η κανονική έκφραση $a*$ αναπαριστά το σύνολο $\{\epsilon, a, aa, aaa, \cdots\}$
4. η κανονική έκφραση $(a|b)*$ αναπαριστά το σύνολο συμβολοσειρών που περιέχουν μηδέν ή περισσότερα a ή b

Εάν δύο κανονικές εκφράσεις r και s αναπαριστούν την ίδια γλώσσα λέμε πως είναι ισοδύναμες και γράφουμε $r = s$.

π.χ $(a|b) = (b|a)$

Αλγεβρικές Ιδιότητες Κανονικών Εκφράσεων

$r\|s = s\|r$	η $\|$ είναι αντιμεταθετική
$r\|(s\|t) = (r\|s)\|t$	η $\|$ είναι προσεταιριστική
$(rs)t = r(st)$	η παράθεση είναι προσεταιριστική
$r(s\|t) = rs\|rt$ $(s\|t)r = sr\|tr$	επιμεριστική ιδιότητα παράθεσης ως προς την $\|$
$\epsilon r = r$ $r\epsilon = r$	η ϵ είναι το ουδέτερο στοιχείο ως προς την παράθεση
$r^* = (r\|\epsilon)^*$	η ϵ είναι υποχρεωτικά στην κλειστότητα
$r^{**} = r^*$	η $*$ είναι αυτοπαθητική

Κανονικοί Ορισμοί

Για ευκολία θέλουμε να δίνουμε ονόματα σε κανονικές εκφράσεις και να χρησιμοποιούμε αυτά τα ονόματα ως σύμβολα ώστε να φτιάχνουμε πιο σύνθετες κανονικές εκφράσεις.

Κανονικοί Ορισμοί

Για ευκολία θέλουμε να δίνουμε ονόματα σε κανονικές εκφράσεις και να χρησιμοποιούμε αυτά τα ονόματα ως σύμβολα ώστε να φτιάχνουμε πιο σύνθετες κανονικές εκφράσεις.

Εάν Σ είναι ένα αλφάβητο από βασικά σύμβολα, ένας *κανονικός ορισμός* είναι μια σειρά από ορισμούς της μορφής

$$
\begin{aligned}
d_1 &\to r_1 \\
d_2 &\to r_2 \\
&\dots \\
d_n &\to r_n
\end{aligned}
$$

όπου κάθε d_i είναι ένα ξεχωριστό όνομα και κάθε r_i είναι μια κανονική έκφραση με τα σύμβολα $\Sigma \cup \{d_1, \dots, d_{i-1}\}$.

Για να εκφράσουμε προσδιοριστικά τα οποία είναι συμβολοσειρές που περιέχουν γράμματα και ψηφία και ξεκινούν με γράμμα

`letter` → `A` | `B` | ⋯ | `Z` | `a` | `b` | ⋯ | `z`

`digit` → 0 | 1 | ⋯ | 9

`id` → `letter` (`letter` | `digit`)*

Για να εκφράσουμε μη προσημασμένους αριθμούς της μορφής:

- 5280
- 39.37
- $6.33e4$
- $1.894e-4$

$$\begin{aligned}
\texttt{digit} &\rightarrow 0 \mid 1 \mid \cdots \mid 9 \\
\texttt{digits} &\rightarrow \texttt{digit digit}^* \\
\texttt{optional_fraction} &\rightarrow \texttt{.\ digits} \mid \epsilon \\
\texttt{optional_exponent} &\rightarrow \texttt{(\ e\ (\ +} \mid - \mid \epsilon \texttt{\ )\ digits\ )} \mid \epsilon \\
\texttt{num} &\rightarrow \texttt{digits optional_fraction optional_exponent}
\end{aligned}$$

Συντομεύσεις

Μερικές κατασκευές προκύπτουν τόσο συχνά που έχει νόημα να δημιουργήσουμε συντομεύσεις:

1. *Ένα ή περισσότερα*
   Αν r είναι μια κανονική έκφραση που περιγράφει την γλώσσα $L(r)$ τότε r^+ περιγράφει την γλώσσα $L(r)^+$
2. *Μηδέν ή μια φορά*
   Η έκφραση $r?$ είναι συντόμευση για $r|\epsilon$
3. Κλάσεις χαρακτήρων
   Η έκφραση $[abc]$ όπου a, b, c είναι σύμβολα του αλφαβήτου περιγράφουν την κανονική έκφραση $a|b|c$
   Η έκφραση $[a-z]$ περιγράφει την κανονική έκφραση $a|b|\cdots|z$

   Με τον τρόπο αυτό τα προσδιοριστικά (identifiers) μπορούν να περιγραφούν ως
   $[A-Za-z][A-Za-z0-9]*$

Μη Κανονικές Γλώσσες

Δεν μπορούμε να περιγράψουμε όλες τις γλώσσες με κανονικές εκφράσεις.

π.χ η γλώσσα

$$L = \{a^n b^n\}$$

όλων των συμβολοσειρών που περιέχουν έναν αριθμό από a που ακολουθείται από τον ίδιο αριθμό από b δεν μπορεί να περιγραφεί από μια κανονική έκφραση.

Παράδειγμα

Αριθμητικές σταθερές χωρίς πρόσημο στην C.

- ακέραιο μέρος που δεν αρχίζει με μηδέν εκτός αν είναι μηδέν
- προαιρετικά υποδιαστολή και κλασματικό μέρος
- προαιρετικα εκθέτης με ή χωρίς πρόσημο

$$([1-9][0-9]^*|0)(.[0-9]^+)?((E|e)(+|-)?[0-9]^+)?$$

Προσοχή, στις περισσότερες υλοποιήσεις των κανονικών εκφράσεων (regular expressions) η τελεία είναι ειδικός χαρακτήρας που συμβολίζει οποιοδήποτε χαρακτήρα. Σε τέτοια περίπτωση πρέπει να κάνουμε escape με \.

Αναγνώριση Γλωσσών

Λέμε πως ένα πρόγραμμα *αναγνωρίζει* (recognizes) μια γλώσσα L εαν όταν πέρνει ως είσοδο μια συμβολοσειρά $x \in L$ επιστρέφει ”ναι” και αν $x \notin L$ επιστρέφει ”όχι”.

Μη-ντετερμινιστικά Πεπερασμένα Αυτόματα

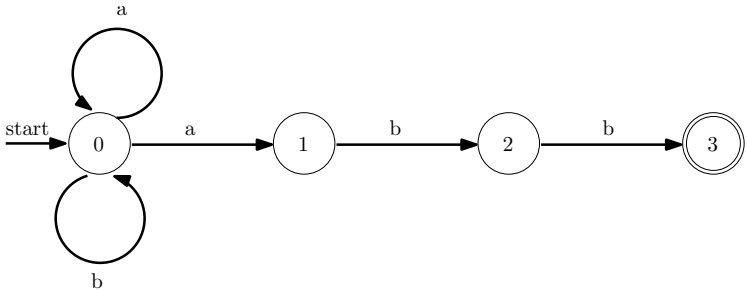
Non-deterministic Finite Automata

Ένα μη-ντετερμινιστικό πεπερασμένο αυτόματο (NFA) είναι ένα μαθηματικό μοντέλο με

1. ένα σύνολο καταστάσεων S (states)
2. ένα σύνολο συμβόλων εισόδου Σ (input symbol alphabet)
3. μια συνάρτηση μετάβασης που αντιστοιχεί ζευγάρια κατάστασης-συμβόλου σε σύνολα από καταστάσεις (transition function)
4. μια κατάσταση s_0 που ονομάζεται αρχική κατάσταση (start state)
5. ένα σύνολο καταστάσεων F που θεωρούνται καταστάσεις αποδοχής (accepting states)

Ένα NFA μπορεί να περιγραφεί σχηματικά ως ένα κατευθυνόμενο γράφημα μετάβασης με ετικέτες (labels) στις ακμές.

π.χ το γράφημα μετάβασης ενός NFA για την γλώσσα $(a|b)^*abb$ φαίνεται παρακάτω:



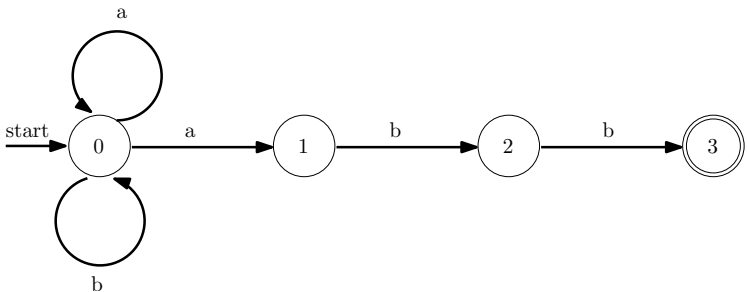
Ένα NFA *δέχεται* (accepts) μια συμβολοσειρά εισόδου x αν και μόνο αν υπάρχει ένα μονοπάτι από την αρχική κατάσταση σε μια κατάσταση αποδοχής, τέτοιο ώστε οι ετικετών κατά μήκος των ακμών του μονοπατιού να δίνουν της συμβολοσειρά x.

Μη-ντετερμινιστικά Πεπερασμένα Αυτόματα

Γλώσσα Αυτόματου

Ένα NFA *δέχεται* (accepts) μια συμβολοσειρά εισόδου x αν και μόνο αν υπάρχει ένα μονοπάτι από την αρχική κατάσταση σε μια κατάσταση αποδοχής, τέτοιο ώστε οι ετικετών κατά μήκος των ακμών του μονοπατιού να δίνουν της συμβολοσειρά x.

π.χ το γράφημα μετάβασης ενός NFA για την γλώσσα $(a|b)^*abb$ φαίνεται παρακάτω:

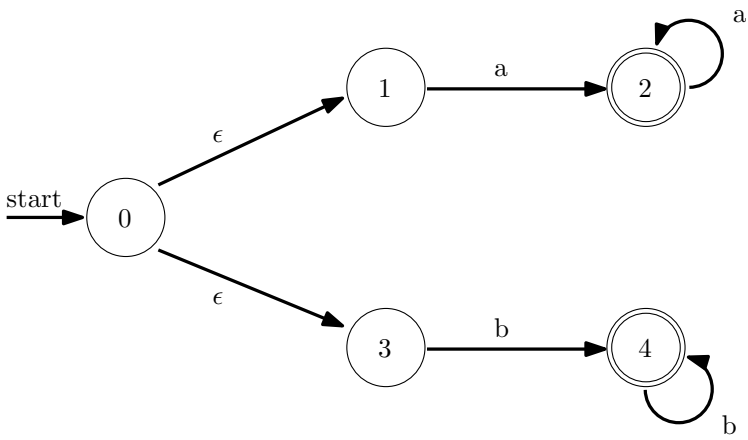


Ένα αυτόματα ορίζει μια γλώσσα, το σύνολο των συμβολοσειρών που δέχεται το αυτόματο.

Μη-ντετερμινιστικά Πεπερασμένα Αυτόματα

Παράδειγμα

NFA που δέχεται την γλώσσα $aa^*|bb^*$



Ντετερμινιστικά Πεπερασμένα Αυτόματα

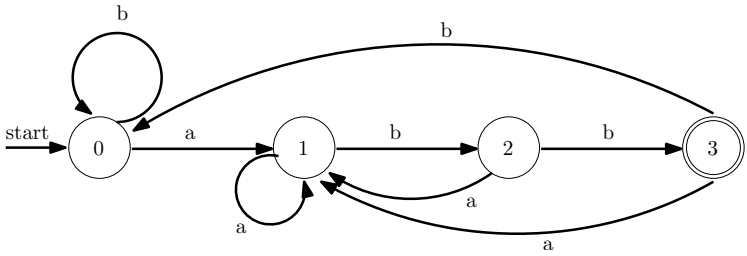
Deterministic Finite Automata

Ένα ντετερμινιστικό πεπερασμένο αυτόματο (DFA) είναι μια ειδική περίπτωση ενός μη-ντετερμινιστικού πεπερασμένου αυτόματου όπου

1. δεν υπάρχουν ϵ-μεταβάσεις, δηλαδή στο γράφημα μετάβασης δεν υπάρχουν ακμές με ετικέτα ϵ
2. για κάθε κατάσταση s και σύμβολο εισόδου a, υπάρχει το πολύ μια εξερχόμενη ακμή από τον κόμβο s με ετικέτα a

Είναι πολύ εύκολο να ελέγξουμε αν ένα DFA δέχεται μία συμβολοσειρά x μιας και το μονοπάτι που πρέπει να ακολουθήσουμε είναι μοναδικό.

π.χ το γράφημα μετάβασης ενός DFA για την γλώσσα $(a|b)^*abb$ φαίνεται παρακάτω:



- Έχοντας ένα NFA μπορούμε να κατασκευάσουμε ένα DFA που αναγνωρίζει την ίδια γλώσσα (subset construction algorithm).
- Με αυτό τον τρόπο μπορούμε να υλοποιήσουμε ένα NFA στον υπολογιστή.

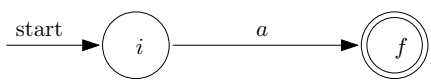
- Έχοντας ένα NFA μπορούμε να κατασκευάσουμε ένα DFA που αναγνωρίζει την ίδια γλώσσα (subset construction algorithm).
- Με αυτό τον τρόπο μπορούμε να υλοποιήσουμε ένα NFA στον υπολογιστή.
- Η ιδέα είναι να φτιάξουμε ένα καινούριο DFA όπου κάθε κατάσταση του να αντιστοιχεί σε ένα σύνολο καταστάσεων του αρχικού NFA.
- Το μέγεθος του DFA αυτού μπορεί να είναι εκθετικό σε σχέση με το μέγεθος του NFA. Συνήθως όμως στην πράξη δεν παρουσιάζεται η χειρότερη περίπτωση.
- Έχοντας ως είσοδο μια κανονική έκφραση r μπορούμε να κατασκευάσουμε ένα ισοδύναμο NFA (Thompson's construction).

Μετατροπή Κανονικών Εκφράσεων σε NFA

Σπάμε μια κανονική έκφραση *r* σε μικρότερα μέρη και κατασκευάζουμε επιμέρους NFA.

- Για την έκφραση ϵ φτιάχνουμε το NFA

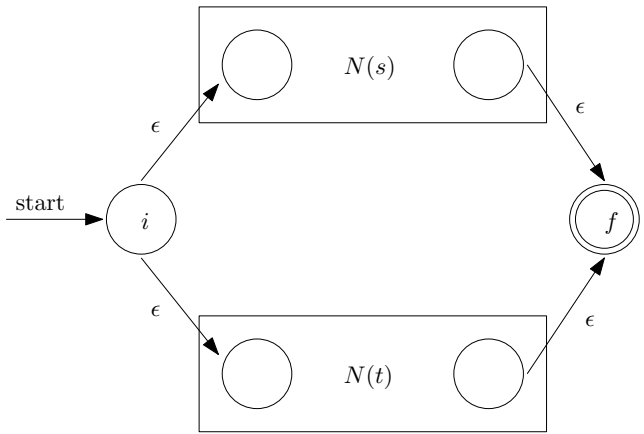
- Για την έκφραση $a \in \Sigma$ φτιάχνουμε το NFA



Μετατροπή Κανονικών Εκφράσεων σε NFA

Σπάμε μια κανονική έκφραση r σε μικρότερα μέρη και κατασκευάζουμε επιμέρους NFA.

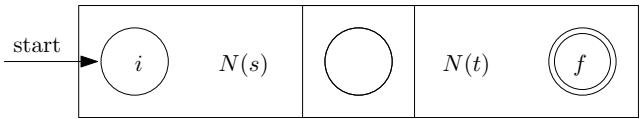
- Για την έκφραση $r = s|t$ κατασκευάζουμε τα NFAs $N(s)$ και $N(t)$ για τις κανονικές εκφράσεις s και t και τα συνδέουμε ως



Μετατροπή Κανονικών Εκφράσεων σε NFA

Σπάμε μια κανονική έκφραση r σε μικρότερα μέρη και κατασκευάζουμε επιμέρους NFA.

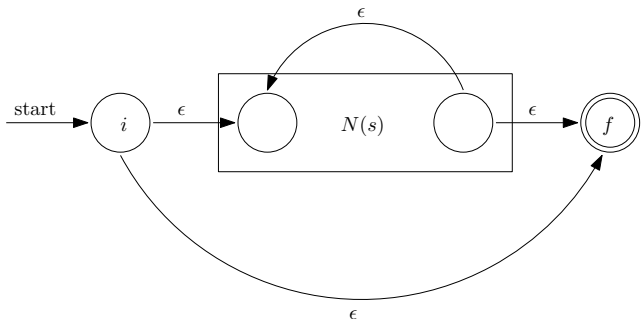
- Για την έκφραση $r = st$ κατασκευάζουμε τα NFAs $N(s)$ και $N(t)$ για τις κανονικές εκφράσεις s και t και τα συνδέουμε ως



Μετατροπή Κανονικών Εκφράσεων σε NFA

Σπάμε μια κανονική έκφραση r σε μικρότερα μέρη και κατασκευάζουμε επιμέρους NFA.

- Για την έκφραση $r = s^*$ κατασκευάζουμε το NFA $N(s)$ ως



Μετατροπή Κανονικών Εκφράσεων σε NFA

Άσκηση

Φτιάξτε ένα μη-ντετερμινιστικό αυτόματο (NFA) για την κανονική έκφραση $r = (a|b)^* abb$.

Κεντρική Ιδέα

Κάθε κατάσταση του DFA αντιστοιχεί σε ένα σύνολο καταστάσεων του NFA.

Διαβάζοντας την είσοδο $a_1 a_2 \cdots a_n$ το DFA είναι σε μια κατάσταση που αντιστοιχεί στο σύνολο των καταστάσεων που θα μπορούσε να βρίσκεται το NFA.

Ο αριθμός των καταστάσεων του DFA μπορεί να είναι εκθετικός ως προς τον αριθμό καταστάσεων του NFA. Στην περίπτωση της λεκτικής ανάλυσης όμως, το DFA έχει συνήθως το ίδιο περίπου μέγεθος με το NFA.

Μετατροπή NFA σε DFA

Subset Construction Algorithm

$\epsilon - closure(s)$	Σύνολο καταστάσεων του NFA που είναι προσβάσιμες από την κατάσταση s μόνο με ϵ-μεταβάσεις.
$\epsilon - closure(T)$	Σύνολο καταστάσεων του NFA που είναι προσβάσιμες από την κατάσταση $s \in T$ μόνο με ϵ-μεταβάσεις.
$move(T, a)$	Σύνολο καταστάσεων του NFA που υπάρχει μετάβαση μέσω του συμβόλου a από μια κατάσταση $s \in T$.

Αρχική κατάσταση

Πριν διαβάσει κάποιο σύμβολο το NFA μπορεί να βρίσκεται στις καταστάσεις $\epsilon - closure(s_0)$ όπου s_0 είναι η αρχική κατάσταση. Αρχική κατάσταση του DFA είναι το $\epsilon - closure(s_0)$.

Αναγωγή

Έστω πως το NFA μπορεί να είναι στο σύνολο καταστάσεων T. Εαν διαβάσουμε το επόμενο σύμβολο a, τότε μπορεί να πάει στις καταστάσεις $move(T, a)$. Μετά όμως μπορεί να κάνει και ϵ-μετάβαση και άρα μπορεί να πάει στις καταστάσεις $\epsilon - closure(move(T, a))$.

Καταστάσεις Αποδοχής

Καταστάσεις αποδοχής του DFA είναι όλες οι καταστάσεις που περιέχουν τουλάχιστον μία κατάσταση αποδοχής του NFA.

Εργαλεία Παραγωγής Λεκτικών Αναλυτών

Η ύπαρξη όλων αυτών των θεωρητικών αποτελεσμάτων μας επιτρέπουν να κατασκευάσουμε εργαλεία που να αυτοματοποιούν την λειτουργία κατασκευής λεκτικών αναλυτών.

Το πρόγραμμα *lex* είναι ένα πρόγραμμα παραγωγής λεκτικών αναλυτών (lexer ή scanner).

- `http://en.wikipedia.org/wiki/Lex_(software)`
- `http://dinosaur.compilertools.net`
- `http://flex.sourceforge.net`
- `http://www.jflex.de`

Lex

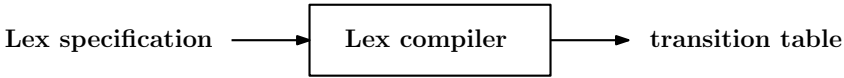
Δέχεται ως είσοδο μια περιγραφή του λεκτικού αναλυτή στην μορφή

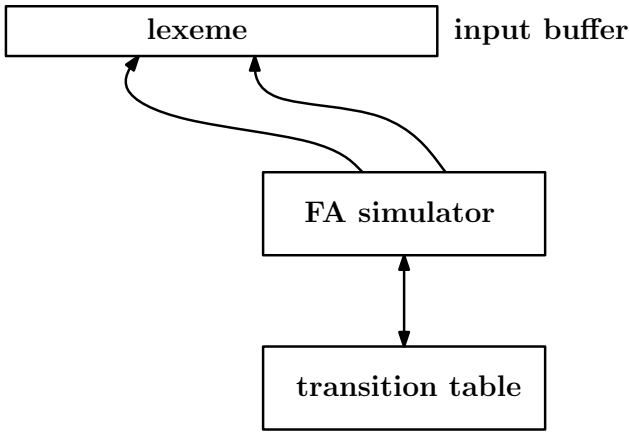
$$
\begin{array}{ll}
p_1 & \{action_1\} \\
p_1 & \{action_2\} \\
\vdots & \vdots \\
p_n & \{action_n\}
\end{array}
$$

όπου κάθε p_i είναι μια κανονική έκφραση και κάθε $action_i$ είναι ένα κομμάτι προγράμματος που πρέπει να εκτελεστεί όταν ένα λέξημα που αντιστοιχεί στο p_i βρίσκεται στην είσοδο.

Εαν έχουμε πάνω από μια αναγνώριση τότε το μακρύτερο λέξημα επιλέγεται. Αν υπάρχουν δύο κανόνες που αναγνωρίζουν το μακρύτερο λέξημα, ο πρώτος εκτελείται.

Lex


Lex specification
Lex compiler
transition table


lexeme
input buffer
FA simulator
transition table

Υλοποίηση Συντακτικού Αναλυτή με Lex

Για τους σκοπούς του μαθήματος θα χρησιμοποιήσουμε Java και άρα θα μιλήσουμε για το εργαλείο JFlex που μπορείτε να κατεβάσετε από την διεύθυνση

- `http://jflex.de`

Ανεξάρτητα από την γλώσσα προγραμματισμού τα εργαλεία τύπου Lex λειτουργούν με τον ίδιο τρόπο.

Μορφή Προγράμματος JFLex

```
User code
%%
Options and declarations
%%
Rules and actions
```

Το πρώτο μέρος περιέχει κώδικα που αντιγράφεται αυτούσιος πριν από τον κώδικα που παράγει αυτόματα ο JFlex.

Με αυτό τον τρόπο μπορούμε να κάνουμε `import` βιβλιοθήκες, κ.τ.λ

```
1  /* HUA – Compilers Course Lexer Example */
2
3  import static java.lang.System.out;
4
5  %%
```

Στο δεύτερο μέρος μπορούμε

- να καθορίσουμε την λειτουργία του αναλυτή που θα δημιουργηθεί μέσω διαφόρων επιλογών
- να δηλώσουμε καταστάσεις (states) και να ορίσουμε macros

Μερικές επιλογές του JFlex είναι

`%class Lexer`	ονομάζει την κλάση που παράγει ως Lexer
`%unicode`	χρησιμοποιεί unicode για την κωδικοποίηση των χαρακτήρων
`%public`	η παραγώμενη κλάση είναι public
`%line`	ενεργοποιεί την μέτρηση γραμμών (yyline)
`%standalone`	δημιουργεί και main συνάρτηση
`%{`	
`...`	αντιγράφει αυτούσιο κώδικα μέσα στην κλάση
`%}`	

Για περισσότερα διαβάστε το manual την ιστοσελίδα του JFlex.

Υλοποιήσεις Κανονικών Εκφράσεων

Στις περισσότερες υλοποιήσεις κανονικών εκφράσεων υπάρχουν μερικές διαφοροποιήσεις σε σχέση με την θεωρία:

a	ο χαρακτήρας *a*
.	οποιοσδήποτε χαρακτήρας εκτός από την αλλαγή γραμμής
$\backslash x$	αν *x* ένα από *a*, *b*, *f*, *n*, *t*, *r*, *v* ή 0 τότε όπως στην γλώσσα C, αλλιώς ο ίδιος ο χαρακτήρας *x*
$\backslash 123$	ο χαρακτήρας ASCII με οκταδική τιμή 123
$\backslash x3f$	ο χαρακτήρας ASCII με δεκαεξαδική τιμή 3F
”*abc*”	συμβολοσειρά abc
[*abc*]	ένας από τους χαρακτήρες *a*, *b*, *c*
[*a* − *z*]	ένας από τους χαρακτήρες *a* έως *z*
[*af* − *kz*]	ένας από τους χαρακτήρες *a*, *f* έως *k*, *z*
[^*a* − *z*]	ένας από τους χαρακτήρες εκτός όσων ανήκουν στην περιοχή *a* έως *z*
{*name*}	η κανονική έκφραση με όνομα name
rs	παράθεση των κανονικών εκφράσεων *r* και *s*
r|*s*	διάζευξη των κανονικών εκφράσεων *r* και *s*
(*r*)	κανονική έκφραση *r*

Υλοποιήσεις Κανονικών Εκφράσεων

Στις περισσότερες υλοποιήσεις κανονικών εκφράσεων υπάρχουν μερικές διαφοροποιήσεις σε σχέση με την θεωρία:

$r*$	η r επαναλαμβάνεται μηδέν ή περισσότερες φορές
$r+$	η r επαναλαμβάνεται μία ή περισσότερες φορές
$r?$	η r επαναλαμβάνεται μηδέν ή μια φορά (προαιρετική)
$r\{3\}$	η r επαναλαμβάνεται ακριβώς 3 φορές
$r\{3,5\}$	η r επαναλαμβάνεται από 3 έως 5 φορές
$r\{4,\}$	η r επαναλαμβάνεται 4 ή περισσότερες φορές
$!r$	τα πάντα εκτός από την r
$\tilde{}r$	τα πάντα μέχρι και την πρώτη εμφάνιση (συμπεριλαμβανομένης) της r
$\hat{}r$	η r αλλά μόνο στην αρχή της γραμμής
$r\$$	η r αλλά μόνο στο τέλος της γραμμής
$<< EOF >>$	τέλος του αρχείου εισόδου
r/s	η κανονική έκφραση r αλλά μόνο αν ακολουθεί η κανονική έκφραση s
$< S > r$	η κανονική έκφραση r αλλά μόνο αν η τρέχουσα αρχική κατάσταση είναι η S
$< S_1, S_2, S_3 > r$	η κανονική έκφραση r αλλά μόνο αν η τρέχουσα αρχική κατάσταση είναι μία από τις S_1, S_2, S_3
$<*> r$	η κανονική έκφραση r σε οποιαδήποτε αρχική κατάσταση

Μπορούμε να δηλώσουμε ονόματα για κανονικές εκφράσεις:

```
LineTerminator = \r|\n|\r\n
WhiteSpace     = {LineTerminator} | [ \t\f]

Identifier     = [:jletter:] [:jletterdigit:]*

DecIntegerLiteral = 0 | [1-9][0-9]*
```

και επίσης να ορίσουμε καταστάσεις (states)

```
%state STRING
```

JFlex

2ο Μέρος

```
%class MyLexer
%unicode
%public
%line
%standalone

%{
    StringBuffer string = new StringBuffer();
%}

LineTerminator = \r|\n|\r\n
WhiteSpace     = {LineTerminator} | [ \t\f]

Comment        = "/*" [^*] ~"*/" | "/*" "*"+ "/"

Identifier     = [:jletter:] [:jletterdigit:]*

DecIntegerLiteral = 0 | [1-9][0-9]*

%state STRING

%%
```

Στο τρίτο μέρος μπορούμε

- να δώσουμε κανόνες και λειτουργίες που πρέπει να συμβούν κατά περίπτωση.

JFlex

3ο Μέρος

```
28  <YYINITIAL> "int"    { out.println("INT"); }
29  <YYINITIAL> "char"   { out.println("CHAR"); }
30  <YYINITIAL> "float"  { out.println("FLOAT"); }
31  <YYINITIAL> "return" { out.println("RETURN"); }
32  <YYINITIAL> "break"  { out.println("BREAK"); }
33
34  <YYINITIAL> {
35      {Identifier}                  { out.println("id:" + yytext()); }
36
37      {DecIntegerLiteral}           { out.println("decimal:" + yytext()); }
38
39      \"                            { string.setLength(0); yybegin(STRING); }
40
41      "="                           { out.println("ASSIGN"); }
42      "=="                          { out.println("EQUALS"); }
43      "+"                           { out.println("PLUS"); }
44      "-"                           { out.println("MINUS"); }
45      "("                           { out.println("LEFT_PARENTHESIS"); }
46      ")"                           { out.println("RIGHT_PARENTHESIS"); }
47      ","                           { out.println("COMMA"); }
48      "*"                           { out.println("STAR"); }
49      "["                           { out.println("LEFT_SQUARE_BRACKET"); }
50      "]"                           { out.println("RIGHT_SQUARE_BRACKET"); }
51      "{"                           { out.println("LEFT_CURLY_BRACKET"); }
52      "}"                           { out.println("RIGHT_CURLY_BRACKET"); }
53      ";"                           { out.println("SEMICOLON"); }
54
55      {Comment}                     { /* ignore */ }
56      {WhiteSpace}                  { /* ignore */ }
57  }
```

```
58  <STRING> {
59      \"                              { yybegin(YYINITIAL);
60                                        out.println("string:" + string.toString() );
61                                      }
62
63      [^\n\r\"\\]+                    { string.append(yytext()); }
64      \\t                             { string.append('\t'); }
65      \\n                             { string.append('\n'); }
66
67      \\r                             { string.append('\r'); }
68      \\\"                            { string.append('\"'); }
69      \\                              { string.append('\\'); }
70  }
71
72  [^]                                 { throw new Error("Illegal character <"+
73                                                        yytext()+">"); }
74  }
```

Αντιγράψτε τον κώδικα του παραδείγματος σε ένα αρχείο `simple.flex` και εκτελέστε τον JFlex.

```
Reading "simple.flex"
Constructing NFA : 150 states in NFA
Converting NFA to DFA :
............................................................
63 states before minimization, 53 states in minimized DFA
Writing code to "MyLexer.java"
```

Παράγει πρώτα ένα NFA και στην συνέχεια το μετατρέπει σε DFA.

Ο κώδικας που υλοποιεί αυτό το DFA είναι στο αρχείο MyLexer.java το οποίο μπορούμε να το κάνουμε compile με

```
javac MyLexer.java
```

Εκτελώντας το scanner μας με το παρακάτω κομμάτι κώδικα

```
/*
  This is a comment
*/

int main(int argc, char* argv[]) {

    printf("Hello world!\n");

    return 0;
}
```

Πέρνουμε στην έξοδο

```
INT
id:main
LEFT_PARENTHESIS
INT
id:argc
COMMA
CHAR
STAR
id:argv
LEFT_SQUARE_BRACKET
RIGHT_SQUARE_BRACKET
RIGHT_PARENTHESIS
LEFT_CURLY_BRACKET
id:printf
LEFT_PARENTHESIS
string:Hello world!

RIGHT_PARENTHESIS
SEMICOLON
RETURN
decimal:0
SEMICOLON
RIGHT_CURLY_BRACKET
```