HY255 Εργαστήριο Λογισμικού – L02

Assignment 1 – transliteration

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Assignment 1

- https://www.csd.uoc.gr/~hy255/as1
- Γράψτε ένα πρόγραμμα «translate» που μετατρέπει κείμενο από το char set iso8859-7 (ελληνικοί χαρακτήρες) σε iso8859-1 (αγγλικοί χαρακτήρες).
- Πχ. Αν γράψουμε στο stdin Παράδειγμα το πρόγραμμα θα τυθώσει στο stdout Paradeigma
- Γενική ιδέα: Αντιστοιχίζουμε χαρακτήρες του iso8859-7 σε χαρακτήρες του iso8859-1, π.χ.
  - Π → P
  - α → a
  - Η άσκηση ορίζει πως γίνεται η αντιστοίχιση αυτή. Η επιλογή είναι σχετικά αυθαίρετη, ώστε απλά να μπορεί να διαβάζεται το τελικό κείμενο (και να εμφανίζονται στις αντιστοιχίσεις «ενδιαφέρουσες περιπτώσεις» που χρειαζόμαστε για την άσκηση).
- Το πρόγραμμα διαβάζει το stdin και γράφει το stdout, οπότε πχ. μπορούμε να το εκτελέσουμε ως:
  - $ translate < in.7 > out.7

Στόχος: Think before you code

- Πολλές φορές όταν σχεδιάζουμε συστήματα θεωρούμε ότι αυτό γίνεται καθώς γράφουμε την υλοποίησή τους σε κώδικα
- Γενικά, αυτή η τακτική μας οδηγεί στο να γράφουμε κώδικα «δοκιμαστικά» χωρίς να έχουμε ξεκάθαρη εικόνα του τι θέλουμε να κάνουμε
- Το αποτέλεσμα είναι ένα διαρκές γράφε-σβήνε-άλλαζε με δοκιμές που πολλές φορές δεν τελειώνουν ποτέ (σε συστήματα που είναι σχετικά πολύπλοκα)
- Η άσκηση αυτή είναι εύκολο να εξελιχθεί σε «γράφε-σβήνε» και δυσνόητη ροή στον κώδικα
- Στόχος μας είναι να χωρίσουμε τον σχεδιασμό (design) από την υλοποίηση (implementation) και να μπορούμε να δείξουμε ότι το το πρόγραμμα μας είναι σωστό με βάση το design, ενώ η υλοποίηση είναι μια μηχανική διαδικασία (που μάλιστα σήμερα αυτοματοποιείται για το συγκεκριμένο πρόγραμμα)

ISO8859-x Character Sets

Ένα character set είναι ένας πίνακας που αντιστοιχίζει έναν χαρακτήρα (περιεχόμενο μιας θέσης) σε έναν αριθμό (θέση στον πίνακα) [https://www.csd.uoc.gr/~hy255/as1]

ISO8859-1

Dec	Hex	Octal	Character
…			
32	0020	40	
33	0021	41	!
34	0022	42	"
35	0023	43	#
36	0024	44	$
37	0025	45	%
38	0026	46	&
….			
48	0030	60	0
49	0031	61	1
50	0032	62	2
51	0033	63	3
52	0034	64	4
….			
65	0041	101	A
66	0042	102	B
67	0043	103	C
68	0044	104	D
….			

ISO8859-7

Hex	Dec	Char	ISO/IEC 10646-1:2000 Character Name
….			
20	32		SPACE
21	33	!	EXCLAMATION MARK
22	34	"	QUOTATION MARK
…			
30	48	0	DIGIT ZERO
31	49	1	DIGIT ONE
32	50	2	DIGIT TWO
…			
41	65	A	LATIN CAPITAL LETTER A
42	66	B	LATIN CAPITAL LETTER B
43	67	C	LATIN CAPITAL LETTER C
44	68	D	LATIN CAPITAL LETTER D
…			
61	97	a	LATIN SMALL LETTER A
62	98	b	LATIN SMALL LETTER B
63	99	c	LATIN SMALL LETTER C
B6	182	Ά	GREEK CAPITAL LETTER ALPHA WITH TONOS
…			
D9	217	Ω	GREEK CAPITAL LETTER OMEGA
DA	218	Ϊ	GREEK CAPITAL LETTER IOTA WITH DIALYTIKA
DB	219	Ϋ	GREEK CAPITAL LETTER UPSILON WITH DIALYTIKA
DC	220	ά	GREEK SMALL LETTER ALPHA WITH TONOS
…			
E1	225	α	GREEK SMALL LETTER ALPHA
E2	226	β	GREEK SMALL LETTER BETA
…			
FA	250	ϊ	GREEK SMALL LETTER IOTA WITH DIALYTIKA
FB	251	ϋ	GREEK SMALL LETTER UPSILON WITH DIALYTIKA
…			

Input / Output στη Γλώσσα C

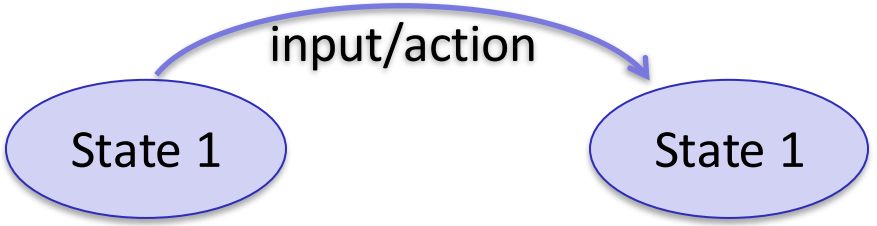
- Στο input / ouput αναφέραμε ότι η γλώσσα C ορίζει τα streams stdin/stdout και δεν ερμηνεύει κάτι περισσότερο. Η ερμηνεία του περιεχομένου είναι θέμα του ”εξωτερικού κόσμου”, που για μας είναι κατά βάση το terminal που τρέχουμε το πρόγραμμα και τα input αρχεία μας.
- Αν στο terminal εκτελέσουμε το πρόγραμμα σαν
  - a.out < in.1 > out.1
  - Τότε το λειτουργικό σύστημα όταν ξεκινά το πρόγραμμά μας, θέτει σαν stdin το in.1 και stdout το out.1
- Αν το in.1 περιέχει χαρακτήρες iso8859-7 το out.1 θα πρέπει να περιέχει χαρακτήρες iso8859-1 με βάση την αντιστοίχιση της άσκησης (και εφόσον το πρόγραμμά μας είναι σωστό).
- Πως μπορούμε να "δούμε” το “σωστό” περιεχόμενο των in.1, out.1;
  - Πχ με $ cat filename
  - Το out.1 είναι εύκολο γιατί περιέχει μόνο χαρακτήρες από τις πρώτες 128 θέσεις του iso8859-1 (λατινικοί χαρακτήρες) και όλα τα terminals “δείχνουν” σωστά τους χαρακτήρες αυτούς γιατί περιέχονται σε όλα τα σετ χαρακτήρων.
  - Το in.1 θα ”φανεί” σωστά (σαν ελληνικοί χαρακτήρες) μόνο αν το terminal μας έχει οριστεί να χρησιμοποιεί το character set iso8859-7. Τα πιο πολλά terminals μπορούν να το κάνουν αυτό με κάποιο τρόπο/setting/παράμετρο.
- Πως μπορούμε να γράψουμε χαρακτήρες του iso8859-7 στο in.1?
  - Με έναν text editor (πχ vi, emacs) που τον τρέχουμε σε ένα terminal το οποίο χρησιμοποιεί το iso8859.7 και επομένως το αρχείο που θα αποθηκεύσει θα περιέχει τους χαρακτήρες στην αντίστοιχη κωδικοποίηση.
  - Με ένα πρόγραμμα C που γράφει στο stdout (και το κάνουμε redirect στο in.1) χαρακτήρες (chars) της γλώσσας C που αντιστοιχούν στα byte values των χαρακτήρων του iso8859-7, πχ putchar(‘\xE2’) για το Ελληνικό μικρό βήτα.
    - Σε character constants (τιμές μέσα σε απλά εισαγωγικά ’A’), μπορούμε να χρησιμοποιήσουμε είτε την δεκαεξαδική αναπαράσταση (π.χ. ‘\x41’) ή την οκταδική (π.χ. ‘\101’), αλλά η γλώσσα δεν επιτρέπει να χρησιμοποιήσουμε δεκαδική αναπαράσταση γιατί δεν ορίζει (η γλώσσα C) αν οι characters αναπαρίστανται σαν signed ή unsigned αριθμοί και επομένως διαφορετικές υλοποιήσεις μπορεί να ακολουθήσουν διαφορετικές αναπαραστάσεις.

Πως δουλεύει το πρόγραμμά μας;

- Στις αντιστοιχίσεις της άσκησης υπάρχουν οι εξής περιπτώσεις:
- 1→1: 1 χαρακτήρας αντιστοιχίζεται σε 1 χαρακτήρα, π.χ. β → b
- 1 → 2: ή 3: 1 χαρακτήρας σε 2 ή 3, π.χ. ά → a’
- 2 → 1: 2 χαρακτήρες σε 1, π.χ. ντ → d
- Αυτές οι αντιστοιχίσεις είναι σχετικά «αυθαίρετες» προκειμένου να δημιουργούν τις περιπτώσεις που θέλουμε.
- Ας σκεφτούμε πρώτα τις δύο πρώτες περιπτώσεις που είναι πιο απλές.
- Ένας απλός πίνακας charmap[] φαίνεται να μας αρκεί:
  - β = iso8859-7[0xE2]
  - b = iso8859-1[0x62]
  - τότε αρκεί charmap[0xE2] = 0x62
- Μάλλον ο πίνακας πρέπει να είναι δισδιάστατος ώστε να συμπεριλάβει και την περίπτωση που ένας χαρακτήρας αντιστοιχίζεται σε 2 ή 3.
  - Αρχικοποιούμε το charmap λοιπόν με βάση τις αντιστοιχίσεις που ορίζει η άσκηση
- Πως δουλεύει λοιπόν το πρόγραμμά μας;
  - Ένα απλό loop που διαβάζει το stdin → c, χρησιμοποιει την τιμή που διάβασε για να κάνει lookup το charmap[c] και γράφει το stdout το περιεχόμενο του charmap[]
- Τι γίνεται με την περίπτωση 3, όπου 2 χαρακτήρες αντιστοιχίζονται σε 1;

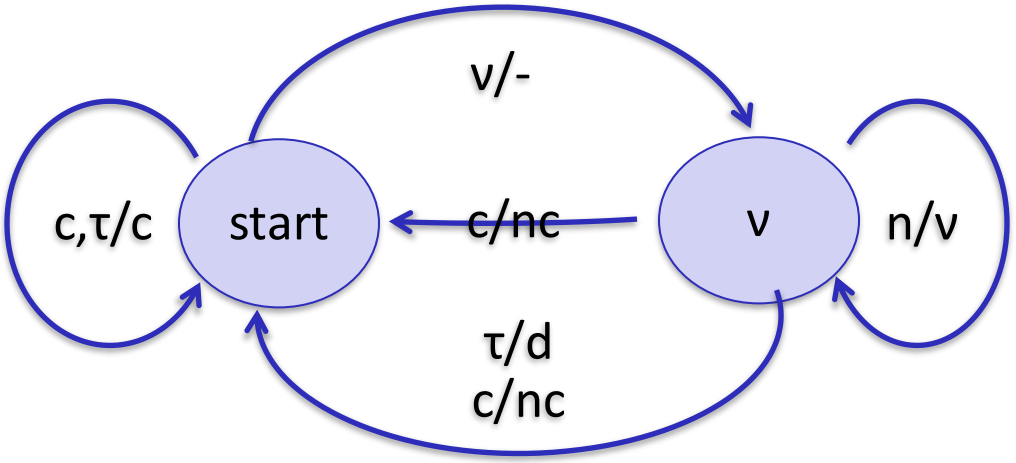
Μέρος 1: 2 → 1

- Έστω ότι βλέπουμε στο stdin το ν
- Πριν τυπώσουμε στο output το n πρέπει να δούμε αν ο επόμενος χαρακτήρας στο stdin είναι το τ. Αν ναι, τότε πρέπει να βάλουμε στο stdout το d διαφορετικά το n και όποιον άλλο χαρακτήρα διαβάσαμε.
- Αυτό ακούγεται σαν ένα σύνολο από συνθήκες/καταστάσεις που πρέπει να τηρούμε για όλες τις περιπτώσεις. Οι συνθήκες αυτές δεν είναι πολλές, αλλά οι συνδιασμοί τους δημιουργούν πολλές περιπτώσεις.
  - Θα μπορούσαμε να γράψουμε το πρόγραμμά μας σε αυτό το σημείο, αλλά θα καταλήξουμε σε μια μεγάλη «περιπτωσιολογία» if this then that.
  - Το πρόβλημα με αυτό είναι ότι μας δυσκολεύει να καταλάβουμε αν έχουμε καλύψει όλες τις περιπτώσεις σωστά και αν η ροή του προγράμματος είναι πάντα σωστή μέσα από τόσο πολλές περιπτώσεις.
- Εναλλακτικά, αυτές οι περιπτώσεις ακούγονται σαν «καταστάσεις»: Αν είμαι στην κατάσταση που έχω διαβάσει το «ν» τότε αν δω το «τ» θα τυπώσω d, διαφορετικά κάτι άλλο.
- Τέτοια συστήματα περιγράφονται εύκολα με μηχανές πεπερασμένων καταστάσεων (finite state machines – FSMs).



- Input είναι ο χαρακτήρας από το stdin και output είναι ο χαρακτήρας στο stdout.

FSM για την περίπτωση ”ντ”



- Το c αναπαριστά οποιονδήποτε χαρακτήρα εκτός από ν,τ. Παρατηρήστε ότι αυτές οι τρεις περιπτώσεις περιγράφουν ΌΛΑ τα δυνατά inputs που μπορεί να δει το πρόγραμμά μας.
- Αν ξεκινήσουμε από μια κατάσταση start τότε:
  - Αν δούμε c δεν έχει πολύ ενδιαφέρον απλά τυπώνουμε charmap[c]
  - Αν δούμε ν δεν είμαστε σίγουροι τι πρέπει να κάνουμε οπότε πηγαίνουμε σε μια κατάσταση ν για να το θυμόμαστε
  - Αν τώρα δούμε ν τότε τυπώνουμε n για το πρώτο ν και μένουμε στην ίδια κατάσταση για να συνεχίσουμε να θυμόμαστε το δεύτερο ν
  - Αν δούμε τ τότε τυπώνουμε d και πηγαίνουμε στην αρχική κατάσταση αφού δεν χρειάζεται να θυμόμαστε τίποτε
  - Αν δούμε c τότε τυπώνουμε n,charmap[c] αφού τελικά το ν δεν το ακολούθησε κάτι ενδιαφέρον
- Παρατηρήστε ότι σε κάθε κατάσταση πρέπει να έχουμε ένα βελάκι για ΚΆΘΕ δυνατό input (ν,τ,c). Αν το κάνουμε αυτό γνωρίζουμε ότι έχουμε μια περίπτωση που χειρίζεται οτιδήποτε input και αν δει το πρόγραμμά μας.
- Μπορούμε έτσι να συμπληρώσουμε και τις υπόλοιπες περιπτώσεις 2 → 1 (μπ, γκ, Ντ, κ.ο.κ.)

Χωρισμός σχεδιασμού από υλοποίηση

- Σε αυτό το σημείο μπορούμε να δούμε (κυριολεκτικά, σε σχήμα) αν το πρόγραμμά μας έχει σχεδιαστεί σωστά και αν αντιμετωπίζει κάθε περίπτωση όπως πρέπει
- Αυτό είναι σημαντικό: μπορούμε να επιχειρηματολογήσουμε για την λογική του προγράμματός μας **(σχεδιασμός**) χωρίς να έχουμε γράψει τον κώδικα **(υλοποίηση)**
- Αν το πρόγραμμά μας ΔΕΝ δίνει το σωστό αποτέλεσμα, μπορούμε κοιτάζοντας τα FSMs να δούμε που ενδεχομένως έχουμε σφάλει
- Η συγγραφή του κώδικα από δω και πέρα είναι μια μηχανική διαδικασία
  - Μάλιστα θα δείτε αργότερα ότι εργαλεία όπως ο lex και ο yacc που χρησιμοποιούνται για την υλοποίηση της πρώτης φάσης των compilers μετατρέπουν τέτοια FSMs σε κώδικα C

Μέρος 2: Μετατροπή FSM σε κώδικα C

- Μια FSM μπορεί να αναπαρασταθεί με διάφορους τρόπους σαν ένας πίνακας πχ

state \ input	v	T	c	…	
start	v/-	start/n	start/charmap[c]		f_s()
v	v/n	d/start	start/nc		f_n()
…					…
…				new state/output	

- Κάθε γραμμή αναπαριστά πως χειριζόμαστε κάθε κατάσταση
- Μπορούμε να υλοποιήσουμε κάθε γραμμή σαν μια συνάρτηση f με παράμετρο τον input char
- Οπότε

Το πρόγραμμα της FSM

- Χρειαζόμαστε μια μεταβλητή που αναπαριστά τις καταστάσεις και επομένως μπορεί να παίρνει ΜΟΝΟ συγκεκριμένες τιμές, πχ

```
enum state_T {S1, ...} s;
```

- Η συνάρτηση main επομένως φαίνεται σαν:

```
while (c is next char){
    switch (state) {
    case S1:
           if (c=='') output(); state = new_state;
           if (c=='') output(); state = new_state;
           break;
    case S2:
    ...
    }
}
```

- Αυτό κάνει την main μας πολύ «μακριά» και δύσκολη να ακολουθήσουμε την ροή της.
- Υπάρχει εναλλακτική; Οι συναρτήσεις μπορούν να μας βοηθήσουν κάπως:
  - Κάθε case μπορεί να αντικατασταθεί με μια συνάρτηση που περιέχει τον κώδικα του αντίστοιχου case
- Παρ΄όλα αυτά η main δεν αλλάζει και πολύ – εξακολουθεί να παραμένει μια μακριά και «δύσκολη» στην ροή συνάρτηση.

Περιεκτικότερη διατύπωση στη C

- Εν τέλη, αυτό που θέλουμε να κάνουμε είναι να επιλέξουμε μια συνάρτηση με βάση την κατάσταση που είμαστε και τον input char
- Αυτό είναι ένα array lookup – μόνο που οι συναρτήσεις των actions για κάθε state πρέπει να βρίσκονται σε ένα array
- Η C μας επιτρέπει να το κάνουμε αυτό (να ορίζουμε arrays από «συναρτήσεις») ως εξής
  - Βάζουμε τις συναρτήσεις σε ένα array, πχ action και

```
main(){
        local vars;
          while (c is next char) {
                  state = (action[state])(c);
          }
          return 0;
}
```

- Την ακριβή δήλωση του actions θα την καταλάβουμε στα επόμενα μαθήματα. Για τώρα μπορούμε να χρησιμοποιήσουμε κάτι σαν:
  - return_type (*actions[])(argument types);
  - Το "*" είναι το μόνο "αστεράκι" στο πρόγραμμά μας, και αυτή τη στιγμή ενδεχομένως δεν κατανοούμε την σκοπιμότητα
- Οπότε το πρόγραμμά μας πλέον είναι ένα σύνολο από συναρτήσεις με μια πολύ απλή main και εξαιρετικά απλή και κατανοητή ροή
  - Προσοχή στις δηλώσεις μας, ιδιαίτερα στις καταστάσεις και στον πίνακα των ενεργειών actions, ή όπως το ονομάσετε
  - Κάθε άλλη μεταβλητή να έχει τον σωστό τύπο για την δουλειά που κάνει
  - Δεν χρειάζεται δυναμική μνήμη (δεν πολυγνωρίζουμε τι είναι και τι κάνει ακόμη)
- Ουσιαστικά, ο κώδικας και τα λάθη που μπορεί να περιέχει αναφέρεται στο πως μια οποιαδήποτε FSM αναπαρίσταται σαν πρόγραμμα και όχι στην λογική της συγκεκριμένης FSM που μας ενδιαφέρει και που μπορούμε να την κάνουμε debug στο σχήμα των καταστάσεων τους Μέρους 1.

Final notes on Assignment 1

- Ο διαχωρισμός του design από την υλοποίηση είναι σχετικά εύκολος σε αυτή την άσκηση διότι η λειτουργικότητά της μπορεί να περιγραφεί σαν μια μηχανή πεπερασμένων καταστάσεων (FSM)
- Μπορεί να γίνει αυτό εύκολα για όλα τα προγράμματα;
  - Οπότε ίσως θα σήμαινε ότι μπορούμε να «σχεδιάζουμε» εμείς κάτι και στη συνέχει ο κώδικας (υλοποίηση) να παράγεται αυτόματα.
- Αυτή η τεχνική έχει ευρεία εφαρμογή σε όλα τα συστήματα που κάνουν parse input files, προγράμματα, κοκ. Ουσιαστικά το πρώτο βήμα όλων το compilers/assembers απότελείται από ένα τέτοιο στάδιο/FSM.
  - Σήμερα πλέον ο κώδικας για αυτό το στάδιο των compilers γίνεται generate αυτόματα με εργαλεία όπως ο lex/yacc.
- Ωστόσο, τα πιο πολλά συστήματα/προγράμματα δεν έχουν τόσο κανονική δομή και δεν περιγράφονται εύκολα με τον ίδιο τρόπο
- Παρόλα αυτά η αρχή ότι σκεφτόμαστε καθώς σχεδιάζουμε ένα σύστημα, στο χαρτί και στον πίνακα, και προχωρούμε στην υλοποίηση με βάση το design μας ισχύει για όλα (ειδικά τα πιο πολύπλοκα) συστήματα, όπως ασύγχρονα προγράμματα, λειτουργικά συστήματα, κατανεμημένα και παράλληλα συστήματα, κ.ο.κ.
  - Διαφορετικά πολλές φορές καταλήγουμε σε συστήματα που δουλεύουν για μερικές περιπτώσεις μόνο και τελικά δεν είναι χρήσιμα
- Lesson: Think before you code!

Εργασία

- Διαβάστε την εκφώνηση για την Άσκηση 1
- Υλοποιήστε την Άσκηση 1