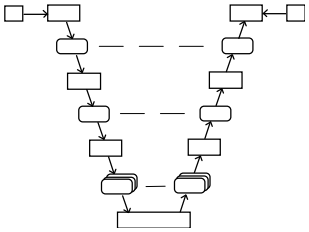
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Μεταγλωττιστές 2025

Θέμα εργασίας

Η γλώσσα Dana

Dana S. Scott (1932–), Prof. Emeritus, Carnegie Mellon University



Μεταγλωττιστές

`http://courses.softlab.ntua.gr/compilers/`

Διδάσκων: *Κωστής Σαγώνας*

Αθήνα, Φεβρουάριος 2025

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από κάθε ομάδα, το πολύ δυο σπουδαστών, ένας μεταγλωττιστής για τη γλώσσα Dana. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C/C++, Rust, Java, SML, OCaml, Haskell ή Python, αλλά έχετε υπ' όψη σας ότι κομμάτια του μεταγλωττιστή και εργαλεία που μπορεί να σας φανούν χρήσιμα μπορεί να μην είναι διαθέσιμα σε κάποιες από τις παραπάνω γλώσσες και σε αυτήν την περίπτωση θα πρέπει να τα υλοποιήσετε μόνοι σας. Η επιλογή κάποιας άλλης γλώσσας υλοποίησης μπορεί να γίνει κατόπιν συνεννόησης με τον διδάσκοντα. Επιτρέπεται να χρησιμοποιηθούν και επίσης συνιστάται η χρήση εργαλείων, π.χ. `flex`/`ML-Lex`/`ocamllex`–`Alex`, `bison`/`ML-Yacc`/`ocamlyacc`–`Happy`, `JavaCC`, κ.λπ., όπως και το `LLVM`. Περισσότερες πληροφορίες σχετικές με κάποια από αυτά τα εργαλεία θα δοθούν στις παραδόσεις.

Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή και η κατανομή μονάδων φαίνονται στον παρακάτω πίνακα. Οι ημερομηνίες παράδοσης αναγράφονται στη σελίδα του μαθήματος.

Τμήμα του μεταγλωττιστή	*Μονάδες*	*Bonus*
Λεκτικός αναλυτής	0.5	–
Συντακτικός αναλυτής	0.5	–
Σημασιολογικός αναλυτής	1.0	0.5
Ενδιάμεσος κώδικας	1.0	–
Βελτιστοποίηση	1.0	0.8
Τελικός κώδικας	1.0	0.7
Συνολική εργασία και έκθεση	5.0	2.0

Για τα διάφορα τμήματα της εργασίας πρέπει να παραδίδεται εμπρόθεσμα από κάθε ομάδα ο αντίστοιχος κώδικας σε ηλεκτρονική μορφή, καθώς και σαφείς οδηγίες για την παραγωγή ενός εκτελέσιμου προγράμματος επίδειξης της λειτουργίας του αντίστοιχου τμήματος, από τον κώδικα αυτόν. Καθυστερημένες ασκήσεις θα βαθμολογούνται με μικρότερο βαθμό, αντιστρόφως ανάλογα προς το χρόνο καθυστέρησης. Παρακαλούμε, **μην παραδίδετε τυπωμένες εργασίες**! Η μορφή και τα περιεχόμενα των παραδοτέων, συμπεριλαμβανομένης και της τελικής έκθεσης, πρέπει να συμφωνούν με τις οδηγίες που δίνονται στην ενότητα 4 του παρόντος.

Προαιρετικά τμήματα και μονάδες bonus

Το σύνολο των μονάδων της παρούσας εργασίας είναι 7. Από αυτές, οι 2 μονάδες είναι bonus (που σημαίνει ότι το σύνολο μονάδων του μαθήματος είναι 12) και αντιστοιχούν στα παρακάτω τμήματα της εργασίας, που είναι προαιρετικά:

- (20%) Layout (off-side rule).
- (40%) Βελτιστοποίηση ενδιάμεσου κώδικα με ανάλυση ροής δεδομένων και ελέγχου.
- (30%) Δέσμευση καταχωρητών και βελτιστοποίηση τελικού κώδικα.

Περιεχόμενα

1 Περιγραφή της γλώσσας Dana

Η γλώσσα Dana είναι μια απλή γλώσσα προστακτικού προγραμματισμού. Τα κύρια χαρακτηριστικά της εν συντομία είναι τα εξής:

- Απλή δομή και σύνταξη εντολών και εκφράσεων που μοιάζει με αυτήν της Python.
- Βασικοί τύποι δεδομένων για ακέραιους αριθμούς δύο μεγεθών και πίνακες.
- Απλές συναρτήσεις, πέρασμα κατ' αξία ή κατ' αναφορά.
- Εμβέλεια μεταβλητών όπως στην Pascal.
- Βιβλιοθήκη συναρτήσεων.

Περισσότερες λεπτομέρειες της γλώσσας δίνονται στις παραγράφους που ακολουθούν.

1.1 Λεκτικές μονάδες

Οι λεκτικές μονάδες της γλώσσας Dana χωρίζονται στις παρακάτω κατηγορίες:

- Τις *λέξεις κλειδιά*, οι οποίες είναι οι παρακάτω:

```
and     as      begin    break    byte     continue    decl
def     elif    else     end      exit     false       if
is      int     loop     not      or       ref         return
skip    true    var
```

- Τα *ονόματα*, τα οποία αποτελούνται από ένα γράμμα του λατινικού αλφαβήτου, πιθανώς ακολουθούμενο από μια σειρά γραμμάτων, δεκαδικών ψηφίων ή χαρακτήρων υπογράμμισης (underscore). Τα ονόματα δεν πρέπει να συμπίπτουν με τις λέξεις κλειδιά που αναφέρθηκαν παραπάνω. Πεζά και κεφαλαία γράμματα θεωρούνται διαφορετικά.

- Τις *ακέραιες σταθερές* χωρίς πρόσημο, που αποτελούνται από ένα ή περισσότερα δεκαδικά ψηφία. Παραδείγματα ακέραιων σταθερών είναι τα ακόλουθα:

```
0     42     1284     00200
```

- Τους *σταθερούς χαρακτήρες*, που αποτελούνται από ένα χαρακτήρα μέσα σε απλά 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 πίνακα 1. Παραδείγματα σταθερών χαρακτήρων είναι οι ακόλουθες:

```
'a'     '1'     '\n'     '\"     '\x1d'
```

- Τις *σταθερές συμβολοσειρές*, που αποτελούνται από μια ακολουθία κοινών χαρακτήρων ή ακολουθιών διαφυγής μέσα σε διπλά εισαγωγικά. Οι συμβολοσειρές δεν μπορούν να εκτείνονται σε περισσότερες από μια γραμμές προγράμματος. Παραδείγματα σταθερών συμβολοσειρών είναι οι ακόλουθες:

```
"abc"          "Route66"          "Helloworld!\n"
     "Name:\t\"DouglasAdams\"\nValue:\t42\n"
```

- Τους *συμβολικούς τελεστές*, οι οποίοι είναι οι παρακάτω:

```
+     -     *     /     %     !     &     |
=     <>    <     >     <=    >=
```

Πίνακας 1: Ακολουθίες διαφυγής (escape sequences).

Ακολουθία διαφυγής	Περιγραφή
\n	ο χαρακτήρας αλλαγής γραμμής (line feed)
\t	ο χαρακτήρας στηλοθέτησης (tab)
\r	ο χαρακτήρας επιστροφής στην αρχή της γραμμής (carriage return)
\0	ο χαρακτήρας με ASCII κωδικό 0
\\	ο χαρακτήρας \ (backslash)
\'	ο χαρακτήρας ' (απλό εισαγωγικό)
\"	ο χαρακτήρας " (διπλό εισαγωγικό)
\x$nn$	ο χαρακτήρας με ASCII κωδικό nn στο δεκαεξαδικό σύστημα

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

```
(     )     [     ]     ,     :     :=
```

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα Dana μπορεί επίσης να περιέχει τα παρακάτω, τα οποία διαχωρίζουν λεκτικές μονάδες και αγνοούνται:

- *Κενούς χαρακτήρες*, 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 μιας γραμμής*, τα οποία αρχίζουν με το χαρακτήρα # και τερματίζονται με το τέλος της τρέχουσας γραμμής.
- *Σχόλια πολλών γραμμών*, τα οποία αρχίζουν με την ακολουθία χαρακτήρων `(*` και τερματίζονται με την ακολουθία χαρακτήρων `*)`. Τα σχόλια αυτής της μορφής επιτρέπεται να είναι φωλιασμένα.

1.2 Τύποι δεδομένων

Η Dana υποστηρίζει δύο βασικούς τύπους δεδομένων:

- `int`: ακέραιοι αριθμοί μεγέθους τουλάχιστον 16 bit (-32768 έως 32767), και
- `byte`: μη αρνητικοί ακέραιοι αριθμοί των 8 bit (0 έως 255).

Εκτός από τους βασικούς τύπους, η Dana υποστηρίζει επίσης τύπους πινάκων, οι οποίοι συμβολίζονται με t `[`n`]`. Το n θα πρέπει να είναι ακέραια σταθερά με θετική τιμή και το t έγκυρος τύπος. Υποστηρίζει επίσης έμμεσα έναν τύπο λογικών εκφράσεων, ο οποίος όμως χρησιμοποιείται μόνο στις συνθήκες της εντολής `if`. Αυτός ο τύπος δεν πρέπει να συγχέεται με τον τύπο `byte`, που μπορεί να χρησιμοποιείται για τη δήλωση μεταβλητών λογικού τύπου.

1.3 Δομή του προγράμματος

Η γλώσσα Dana είναι μια δομημένη (block structured) γλώσσα. Ένα πρόγραμμα έχει χοντρικά την ίδια δομή με ένα πρόγραμμα Pascal. Οι δομικές μονάδες μπορούν να είναι φωλιασμένες η μία μέσα στην άλλη και οι κανόνες εμβέλειας είναι οι ίδιοι με αυτούς της Pascal. Το κύριο πρόγραμμα είναι μία δομική μονάδα που δεν επιστρέφει αποτέλεσμα και δε δέχεται παραμέτρους.

Κάθε δομική μονάδα μπορεί να περιέχει προαιρετικά:

- Δηλώσεις μεταβλητών.

- Ορισμούς υποπρογραμμάτων.
- Δηλώσεις υποπρογραμμάτων, οι ορισμοί των οποίων θα ακολουθήσουν.

Τελευταίο συστατικό στον ορισμό μιας δομικής μονάδας είναι το σώμα της, που είναι ένα μπλοκ εντολών (βλ. ενότητα 1.6).

1.3.1 Μεταβλητές

Οι δηλώσεις μεταβλητών γίνονται με τη λέξη κλειδί `var`. Ακολουθούν ένα ή περισσότερα ονόματα μεταβλητών, η λέξη κλειδί `is` και ένας τύπος δεδομένων. Περισσότερες συνεχόμενες δηλώσεις μεταβλητών μπορούν να γίνουν επαναλαμβάνοντας τη λέξη κλειδί `var`. Παραδείγματα δηλώσεων είναι:

```
var i     is int
var x y z is int
var s     is byte [80]
```

1.3.2 Δομικές μονάδες

Ο *ορισμός* μίας δομικής μονάδας γίνεται με τη λέξη κλειδί `def`, που ακολουθείται από την επικεφαλίδα της δομικής μονάδας, τις τοπικές δηλώσεις και το σώμα της. Στην επικεφαλίδα αναφέρεται το όνομα της δομικής μονάδας, ο τύπος επιστροφής (προαιρετικά) και οι τυπικές της παράμετροι (προαιρετικά). Ο τύπος επιστροφής παραλείπεται για τις δομικές μονάδες που δεν επιστρέφουν αποτέλεσμα (πρβλ. διαδικασίες στην Pascal). Αν υπάρχει, ο τύπος επιστροφής ακολουθεί τη λέξη κλειδί `is` και δεν μπορεί να είναι τύπος πίνακα. Οι τυπικές παράμετροι, αν υπάρχουν, γράφονται μετά το διαχωριστή `:` (δύο τελείες) και η σύνταξή τους μοιάζει με αυτή της δήλωσης μεταβλητών, με τη διαφορά ότι η λέξη κλειδί `as` αντικαθιστά την `is`.

Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομά της, τον τύπο της και τον τρόπο περάσματος. Η γλώσσα Dana υποστηρίζει πέρασμα παραμέτρων κατ' αξία (by value) και κατ' αναφορά (by reference). Κατ' αξία περνούν μόνο οι παράμετροι των βασικών τύπων. Αν όμως προηγείται των ονομάτων τους η λέξη κλειδί `ref`, τότε περνούν κατ' αναφορά. Επίσης, όλες οι παράμετροι τύπου πίνακα περνούν κατ' αναφορά χωρίς να γίνεται χρήση της λέξης κλειδί `ref`. Στους τύπους πίνακα που χρησιμοποιούνται για τις δηλώσεις τυπικών παραμέτρων μπορεί να παραλείπεται το μέγεθος της πρώτης διάστασης.

Ακολουθούν παραδείγματα επικεφαλίδων ορισμών δομικών μονάδων.

```
def p1
def p2: n as int
def p3: a b as int, c as ref byte
def f1 is int: x as int
def f2 is int: s as byte []
def matrix_mult: a b c as int [10][10]
```

Οι τοπικές δηλώσεις μιας δομικής μονάδας ακολουθούν την επικεφαλίδα. Η Dana ακολουθεί τους κανόνες εμβέλειας της Pascal, όσον αφορά στην ορατότητα των ονομάτων μεταβλητών, δομικών μονάδων και παραμέτρων.

Στην περίπτωση αμοιβαία αναδρομικών υποπρογραμμάτων, το όνομα ενός υποπρογράμματος χρειάζεται να εμφανιστεί πριν τον ορισμό της. Στην περίπτωση αυτή, για να μην παραβιαστούν οι κανόνες εμβέλειας, πρέπει να έχει προηγηθεί μια *δήλωση* της επικεφαλίδας αυτού του υποπρογράμματος, χωρίς το σώμα του. Αυτό γίνεται με τη λέξη κλειδί `decl` αντί της `def`.

1.4 Εκφράσεις και συνθήκες

Κάθε έκφραση της Dana διαθέτει ένα μοναδικό τύπο και μπορεί να αποτιμηθεί δίνοντας ως αποτέλεσμα μια τιμή αυτού του τύπου. Οι εκφράσεις διακρίνονται σε δύο κατηγορίες: αυτές που δίνουν l-values, οι

οποίες περιγράφονται στην ενότητα 1.4.1 και αυτές που δίνουν r-values, που περιγράφονται στις ενότητες 1.4.2 ως 1.4.4. Τα δυο αυτά είδη τιμών έχουν πάρει το όνομά τους από τη θέση τους σε μια εντολή ανάθεσης: οι l-values εμφανίζονται στο αριστερό μέλος της ανάθεσης ενώ οι r-values στο δεξιό.

Οι συνθήκες της Dana περιγράφονται στην ενότητα 1.4.3. Χρησιμοποιούνται μόνο σε συνδυασμό με την εντολή `if` και ο υπολογισμός τους δίνει ως αποτέλεσμα μια λογική τιμή (αληθή ή ψευδή).

Οι εκφράσεις και οι συνθήκες μπορούν να εμφανίζονται μέσα σε παρενθέσεις, που χρησιμοποιούνται για λόγους ομαδοποίησης.

1.4.1 L-values

Οι l-values αντιπροσωπεύουν αντικείμενα που καταλαμβάνουν χώρο στη μνήμη του υπολογιστή κατά την εκτέλεση του προγράμματος και τα οποία μπορούν να περιέχουν τιμές. Τέτοια αντικείμενα είναι οι μεταβλητές, οι παράμετροι των δομικών μονάδων, τα στοιχεία πινάκων και οι σταθερές συμβολοσειρές. Συγκεκριμένα:

- Το όνομα μιας μεταβλητής ή μιας παραμέτρου συνάρτησης είναι l-value και αντιστοιχεί στο εν λόγω αντικείμενο. Ο τύπος της l-value είναι ο τύπος του αντίστοιχου αντικειμένου.

- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1 είναι l-values. Έχουν τύπο `byte [`n`]`, όπου n είναι ο αριθμός χαρακτήρων που περιέχονται στη συμβολοσειρά προσαυξημένος κατά ένα. Κάθε τέτοια l-value αντιστοιχεί σε ένα αντικείμενο τύπου πίνακα, στον οποίο βρίσκονται αποθηκευμένοι με τη σειρά οι ASCII κωδικοί των χαρακτήρων της συμβολοσειράς. Στο τέλος του πίνακα αποθηκεύεται αυτόματα ο χαρακτήρας `'\0'`, σύμφωνα με τη σύμβαση που ακολουθεί η γλώσσα C για τις συμβολοσειρές. Οι σταθερές συμβολοσειρές είναι το μόνο είδος σταθεράς τύπου πίνακα που επιτρέπεται στην Dana.

- Αν l είναι μια l-value τύπου t `[`m`]` και e είναι μια έκφραση τύπου `int`, τότε l`[`e`]` είναι μια l-value με τύπο t. Αν η τιμή της έκφρασης e είναι ο μη αρνητικός ακέραιος n τότε αυτή η l-value αντιστοιχεί στο στοιχείο με δείκτη n του πίνακα που αντιστοιχεί στην l. Η αρίθμηση των στοιχείων του πίνακα ξεκινά από το μηδέν. Η τιμή του n δεν πρέπει να υπερβαίνει τα πραγματικά όρια του πίνακα ($n < m$).

Αν μια l-value χρησιμοποιηθεί ως έκφραση, η τιμή αυτής της έκφρασης είναι ίση με την τιμή που περιέχεται στο αντικείμενο που αντιστοιχεί στην l-value.

1.4.2 Σταθερές

Στις r-values της γλώσσας Dana συγκαταλέγονται οι ακόλουθες σταθερές:

- Οι ακέραιες σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `int` και η τιμή τους είναι ίση με τον μη αρνητικό ακέραιο αριθμό που παριστάνουν.

- Οι σταθεροί χαρακτήρες, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `byte` και η τιμή τους είναι ίση με τον κωδικό ASCII του χαρακτήρα που παριστάνουν.

- Οι λέξεις κλειδιά `true` και `false`. Είναι ισοδύναμες με τους σταθερούς χαρακτήρες `'\x01'` και `'\0'`, αντίστοιχα.

1.4.3 Τελεστές

Οι τελεστές της Dana διακρίνονται σε τελεστές με ένα ή δύο τελούμενα. Οι τελεστές με ένα τελούμενο γράφονται πρίν από αυτό (prefix), ενώ οι τελεστές με δύο τελούμενα γράφονται πάντα μεταξύ των τελουμένων (infix). Η αποτίμηση των τελουμένων γίνεται από αριστερά προς τα δεξιά. Οι τελεστές με δύο τελούμενα αποτιμούν υποχρεωτικά και τα δύο τελούμενα, με εξαίρεση τους τελεστές `and` και `or`, όπως περιγράφεται παρακάτω.

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της Dana.

Τελεστές	Περιγραφή	Αριθμός τελουμένων	Θέση και προσεταιριστικότητα
`+ - !`	Πρόσημα, λογική άρνηση για byte	1	prefix
`* / % &`	Πολλαπλασιαστικοί τελεστές	2	infix, αριστερή
`+ - |`	Προσθετικοί τελεστές	2	infix, αριστερή
`= <> > < <= >=`	Σχεσιακοί τελεστές	2	infix, καμία
`not`	Λογική άρνηση συνθηκών	1	prefix
`and`	Λογική σύζευξη συνθηκών	2	infix, αριστερή
`or`	Λογική διάζευξη συνθηκών	2	infix, αριστερή

Όλοι οι τελεστές της Dana έχουν ως αποτέλεσμα r-value ή συνθήκη. Περιγράφονται εκτενώς παρακάτω.

- Οι τελεστές με ένα τελούμενο `+` και `-` υλοποιούν τους τελεστές προσήμου. Το τελούμενο πρέπει να είναι έκφραση τύπου `int` και το αποτέλεσμα είναι r-value του ίδιου τύπου.
- Οι τελεστές με ένα τελούμενο `!` και `not` υλοποιούν τη λογική άρνηση. Ο πρώτος εφαρμόζεται σε λογικές τιμές που είναι εκφράσεις, ενώ ο δεύτερος σε συνθήκες. Το τελούμενο του `!` πρέπει να είναι έκφραση τύπου `byte` και το αποτέλεσμα είναι r-value του ίδιου τύπου. Το τελούμενο του `not` πρέπει να είναι συνθήκη και το ίδιο είναι το αποτέλεσμά του.
- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `/` και `%` υλοποιούν τις αριθμητικές πράξεις. Τα τελούμενα πρέπει να είναι εκφράσεις του ίδιου τύπου `int` ή `byte` και το αποτέλεσμα είναι r-value του ίδιου τύπου.
- Οι τελεστές `=`, `<>`, `<`, `>`, `<=` και `>=` υλοποιούν τις σχέσεις σύγκρισης μεταξύ αριθμών. Τα τελούμενα πρέπει να είναι εκφράσεις του ίδιου τύπου `int` ή `byte` και το αποτέλεσμα είναι συνθήκη.
- Οι τελεστές `&` και `|` υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης για εκφράσεις. Τα τελούμενα πρέπει να είναι εκφράσεις τύπου `byte` και το αποτέλεσμα είναι r-value του ίδιου τύπου. Το αποτέλεσμα του `&` έχει την τιμή `false` αν ένα από τα τελούμενά του έχει την τιμή `false`, διαφορετικά έχει την τιμή `true`. Το αποτέλεσμα του `|` έχει την τιμή `false` αν και τα δύο τελούμενά του έχουν την τιμή `false`, διαφορετικά έχει την τιμή `true`.
- Οι τελεστές `and` και `or` υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης για συνθήκες. Τα τελούμενα πρέπει να είναι συνθήκες και το ίδιο είναι και το αποτέλεσμα. Η αποτίμηση συνθηκών που χρησιμοποιούν αυτούς τους τελεστές γίνεται με *βραχυκύκλωση* (short-circuit). Δηλαδή, αν το αποτέλεσμα της συνθήκης είναι γνωστό από την αποτίμηση μόνο του πρώτου τελούμενου, το δεύτερο τελούμενο δεν αποτιμάται καθόλου.

Στον πίνακα 2 ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της Dana. Οι γραμμές που βρίσκονται υψηλότερα στον πίνακα περιέχουν τελεστές μεγαλύτερης προτεραιότητας. Τελεστές που βρίσκονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα.

1.4.4 Κλήση δομικών μονάδων ως συναρτήσεων

Αν f είναι το όνομα μιας δομικής μονάδας με υπαρκτό τύπο επιστροφής t, τότε η έκφραση $f(e_1, \ldots, e_n)$ είναι μια r-value με τύπο t. Ο αριθμός των πραγματικών παραμέτρων n πρέπει να συμπίπτει με τον αριθμό των τυπικών παραμέτρων της f. Επίσης, ο τύπος και το είδος κάθε πραγματικής παραμέτρου πρέπει να

συμπίπτει με τον τύπο και τον τρόπο περάσματος της αντίστοιχης τυπικής παραμέτρου, σύμφωνα με τους παρακάτω κανόνες.

- Αν η τυπική παράμετρος είναι τύπου t και περνά κατ' αξία, τότε η αντίστοιχη πραγματική παράμετρος πρέπει να είναι έκφραση τύπου t.

- Αν η τυπική παράμετρος είναι τύπου t και περνά κατ' αναφορά, τότε η αντίστοιχη πραγματική παράμετρος πρέπει να είναι l-value τύπου t.

Κατά την κλήση μιας δομικής μονάδας, οι πραγματικές παράμετροι αποτιμώνται από αριστερά προς τα δεξιά.

1.5 Εντολές

Οι εντολές που υποστηρίζει η γλώσσα Dana είναι οι ακόλουθες:

- Η κενή εντολή `skip` που δεν κάνει καμία ενέργεια.

- Η εντολή ανάθεσης ℓ `:=` e που αναθέτει την τιμή της έκφρασης e στην l-value ℓ. Η l-value ℓ πρέπει να είναι τύπου t που δεν είναι τύπος πίνακα, ενώ ή έκφραση e πρέπει να είναι του ίδιου τύπου t.

- Η εντολή κλήσης δομικής μονάδας. Αν η δομική μονάδα δε δέχεται παραμέτρους, αυτό γίνεται με αναγραφή του ονόματός της. Διαφορετικά, οι πραγματικές παράμετροι ακολουθούν το διαχωριστή `:` (δύο τελείες), χωρισμένες με κόμματα. Οι προϋποθέσεις για την κλήση είναι οι ίδιες όπως στην ενότητα 1.4.4 με τη διαφορά ότι δεν πρέπει να ορίζεται τύπος επιστροφής για τη δομική μονάδα.

- Η εντολή ελέγχου `if` c_1 `:` b_1 `elif` c_2 `:` b_2 `else :` b_2, όπου το σκέλος `elif` μπορεί να επαναλαμβάνεται μηδέν ή περισσότερες φορές και το σκέλος `else` είναι προαιρετικό. Τα c_1 και c_2 πρέπει να είναι έγκυρες συνθήκες. Τα b_1, b_2 και b_3 πρέπει να είναι έγκυρα μπλοκ εντολών. Η σημασιολογία αυτής της εντολής είναι όπως στη Python.

- Η εντολή ελέγχου `loop` i `:` b. Το αναγνωριστικό i είναι προαιρετικό: αν υπάρχει τότε χρησιμοποιείται ως όνομα του βρόχου και ακολουθεί τους κανόνες εμβέλειας της Pascal. Το b πρέπει να είναι ένα έγκυρο μπλοκ εντολών. Ο βρόχος `loop` εκτελεί το μπλοκ εντολών b επ' άπειρον, εκτός αν τερματιστεί με μία από τις εντολές `break`, `exit` ή `return`.

- Η εντολή άλματος `break :` i που τερματίζει την εκτέλεση του βρόχου i. Η εντολή αυτή πρέπει να εμφανίζεται μέσα στο σώμα του βρόχου i. Ο διαχωριστής `:` (δύο τελείες) και το αναγνωριστικό i είναι προαρετικά. Αν απουσιάζουν, τότε τερματίζεται ο εσωτερικότερος βρόχος.

- Η εντολή άλματος `continue :` i που επαναλαμβάνει την εκτέλεση του βρόχου i από την αρχή του. Η εντολή αυτή πρέπει να εμφανίζεται μέσα στο σώμα του βρόχου i. Ο διαχωριστής `:` (δύο τελείες) και το αναγνωριστικό i είναι προαρετικά. Αν απουσιάζουν, τότε επαναλαμβάνεται ο εσωτερικότερος βρόχος.

- Η εντολή άλματος `exit` που τερματίζει την εκτέλεση της τρέχουσας δομικής μονάδας. Η εντολή αυτή πρέπει να εμφανίζεται στο σώμα μίας δομικής μονάδας χωρίς τύπο επιστροφής.

- Η εντολή άλματος `return :` e που τερματίζει την εκτέλεση της τρέχουσας δομικής μονάδας και επιστρέφει ως αποτέλεσμα την τιμή της έκφρασης e. Η εντολή αυτή πρέπει να εμφανίζεται στο σώμα μίας δομικής μονάδας με τύπο επιστροφής t και η έκφραση e πρέπει να έχει τον ίδιο τύπο t.

1.6 Μπλοκ εντολών και μηχανισμός διάταξης

Τα μπλοκ εντολών εμφανίζονται στα προγράμματα της Dana σε τρία σημεία: στα σώματα των δομικών μονάδων, στην εντολή `if` και στην εντολή `loop`. Αποτελούνται από ακολουθίες μίας ή περισσότερων εντολών και έχουν δύο δυνατές μορφές:

- Στην απλούστερη μορφή του, ένα μπλοκ εντολών είναι μία ακολουθία μίας ή περισσότερων εντολών που περικλείεται από τις λέξεις κλειδιά `begin` και `end`, όπως παρακάτω.

  ```
  def fact is int: n as int
  begin
    if n <= 1:  begin return: 0             end
    else:       begin return: n * fact(n-1) end
  end
  ```

  Προσέξτε ότι σε αυτήν τη μορφή οι λέξεις κλειδιά `begin` και `end` είναι υποχρεωτικές ακόμα και όταν περικλείουν μόνο μία εντολή. Σημειώστε επίσης ότι μεταξύ των εντολών δεν απαιτείται κανένα διαχωριστικό (αυτό δε φαίνεται στο παραπάνω παράδειγμα).

- Για να αποφεύγονται οι λέξεις κλειδιά `begin` και `end`, η Dana διαθέτει μία δεύτερη μορφή μπλοκ εντολών, που καταργεί αυτές τις δύο λέξεις. Η μορφή αυτή υποστηρίζεται από το μηχανισμό *διάταξης* (layout), γνωστό επίσης ως "κανόνα οφσάιντ" (off-side rule), που πρωτοεμφανίστηκε στη γλώσσα ISWIM και σήμερα συναντάται στην Python και τη Haskell.

  Σύμφωνα με το μηχανισμό διάταξης, η ομαδοποίηση των εντολών γίνεται βάσει της στοίχισής τους (indentation) μέσα στον κώδικα του προγράμματος. Η γλώσσα Dana χρησιμοποιεί έναν αρκετά απλό κανόνα οφσάιντ, αρκετά απλούστερο από αυτόν της Python ή της Haskell:

  1. Για κάθε μπλοκ εντολών αναζητούμε την αμέσως προηγούμενη λεκτική μονάδα `def`, `if`, `elif`, `else` ή `loop`. Η λεκτική αυτή μονάδα ονομάζεται *οδηγός*. Για παράδειγμα, στην παραπάνω δομική μονάδα, οι οδηγοί των τριών μπλοκ εντολών είναι οι λεκτικές μονάδες `def`, `if` και `else` στην πρώτη, την τρίτη και την τέταρτη γραμμή αντίστοιχα.
  2. Ένα μπλοκ εντολών που δεν αρχίζει με τη λέξη κλειδί `begin` τερματίζεται με την πρώτη λεκτική μονάδα[1] που:

     (α□) είναι η πρώτη στη γραμμή της, δηλαδή προηγούνται αυτής μόνο λευκοί χαρακτήρες ή/και σχόλια, και

     (β□) ο πρώτος χαρακτήρας της γράφεται *όχι δεξιότερα* από τον πρώτο χαρακτήρα του οδηγού.
  3. Με το τέλος του προγράμματος τερματίζονται όλα τα τυχόν ανοιχτά μπλοκ εντολών αυτής της μορφής.

  Βάσει του κανόνα οφσάιντ, ο παραπάνω κώδικας μπορεί ισοδύναμα να γραφεί με τη μορφή:

  ```
  def fact is int: n as int
    if n <= 1:  return: 0
    else:       return: n * fact(n-1)
  ```

  Το μπλοκ εντολών του `def` συμπεριλαμβάνει τη δεύτερη και την τρίτη γραμμή, αφού και οι δύο είναι στοιχισμένες δεξιότερα από τον οδηγό. Το μπλοκ εντολών του `if` τερματίζεται με τη λεκτική μονάδα `else` (που έχει την ίδια στοίχιση με τον οδηγό). Τα μπλοκ εντολών του `else` και του `def` τερματίζονται με το τέλος του προγράμματος.

Για περισσότερα και πιο πολύπλοκα παραδείγματα, προσέξτε καλά τη στοίχιση των προγραμμάτων της ενότητας 3. Όλα τα προγράμματα έχουν μπλοκ εντολών μόνο της δεύτερης μορφής.

[1] Στις λεκτικές μονάδες δε συγκαταλέγονται οι "λευκοί χαρακτήρες", π.χ. τα κενά διαστήματα ή τα σχόλια.

1.7 Βιβλιοθήκη έτοιμων συναρτήσεων

Η Dana υποστηρίζει ένα σύνολο προκαθορισμένων δομικών μονάδων, οι οποίες έχουν υλοποιηθεί σε assembly του x86 ως μια *βιβλιοθήκη χρόνου εκτέλεσης* (run-time library). Είναι ορατές σε κάθε δομική μονάδα, εκτός αν επισκιάζονται από μεταβλητές, παραμέτρους ή άλλες δομικές μονάδες με το ίδιο όνομα. Παρακάτω δίνονται οι δηλώσεις τους και εξηγείται η λειτουργία τους.

1.7.1 Είσοδος και έξοδος

```
decl writeInteger: n as int
decl writeByte:    b as byte
decl writeChar:    b as byte
decl writeString:  s as byte []
```

Οι συναρτήσεις αυτές χρησιμοποιούνται για την εκτύπωση τιμών που ανήκουν στους βασικούς τύπους της Dana, καθώς και για την εκτύπωση συμβολοσειρών. Οι `writeByte` και `writeChar` διαφέρουν ως προς το ότι η πρώτη εκτυπώνει την αριθμητική τιμή του byte ενώ η δεύτερη το χαρακτήρα που αντιστοιχεί σε αυτόν τον ASCII κωδικό.

```
decl readInteger is int
decl readByte is byte
decl readChar is byte
decl readString: n as int, s as byte []
```

Αντίστοιχα, οι παραπάνω συναρτήσεις χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της Dana και για την εισαγωγή συμβολοσειρών. Η συνάρτηση `readString` χρησιμοποιείται για την ανάγνωση μιας συμβολοσειράς μέχρι τον επόμενο χαρακτήρα αλλαγής γραμμής. Οι παράμετροι της καθορίζουν το μέγιστο αριθμό χαρακτήρων (συμπεριλαμβανομένου του τελικού `'\0'`) που επιτρέπεται να διαβαστούν και τον πίνακα χαρακτήρων στον οποίο αυτοί θα τοποθετηθούν. Ο χαρακτήρας αλλαγής γραμμής δεν αποθηκεύεται. Αν το μέγεθος του πίνακα εξαντληθεί πριν συναντηθεί χαρακτήρας αλλαγής γραμμής, η ανάγνωση θα συνεχιστεί αργότερα από το σημείο όπου διακόπηκε.

1.7.2 Συναρτήσεις μετατροπής

```
decl extend is int:  b as byte
decl shrink is byte: i as int
```

Η πρώτη επεκτείνει μια τιμή τύπου `byte` στον αντίστοιχο αριθμό τύπου `int`. Η δεύτερη επιστρέφει μια τιμή τύπου `byte` που περιέχει τα 8 λιγότερο σημαντικά bits της παραμέτρου της.

1.7.3 Συναρτήσεις διαχείρισης συμβολοσειρών

```
decl strlen is int: s as byte []
decl strcmp is int: s1 s2 as byte []
decl strcpy: trg src as byte []
decl strcat: trg src as byte []
```

Οι συναρτήσεις αυτές έχουν ακριβώς την ίδια λειτουργία με τις συνώνυμές τους στη βιβλιοθήκη συναρτήσεων της γλώσσας C.

2 Πλήρης γραμματική της Dana

Η σύνταξη της γλώσσας Dana δίνεται παρακάτω σε μορφή EBNF. Η γραμματική που ακολουθεί είναι *διφορούμενη*, οι περισσότερες αμφισημίες όμως μπορούν να ξεπερασθούν αν λάβει κανείς υπόψη τους κανόνες προτεραιότητας και προσεταιριστικότητας των τελεστών, όπως περιγράφονται στον πίνακα 2.

Τα σύμβολα ⟨id⟩, ⟨int-const⟩, ⟨char-const⟩, ⟨string-literal⟩ και ⟨auto-end⟩ είναι τερματικά σύμβολα της γραμματικής. Το σύμβολο ⟨auto-end⟩ εισάγεται αυτόματα από το λεκτικό αναλυτή, βάσει του μηχανισμού διάταξης (βλ. ενότητα 1.6).

⟨program⟩ ::= ⟨func-def⟩
⟨func-def⟩ ::= "**def**" ⟨header⟩ (⟨local-def⟩)* ⟨block⟩
⟨header⟩ ::= ⟨id⟩ ["**is**" ⟨data-type⟩] ["**:**" ⟨fpar-def⟩ ("**,**" ⟨fpar-def⟩)*]
⟨fpar-def⟩ ::= (⟨id⟩)+ "**as**" ⟨fpar-type⟩
⟨data-type⟩ ::= "**int**" | "**byte**"
⟨type⟩ ::= ⟨data-type⟩ ("**[**" ⟨int-const⟩ "**]**")*
⟨fpar-type⟩ ::= ⟨type⟩ | "**ref**" ⟨data-type⟩ | ⟨data-type⟩ "**[**" "**]**" ("**[**" ⟨int-const⟩ "**]**")*
⟨local-def⟩ ::= ⟨func-def⟩ | ⟨func-decl⟩ | ⟨var-def⟩
⟨func-decl⟩ ::= "**decl**" ⟨header⟩
⟨var-def⟩ ::= "**var**" (⟨id⟩)+ "**is**" ⟨type⟩
⟨stmt⟩ ::= "**skip**" | ⟨l-value⟩ "**:=**" ⟨expr⟩ | ⟨proc-call⟩ | "**exit**" | "**return**" "**:**" ⟨expr⟩
| "**if**" ⟨cond⟩ "**:**" ⟨block⟩ ("**elif**" ⟨cond⟩ "**:**" ⟨block⟩)* ["**else**" "**:**" ⟨block⟩]
| "**loop**" [⟨id⟩] "**:**" ⟨block⟩ | "**break**" ["**:**" ⟨id⟩] | "**continue**" ["**:**" ⟨id⟩]
⟨block⟩ ::= "**begin**" (⟨stmt⟩)+ "**end**" | (⟨stmt⟩)+ ⟨auto-end⟩
⟨proc-call⟩ ::= ⟨id⟩ ["**:**" ⟨expr⟩ ("**,**" ⟨expr⟩)*]
⟨func-call⟩ ::= ⟨id⟩ "**(**" [⟨expr⟩ ("**,**" ⟨expr⟩)*] "**)**"
⟨l-value⟩ ::= ⟨id⟩ | ⟨string-literal⟩ | ⟨l-value⟩ "**[**" ⟨expr⟩ "**]**"
⟨expr⟩ ::= ⟨int-const⟩ | ⟨char-const⟩ | ⟨l-value⟩ | "**(**" ⟨expr⟩ "**)**" | ⟨func-call⟩
| ("**+**" | "**-**") ⟨expr⟩ | ⟨expr⟩ ("**+**" | "**-**" | "*****" | "**/**" | "**%**") ⟨expr⟩
| "**true**" | "**false**" | "**!**" ⟨expr⟩ | ⟨expr⟩ ("**&**" | "**|**") ⟨expr⟩
⟨cond⟩ ::= ⟨expr⟩ | "**(**" ⟨cond⟩ "**)**" | "**not**" ⟨cond⟩ | ⟨cond⟩ ("**and**" | "**or**") ⟨cond⟩
| ⟨expr⟩ ("**=**" | "**<>**" | "**<**" | "**>**" | "**<=**" | "**>=**") ⟨expr⟩

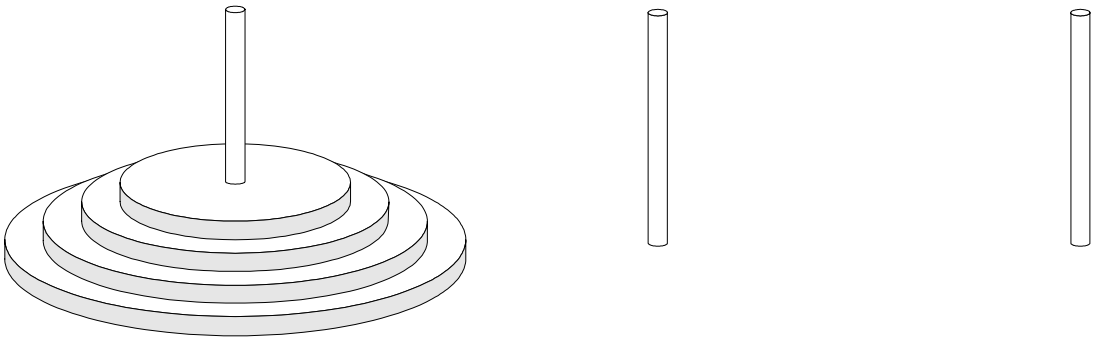
3 Παραδείγματα

Στην παράγραφο αυτή δίνονται πέντε παραδείγματα προγραμμάτων στη γλώσσα Dana, η πολυπλοκότητα των οποίων κυμαίνεται σημαντικά. Για κάποια από αυτά τα παραδείγματα (ή για παρόμοια προγράμματα), μπορείτε να βρείτε τον αρχικό, τον ενδιάμεσο κώδικα (χωρίς βελτιστοποίηση), τη μορφή των εγγραφημάτων δραστηριοποίησης των δομικών μονάδων, καθώς και τον τελικό κώδικα σε αντίστοιχα φυλλάδια περιγραφής γλωσσών που δόθηκαν ως θέματα εργασίας στο ίδιο μάθημα σε προηγούμενα έτη, μέσω της ιστοσελίδας του μαθήματος.

3.1 Πες γεια!

Το παρακάτω παράδειγμα είναι ένα από τα απλούστερα πρόγραμματα στη γλώσσα Dana που παράγει κάποιο αποτέλεσμα ορατό στο χρήστη. Το πρόγραμμα αυτό τυπώνει απλώς ένα μήνυμα.

```
def hello
  writeString: "Hello world!\n"
```



Σχήμα 1: Οι πύργοι του Hanoi.

3.2 Οι πύργοι του Hanoi

Το πρόγραμμα που ακολουθεί λύνει το πρόβλημα των πύργων του Hanoi. Μια σύντομη περιγραφή του προβλήματος δίνεται παρακάτω.

Υπάρχουν τρεις στύλοι, στον πρώτο από τους οποίους είναι περασμένοι n το πλήθος δακτύλιοι. Οι εξωτερικές διάμετροι των δακτυλίων είναι διαφορετικές και αυτοί είναι περασμένοι από κάτω προς τα πάνω σε φθίνουσα σειρά εξωτερικής διαμέτρου, όπως φαίνεται στο σχήμα 1. Ζητείται να μεταφερθούν οι δακτύλιοι από τον πρώτο στον τρίτο στύλο (χρησιμοποιώντας το δεύτερο ως βοηθητικό χώρο), ακολουθώντας όμως τους εξής κανόνες:

- Κάθε φορά επιτρέπεται να μεταφερθεί ένας μόνο δακτύλιος, από κάποιο στύλο σε κάποιον άλλο στύλο.

- Απαγορεύεται να τοποθετηθεί δακτύλιος με μεγαλύτερη διάμετρο πάνω από δακτύλιο με μικρότερη διάμετρο.

Το πρόγραμμα στη γλώσσα Dana που λύνει αυτό το πρόβλημα δίνεται παρακάτω. Η συνάρτηση `hanoi` είναι αναδρομική.

```
def solve

  def hanoi: rings as int, source target auxiliary as byte []

    def move: source target as byte []
      writeString: ”Moving from ”
      writeString: source
      writeString: ” to ”
      writeString: target
      writeString: ”.\n”

    if rings >= 1:
      hanoi: rings-1, source, auxiliary, target
      move: source, target
      hanoi: rings-1, auxiliary, target, source

  var numberOfRings is int

  writeString: ”Rings: ”
  numberOfRings := readInteger()
  hanoi: numberOfRings, ”left”, ”right”, ”middle”
```

3.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα Dana είναι ένα πρόγραμμα που υπολογίζει τους πρώτους αριθμούς μεταξύ 1 και n, όπου το n καθορίζεται από το χρήστη. Το πρόγραμμα αυτό χρησιμοποιεί έναν απλό αλγόριθμο για τον υπολογισμό των πρώτων αριθμών. Μια διατύπωση αυτού του αλγορίθμου σε ψευδογλώσσα δίνεται παρακάτω. Λαμβάνεται υπόψη ότι οι αριθμοί 2 και 3 είναι πρώτοι, και στη συνέχεια εξετάζονται μόνο οι αριθμοί της μορφής $6k \pm 1$, όπου k φυσικός αριθμός.

Κύριο πρόγραμμα

τύπωσε τους αριθμούς 2 και 3
για $t := 6$ μέχρι n με βήμα 6 κάνε τα εξής:
    αν ο αριθμός $t - 1$ είναι πρώτος τότε τύπωσέ τον
    αν ο αριθμός $t + 1$ είναι πρώτος τότε τύπωσέ τον

Αλγόριθμος ελέγχου (είναι ο αριθμός t πρώτος;)

αν $t < 0$ τότε έλεγξε τον αριθμό $-t$
αν $t < 2$ τότε ο t δεν είναι πρώτος
αν $t = 2$ τότε ο t είναι πρώτος
αν ο t διαιρείται με το 2 τότε ο t δεν είναι πρώτος
για $i := 3$ μέχρι $t/2$ με βήμα 2 κάνε τα εξής:
    αν ο t διαιρείται με τον i τότε ο t δεν είναι πρώτος
ο t είναι πρώτος

Το αντίστοιχο πρόγραμμα στη γλώσσα Dana είναι το ακόλουθο.

```
def main

  def prime is byte: n as int
    var i is int

    if n < 0:        return: prime(-n)
    elif n < 2:      return: false
    elif n = 2:      return: true
    elif n % 2 = 0:  return: false
    else:
      i := 3
      loop:
        if i > n / 2:    break
        elif n % i = 0:  return: false
        i := i + 2
      return: true

  var limit number counter is int

  writeString: "Limit: "
  limit := readInteger()
  writeString: "Primes:\n"
  counter := 0
  if limit >= 2:
    counter := counter + 1
    writeInteger: 2
    writeString: "\n"
  if limit >= 3:
    counter := counter + 1
    writeInteger: 3
```

```
    writeString: "\n"
  number := 5
  loop:
    if number > limit:  break
    elif prime(number):
      counter := counter + 1
      writeInteger: number
      writeString: "\n"
    number := number + 2
    if number > limit:  break
    elif prime(number):
      counter := counter + 1
      writeInteger: number
      writeString: "\n"
    number := number + 4

  writeString: "\nTotal: "
  writeInteger: counter
  writeString: "\n"
```

3.4 Αντιστροφή συμβολοσειράς

Το πρόγραμμα που ακολουθεί στη γλώσσα Dana εκτυπώνει το μήνυμα "Hello world!" αντιστρέφοντας τη δοθείσα συμβολοσειρά.

```
def main
  var r is byte [32]

  def reverse: s as byte []
    var i l is int

    l := strlen(s)
    i := 0
    loop:
      if i < l:
        r[i] := s[l-i-1]
        i := i+1
      else:  break
    r[i] := '\0'

  reverse: "\n!dlrow olleH"
  writeString: r
```

3.5 Ταξινόμηση με τη μέθοδο της φυσαλίδας

Ο αλγόριθμος της φυσαλίδας (bubble sort) είναι ένας από τους πιο γνωστούς και απλούς αλγορίθμους ταξινόμησης. Το παρακάτω πρόγραμμα σε Dana τον χρησιμοποιεί για να ταξινομήσει έναν πίνακα ακεραίων αριθμών κατ' αύξουσα σειρά. Αν x είναι ο πίνακας που πρέπει να ταξινομηθεί και n είναι το μέγεθός του (θεωρούμε σύμφωνα με τη σύμβαση της Dana ότι τα στοιχεία του είναι τα $x[0], x[1], \ldots x[n-1]$), μια παραλλαγή του αλγορίθμου περιγράφεται με ψευδοκώδικα ως εξής:

Αλγόριθμος της φυσαλίδας (bubble sort)

επανάλαβε το εξής:
    για i από 0 ως $n-2$
        αν $x[i] > x[i+1]$
            αντίστρεψε τα $x[i]$ και $x[i+1]$
όσο μεταβάλλεται η σειρά των στοιχείων του x

Το αντίστοιχο πρόγραμμα σε γλώσσα Dana είναι το εξής:

```
def main

  def bsort: n as int, x as int []

    def swap: x y as ref int
      var t is int
      t := x
      x := y
      y := t

    var changed is byte
    var i is int

    loop:
      changed := false
      i := 0
      loop:
        if i < n-1:
          if x[i] > x[i+1]:
            swap: x[i], x[i+1]
            changed := true
          i := i+1
        else:  break
      if not changed: break

  def writeArray: msg as byte [], n as int, x as int []
    var i is int

    writeString: msg
    i := 0
    loop:
      if i < n:
        if i > 0: writeString: ”, ”
        writeInteger: x[i]
        i := i+1
      else:  break
    writeString: ”\n”

  var seed i is int
  var x      is int [16]

  seed := 65
  i := 0
  loop:
    if i < 16:
      seed := (seed * 137 + 220 + i) % 101
      x[i] := seed
      i := i+1
    else:  break
  writeArray: ”Initial array: ”, 16, x
  bsort: 16, x
  writeArray: ”Sorted array: ”, 16, x
```

4 Οδηγίες για την παράδοση

Τα τελευταία χρόνια, ο πιο εύκολος τρόπος παράδοσης της εργασίας σας είναι μέσω ενός *private repository* στο GitHub στο οποίο κάνετε κάποια στιγμή collaborator τον διδάσκοντα, στέλνοντάς του την αντίστοιχη πρόσκληση, και στο οποίο αναφέρονται στο `README.md` του οι αναλυτικές οδηγίες εγκατάστασης και χρήσης του μεταγλωττιστή σας (κατά προτίμηση για κάποιο Debian-based Linux). Στη συνέχεια, το repository αυτό χρησιμοποιείται για συζήτηση πιθανών issues και διορθώσεων του κώδικα του μεταγλωττιστή σας.

Ο τελικός μεταγλωττιστής πρέπει να μπορεί να εξάγει κατά βούληση ενδιάμεσο και τελικό κώδικα. Εφόσον δεν έχουν καθορισθεί οι παράμετροι λειτουργίας `-f` ή `-i`, που εξηγούνται παρακάτω, ο μεταγλωττιστής θα δέχεται το πηγαίο πρόγραμμα από ένα αρχείο με οποιαδήποτε κατάληξη (π.χ. `*.dana`) που θα δίνεται ως το μοναδικό του όρισμα. Ο ενδιάμεσος κώδικας θα τοποθετείται σε αρχείο με κατάληξη `*.imm` και ο τελικός σε αρχείο με κατάληξη `*.asm`. Τα αρχεία αυτά θα βρίσκονται στον ίδιο κατάλογο και θα έχουν το ίδιο κυρίως όνομα. Π.χ. από το πηγαίο αρχείο `/tmp/hello.dana` θα παράγονται τα `/tmp/hello.imm` και `/tmp/hello.asm`.

Το εκτελέσιμο του τελικού μεταγλωττιστή θα πρέπει να δέχεται τις παρακάτω παραμέτρους:

`-o file` το εκτελέσιμο που παράγεται θα αποθηκεύεται στο `file`. Αν η παράμετρος αυτή δεν έχει δοθεί το εκτελέσιμο θα πρέπει να βρίσκεται στο αρχείο `./a.out` (δηλ. στο τρέχον directory από το οποίο καλείται ο μεταγλωττιστής σας).

`-O` σημαία βελτιστοποίησης (προαιρετική).

`-f` πρόγραμμα στο standard input, έξοδος τελικού κώδικα στο standard output.

`-i` πρόγραμμα στο standard input, έξοδος ενδιάμεσου κώδικα στο standard output.

Επίσης, η τιμή που θα επιστρέφεται στο λειτουργικό σύστημα από το μεταγλωττιστή θα πρέπει να είναι μηδενική στην περίπτωση επιτυχούς μεταγλώττισης και μη μηδενική σε αντίθετη περίπτωση.

Για την εύκολη ανάπτυξη του μεταγλωττιστή προτείνεται η χρήση ενός αρχείου `Makefile` με τα εξής (τουλάχιστον) χαρακτηριστικά:

- Με απλό `make`, θα δημιουργεί το εκτελέσιμο του τελικού μεταγλωττιστή.
- Με `make clean`, θα σβήνει όλα ανεξαιρέτως τα αρχεία που παράγονται αυτόματα (π.χ. αυτά που παράγουν `bison` και `flex`, τα object files) εκτός από το τελικό εκτελέσιμο.
- Με `make distclean`, θα σβήνει όλα τα παραπάνω και το τελικό εκτελέσιμο.

Ένα παράδειγμα ενός τέτοιου `Makefile`, υποθέτοντας ότι γλώσσα υλοποίησης είναι η C και γίνεται χρήση των εργαλείων `flex` και `bison`, είναι το παρακάτω.

```
CC=gcc
CFLAGS=-Wall

danac: lexer.o parser.o symbol.o # and any other files
	$(CC) $(CFLAGS) -o $@ $^ -lfl

lexer.c: lexer.l parser.h
	flex -s -o $@ $<

parser.c parser.h: parser.y
	bison -dv -o $@ $<

clean:
	$(RM) *.o lexer.c parser.c parser.h core *~

distclean: clean
	$(RM) danac
```

Η μορφή εμφάνισης των τετράδων και του τελικού κώδικα θα πρέπει να είναι αυτή που προτείνεται στο βιβλίο και στις διαλέξεις. Επίσης να ληφθούν υπόψη οι παρακάτω οδηγίες:

- Ενδιάμεσος κώδικας: να υιοθετηθεί μορφοποίηση ισοδύναμη με

```
printf("%d: %s, %s, %s, %s\n", ...)
```

- Τελικός κώδικας: προσοχή να δοθεί στη στηλοθέτηση (indentation). Να ακολουθηθεί το παρακάτω υπόδειγμα:

```
ετικέττα: <tab> εντολή <tab> όρισμα-1, όρισμα-2
          <tab> εντολή <tab> όρισμα-1, όρισμα-2
```