Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Μεταγλωττιστές 2017

Θέμα εργασίας

Η γλώσσα Grace

Grace Brewster Murray Hopper (1906-1992)

`http://eclass.uoa.gr/courses/D38/`

Διδάσκων: *Άγγελος Χαραλαμπίδης*

Βοηθοί: *Αλέξανδρος Τάσος*
Χρήστος Γαβάνας

Αθήνα, Μάρτιος 2017

Compiling in ’51, nobody believed that. I had a running compiler and nobody would touch it, because, they carefully told me, computers could only do arithmetic, they could not write programs.

– Admiral Grace Hopper

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από μια ομάδα μέχρι δύο (2) φοιτητές ενός μεταγλωττιστή για τη γλώσσα Grace. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C/C++ ή Java. Η επιλογή κάποιας άλλης γλώσσας υλοποίησης μπορεί να γίνει κατόπιν συνεννόησης. Επιτρέπεται να χρησιμοποιηθούν και επίσης συνιστάται η χρήση εργαλείων, όπως για παράδειγμα `flex`, `bison`, `SableCC`, κ.λπ.

Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή φαίνονται στον παρακάτω πίνακα.

Τμήμα του μεταγλωττιστή	*Μονάδες*	*Bonus*	*Ημ/νία παράδοσης*
Λεκτικός αναλυτής	0.5	—	7 Απριλίου 2017
Συντακτικός αναλυτής	1.5	—	7 Απριλίου 2017
Σημασιολογικός αναλυτής	1.0	—	2 Ιουνίου 2017
Ενδιάμεσος κώδικας	1.0	—	2 Ιουνίου 2017
Βελτιστοποίηση	—	1.0	την ημ/νία εξετάσεων Ιουνίου
Εκτέλεση	1.0	—	την ημ/νία εξετάσεων Ιουνίου
Συνολική εργασία	5.0	1.0	

Η εκτέλεση του προγράμματος μπορεί να γίνει με κάποιον από τους παρακάτω τρόπους:

1. παραγωγή x86 τελικού κώδικα και εκτέλεση σε επεξεργαστή x86,
2. παραγωγή Java bytecode και διερμηνεία αυτού στην JavaVM,
3. διερμηνεία ενδιάμεσου κώδικα.

Μπορείτε να επιλέξετε όποιον τρόπο σας φαίνεται πιο βολικός.

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. Παρακαλούμε, **μην παραδίδετε τυπωμένες εργασίες**! Η μορφή και τα περιεχόμενα των παραδοτέων πρέπει να συμφωνούν με τις οδηγίες που δίνονται στην ενότητα 4 του παρόντος.

Προαιρετικό τμήμα και μονάδες bonus

Το σύνολο των μονάδων της παρούσας εργασίας είναι 6. Η επιπλέον μονάδα είναι bonus (που σημαίνει ότι το σύνολο μονάδων του μαθήματος είναι 11) και αντιστοιχούν στο τμήμα της εργασίας που αντιστοιχεί στην βελτιστοποίηση ενδιάμεσου κώδικα με ανάλυση ροής δεδομένων και ελέγχου.

Περιεχόμενα

1 Περιγραφή της γλώσσας Grace

Η γλώσσα Grace 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 της C.
- Βασικοί τύποι δεδομένων για χαρακτήρες, ακέραιους αριθμούς 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 Grace χωρίζονται στις παρακάτω κατηγορίες:

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

```
and      char     div      do         else     fun     if
int      mod      not      nothing    or       ref     return
then     var      while
```

- Τα *ονόματα* αποτελούνται από ένα γράμμα του λατινικού αλφαβήτου, πιθανώς ακολουθούμενο από μια σειρά γραμμάτων, δεκαδικών ψηφίων, 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:

```
+     -     *     #     =     <     >     <=     >=
```

- Τους *διαχωριστές*, 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

```
(     )     [     ]     {     }     ,     ;     :     <-
```

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα Grace 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 και τελειώνουν με την ακολουθία χαρακτήρων `$$`. Τα σχόλια αυτής της μορφής δεν επιτρέπεται να είναι φωλιασμένα.

1.2 Τύποι δεδομένων

Η Grace υποστηρίζει δύο βασικούς τύπους δεδομένων:

- `int`: ακέραιοι αριθμοί μεγέθους 32 bit,
- `char`: χαρακτήρες.

Εκτός από τους βασικούς τύπους, η Grace υποστηρίζει επίσης τύπους πινάκων, οι οποίοι συμβολίζονται με t `[n]`, όπου t έγκυρος τύπος, και το n ακέραια σταθερά με θετική τιμή. Υποστηρίζει επίσης έμμεσα έναν τύπο λογικών εκφράσεων, ο οποίος όμως χρησιμοποιείται μόνο στις συνθήκες των εντολών `if` και `while`. Αυτός ο τύπος δεν πρέπει να συγχέεται με τους τύπους δεδομένων `int` και `char`.

1.3 Δομή του προγράμματος

Η γλώσσα Grace 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 στην ίδια εμβέλεια.

1.3.1 Μεταβλητές

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

```
var i       : int;
var x, y, z : int;
var s       : char[80];
```

1.3.2 Δομικές μονάδες

Ο *ορισμός* μίας δομικής μονάδας γίνεται γράφοντας την επικεφαλίδα της δομικής μονάδας, τις τοπικές δηλώσεις και το σώμα της. Η επικεφαλίδα αρχίζει με τη λέξη κλειδί `fun`. Στην επικεφαλίδα αναφέρεται το όνομα της δομικής μονάδας οι τυπικές της παράμετροι (προαιρετικά) και ο τύπος επιστροφής. Αν η δομική μονάδα δεν επιστρέφει αποτέλεσμα τότε ο τύπος της είναι `nothing`. Ο τύπος επιστροφής δεν μπορεί να είναι τύπος πίνακα. Οι τυπικές παράμετροι γράφονται μέσα σε παρενθέσεις.

Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομά της, τον τύπο της και τον τρόπο περάσματος. Η δήλωση των παραμέτρων μοιάζει με εκείνη των μεταβλητών. Συνεχόμενες δηλώσεις παραμέτρων με διαφορετικούς τύπους ή τρόπο περάσματος διαχωρίζονται μεταξύ τους με κόμματα (`,`). Η γλώσσα Grace υποστηρίζει πέρασμα παραμέτρων κατ' αξία (by value) και κατ' αναφορά (by reference). Αν η δήλωση ξεκινά με τη λέξη κλειδί `ref`, τότε οι παράμετροι που δηλώνονται περνούν κατ' αναφορά, διαφορετικά περνούν κατ' αξία. Οι παράμετροι τύπου πίνακα περνούν υποχρεωτικά κατ' αναφορά. Στους τύπους πίνακα που χρησιμοποιούνται για τις δηλώσεις τυπικών παραμέτρων μπορεί να παραλείπεται το μέγεθος της πρώτης διάστασης.

Ακολουθούν παραδείγματα επικεφαλίδων ορισμών δομικών μονάδων.

```
fun p1 () : nothing
fun p2 (n : int) : nothing
fun p3 (a, b : int; ref c : char) : nothing
fun f1 (x : int) : int
fun f2 (ref s : char[]) : int
fun matrix_mult(ref a, b, c : int[10][10]) : nothing
```

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

Στην περίπτωση αμοιβαία αναδρομικών υποπρογραμμάτων, το όνομα μιας δομικής μονάδας 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 αυτής της δομικής μονάδας (χωρίς τις τοπικές μεταβλητές ή το σώμα) που τερματίζεται με το διαχωριστή `;`.

1.4 Εκφράσεις και συνθήκες

Κάθε έκφραση της Grace 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 Grace περιγράφονται στην ενότητα 1.4.3. Χρησιμοποιούνται μόνο σε συνδυασμό με τις εντολές `if` και `while` και ο υπολογισμός τους δίνει ως αποτέλεσμα μια λογική τιμή (αληθή ή ψευδή).

Οι εκφράσεις 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 και οι παράμετροι των δομικών μονάδων και τα στοιχεία πινάκων. 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.

- Αν e_1 είναι μια έκφραση τύπου t `[]` και e_2 είναι μια έκφραση τύπου `int`, τότε e_1`[`e_2`]` είναι μια l-value με τύπο t. Αν η τιμή της έκφρασης e_2 είναι ο μη αρνητικός ακέραιος n τότε αυτή η l-value αντιστοιχεί στο στοιχείο με δείκτη n του πίνακα που αντιστοιχεί στην e_1. Η αρίθμηση των στοιχείων του πίνακα ξεκινά από το μηδέν. Η τιμή του n δεν πρέπει να υπερβαίνει τα πραγματικά όρια του πίνακα.

- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1 είναι l-value. Έχουν τύπο `char[`n`]` όπου n είναι ο αριθμός χαρακτήρων που περιέχονται στη συμβολοσειρά προσαυξημένος κατά ένα. Κάθε τέτοια l-value αντιστοιχεί σε έναν πίνακα, στον οποίο βρίσκονται αποθηκευμένοι οι χαρακτήρες 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.

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

1.4.2 Σταθερές

Στις r-values της γλώσσας Grace 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 `char` και η τιμή τους είναι ίση με το χαρακτήρα που παριστάνουν.

1.4.3 Τελεστές

Οι τελεστές της Grace 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. Όλοι οι τελεστές της Grace έχουν ως αποτέλεσμα r-value ή συνθήκη.

- 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.

- Ο τελεστής με ένα τελούμενο `not` υλοποιεί τη λογική άρνηση. Το τελούμενο πρέπει να είναι έκφραση τύπου `bool` και το ίδιο είναι το αποτέλεσμά του.

- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `div` και `mod` υλοποιούν τις αριθμητικές πράξεις. Τα τελούμενα πρέπει να είναι εκφράσεις τύπου `int` και το αποτέλεσμα είναι r-value του ίδιου τύπου.

- Οι τελεστές `=`, `#`, `<`, `>`, `<=` και `>=` υλοποιούν τις σχέσεις σύγκρισης μεταξύ βασικών τύπων. Τα τελούμενα πρέπει να είναι εκφράσεις του ίδιου βασικού τύπου, δηλαδή `int` ή `char` και το αποτέλεσμα είναι συνθήκη.

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της Grace.

Τελεστές	Περιγραφή	Αριθμός τελουμένων	Θέση και προσεταιριστικότητα
`+ -`	Πρόσημα	1	prefix
`* div mod`	Πολλαπλασιαστικοί τελεστές	2	infix, αριστερή
`+ -`	Προσθετικοί τελεστές	2	infix, αριστερή
`= # > < <= >=`	Σχεσιακοί τελεστές	2	infix, καμία
`not`	Λογική άρνηση	1	prefix
`and`	Λογική σύζευξη	2	infix, αριστερή
`or`	Λογική διάζευξη	2	infix, αριστερή

- Οι τελεστές `and` και `or` υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης. 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 και μόνο του πρώτου τελούμενου, το δεύτερο τελούμενο δεν αποτιμάται καθόλου.

Στον πίνακα 2 ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της Grace. 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 Grace είναι οι ακόλουθες:

- Η κενή εντολή `;` 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 η έκφραση e πρέπει να είναι του ίδιου τύπου t.
- Η εντολή κλήσης δομικής μονάδας $f(e_1, \ldots, e_n)$`;`. Οι πραγματικές παράμετροι γράφονται ανάμεσα σε παρενθέσεις. Αν είναι περισσότερες, χωρίζονται με κόμματα. Οι προϋποθέσεις για την κλήση είναι οι ίδιες όπως στην ενότητα 1.4.4 με τη διαφορά ότι ο τύπος επιστροφής της f πρέπει να είναι `nothing`.

- Η σύνθετη εντολή, που αποτελείται από μια ακολουθία έγκυρων εντολών ανάμεσα σε άγκιστρα `{` και `}`. Οι εντολές εκτελούνται διαδοχικά.

- Η εντολή ελέγχου `if` c `then` s_1 `else` s_2 όπου c πρέπει να είναι έγκυρη συνθήκη και s_1, s_2 να είναι έγκυρες εντολές. Το σκέλος `else` είναι προαιρετικό. Η σημασιολογία της εντολής είναι όπως στη C.

- Η εντολή ελέγχου `while` c `do` s, όπου c πρέπει να είναι έγκυρη συνθήκη και s μια έγκυρη εντολή. Η σημασιολογία αυτής της εντολής είναι όπως στη C.

- Η εντολή άλματος `return` e `;` που τερματίζει την εκτέλεση της τρέχουσας δομικής μονάδας και επιστρέφει ως αποτέλεσμα την τιμή της έκφρασης e. Αν η τρέχουσα δομική μονάδα έχει τύπο `nothing` τότε η e πρέπει να παραλείπεται. Διαφορετικά, 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 Βιβλιοθήκη συναρτήσεων

Η Grace υποστηρίζει ένα σύνολο προκαθορισμένων δομικών μονάδων, τις οποίες θα χρειαστεί να υλοποιήσετε στο προαιρετικό μέρος της εργασίας. 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.

1.6.1 Είσοδος και έξοδος

```
fun puti (n : int) : nothing;
fun putc (c : char) : nothing;
fun puts (ref s : char[]) : nothing;
```

Οι συναρτήσεις αυτές χρησιμοποιούνται για την εκτύπωση τιμών που ανήκουν στους βασικούς τύπους της Grace, καθώς και για την εκτύπωση συμβολοσειρών.

```
fun geti () : int;
fun getc () : char;
fun gets (n : int, ref s : char[]) : nothing;
```

Αντίστοιχα, οι παραπάνω συναρτήσεις χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της Grace και για την εισαγωγή συμβολοσειρών. Η συνάρτηση `gets` 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.

1.6.2 Συναρτήσεις μετατροπής

```
fun abs (n : int)  : int;
fun ord (c : char) : int;
fun chr (n : int)  : char;
```

Η συνάρτηση `abs` υπολογίζει την απόλυτη τιμή ενός ακέραιου αριθμού. Οι συναρτήσεις `ord` και `chr` μετατρέπουν από ένα χαρακτήρα στον αντίστοιχο κωδικό ASCII και αντίστροφα.

1.6.3 Συναρτήσεις διαχείρισης συμβολοσειρών

```
fun strlen (ref s : char[]) : int;
fun strcmp (ref s1, s2 : char[]) : int;
fun strcpy (ref trg, src : char[]) : nothing;
fun strcat (ref trg, src : char[]) : nothing;
```

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

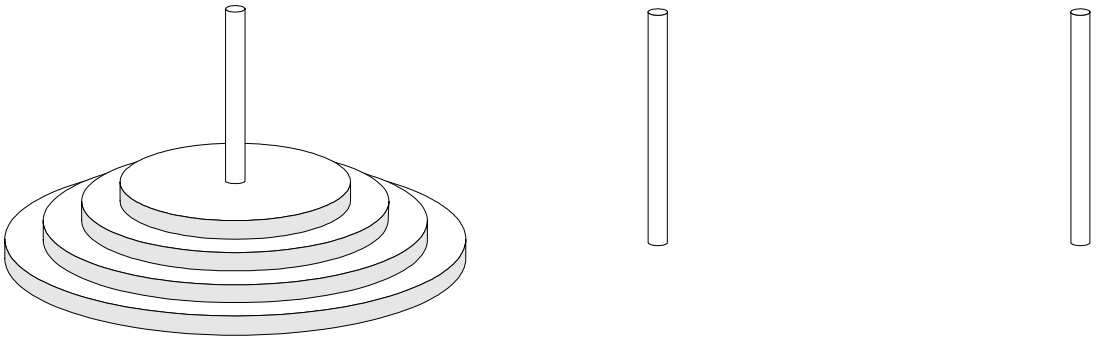
2 Πλήρης γραμματική της Grace

Η σύνταξη της γλώσσας Grace 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⟩ και ⟨string-literal⟩ είναι τερματικά σύμβολα της γραμματικής.

⟨program⟩ ::= ⟨func-def⟩
⟨func-def⟩ ::= ⟨header⟩ (⟨local-def⟩)* ⟨block⟩
⟨header⟩ ::= “**fun**” ⟨id⟩ “**(**” [⟨fpar-def⟩ (“**;**” ⟨fpar-def⟩)*] “**)**” “**:**” ⟨ret-type⟩
⟨fpar-def⟩ ::= [“**ref**”] ⟨id⟩ (“**,**” ⟨id⟩)* “**:**” ⟨fpar-type⟩
⟨data-type⟩ ::= “**int**” | “**char**”
⟨type⟩ ::= ⟨data-type⟩ (“**[**” ⟨int-const⟩ “**]**”)*
⟨ret-type⟩ ::= ⟨data-type⟩ | “**nothing**”
⟨fpar-type⟩ ::= ⟨data-type⟩ [“**[**” “**]**”] (“**[**” ⟨int-const⟩ “**]**”)*
⟨local-def⟩ ::= ⟨func-def⟩ | ⟨func-decl⟩ | ⟨var-def⟩
⟨var-def⟩ ::= “**var**” ⟨id⟩ (“**,**” ⟨id⟩)* “**:**” ⟨type⟩ “**;**”
⟨func-decl⟩ ::= ⟨header⟩ “**;**”
⟨stmt⟩ ::= “**;**” | ⟨l-value⟩ “**<-**” ⟨expr⟩ “**;**” | ⟨block⟩ | ⟨func-call⟩ “**;**”
| “**if**” ⟨cond⟩ “**then**” ⟨stmt⟩ [“**else**” ⟨stmt⟩]
| “**while**” ⟨cond⟩ “**do**” ⟨stmt⟩ | “**return**” [⟨expr⟩] “**;**”
⟨block⟩ ::= “**{**” (⟨stmt⟩)* “**}**”
⟨func-call⟩ ::= ⟨id⟩ “**(**” [⟨expr⟩ (“**,**” ⟨expr⟩)*] “**)**”
⟨l-value⟩ ::= ⟨id⟩ | ⟨string-literal⟩ | ⟨l-value⟩ “**[**” ⟨expr⟩ “**]**”
⟨expr⟩ ::= ⟨int-const⟩ | ⟨char-const⟩ | ⟨l-value⟩ | ⟨func-call⟩ | “**(**” ⟨expr⟩ “**)**”
| (“**+**” | “**-**”) ⟨expr⟩ | ⟨expr⟩ (“**+**” | “**-**” | “*****” | “**div**” | “**mod**”) ⟨expr⟩
⟨cond⟩ ::= “**(**” ⟨cond⟩ “**)**” | “**not**” ⟨cond⟩ | ⟨cond⟩ (“**and**” | “**or**”) ⟨cond⟩
| ⟨expr⟩ (“**=**” | “**#**” | “**<**” | “**>**” | “**<=**” | “**>=**”) ⟨expr⟩

3 Παραδείγματα

Στην παράγραφο αυτή δίνονται έξι παραδείγματα προγραμμάτων στη γλώσσα Grace, η πολυπλοκότητα των οποίων κυμαίνεται σημαντικά.



Σχήμα 1: Οι πύργοι του Hanoi.

3.1 Πες γεια!

Το παρακάτω παράδειγμα είναι το απλούστερο πρόγραμμα στη γλώσσα Grace που παράγει κάποιο αποτέλεσμα ορατό στο χρήστη. Το πρόγραμμα αυτό τυπώνει απλώς ένα μήνυμα.

```
fun hello () : nothing
{
  puts(”Hello world!\n”);
}
```

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 Grace που λύνει αυτό το πρόβλημα δίνεται παρακάτω. Η συνάρτηση `hanoi` είναι αναδρομική.

```
fun solve () : nothing
      fun hanoi (rings : int; ref source, target, auxiliary : char[]) : nothing
         fun move (ref source, target : char[]) : nothing
         {
            puts(”Moving from ”);
            puts(source);
            puts(” to ”);
            puts(target);
            puts(”.\n”);
         }
      {
         if rings >= 1 then {
            hanoi(rings-1, source, auxiliary, target);
```

```
            move(source, target);
            hanoi(rings-1, auxiliary, target, source);
          }
      }

      var NumberOfRings : int;
{
  puts(”Rings: ”);
  NumberOfRings <- geti();
  hanoi(NumberOfRings, ”left”, ”right”, ”middle”);
}
```

3.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα Grace 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 Grace είναι το ακόλουθο.

```
fun main () : nothing
    fun prime (n : int) : int
      var i : int;
    {
      if n<0              then return prime(-1);
      else if n<2         then return 0;
      else if n=2         then return 1;
      else if n mod 2 = 0 then return 0;
      else {
        i <- 3;
        while i <= n div 2 do {
          if n mod i = 0 then
            return 0;
          i <- i + 2;
        }
        return 1;
      }
    }
```

```
    var limit, number, counter : int;

{
   puts(”Limit: ”);
   limit <- geti();
   puts(”Primes:\n”);
   counter <- 0;
   if limit >= 2 then {
      counter <- counter + 1;
      puti(2);
      puts(”\n”);
   }
   if limit >= 3 then {
      counter <- counter + 1;
      puti(3);
      puts(”\n”);
   }
   number <- 6;
   while number <= limit do {
      if prime(number - 1) = 1 then {
         counter <- counter + 1;
         puti(number - 1);
         puts(”\n”);
      }
      if number # limit and prime(number + 1) = 1 then {
         counter <- counter + 1;
         puti(number + 1);
         puts(”\n”);
      }
      number <- number + 6;
   }
   puts(”\nTotal: ”);
   puti(counter);
   puts(”\n”);
}
```

3.4 Αντιστροφή συμβολοσειράς

Το πρόγραμμα που ακολουθεί στη γλώσσα Grace εκτυπώνει το μήνυμα “Hello world!” αντιστρέφοντας τη δοθείσα συμβολοσειρά.

```
fun main () : nothing
  var r : char[32];

  fun reverse (ref s : char[]) : nothing
    var i, l : int;
  {
    l <- strlen(s);
    i <- 0;
    while i < l do {
      r[i] <- s[l-i-1];
      i <- i+1;
    }
    r[i] <- ’\0’;
  }

{
  reverse(”\n!dlrow olleH”);
```

```
  puts(r);
}
```

3.5 Ταξινόμηση πίνακα 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 Grace 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 Grace 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 Grace είναι το εξής:

```
fun main () : nothing

   fun bsort (n : int; ref x : int[]) : nothing

      fun swap (ref x, y : int) : nothing
         var t : int;
      {
         t <- x;
         x <- y;
         y <- t;
      }

      var changed, i : int;
   {
      changed <- 1;
      while changed > 0 do {
        changed <- 0;
        i <- 0;
        while i < n-1 do {
          if x[i] > x[i+1] then {
            swap(x[i],x[i+1]);
            changed <- 1;
          }
          i <- i+1;
        }
      }
   }

   fun putArray (ref msg : char[]; n : int; ref x : int[]) : nothing
      var i : int
   {
      puts(msg);
      i <- 0;
      while i < n do {
        if i > 0 then puts(", ");
        puti(x[i]);
        i <- i+1;
```

```
      }
      puts(”\n”);
   }

   var seed, i : int;
   var x : int[16];
{
  seed <- 65;
  i <- 0;
  while i < 16 do {
    seed <- (seed * 137 + 220 + i) mod 101;
    x[i] <- seed;
    i <- i+1;
  }
  putArray(”Initial array: ”, 16, x);
  bsort(16,x);
  putArray(”Sorted array: ”, 16, x);
}
```

4 Οδηγίες για την παράδοση

Ο μεταγλωττιστής θα δέχεται το πηγαίο πρόγραμμα από ένα αρχείο ή από την προκαθορισμένη είσοδο (stdin). Το αρχείο μπορεί να έχει οποιαδήποτε κατάληξη (πχ. `*.grace`) και θα δίνεται ως όρισμα στον μεταγλωττιστή.

Η έξοδος του μεταγλωττιστή εξαρτάται από την επιλογή που θα ακολουθήσετε για το τμήμα της εκτέλεσης. Σε κάθε περίπτωση θα πρέπει να αναφέρεται ο τρόπος χρήσης στο αρχείο `README` που θα συνοδεύει τον κώδικά σας.

Σε περίπτωση που δεν υλοποιήσετε όλα τα τμήματα της εργασίας θα πρέπει να παρέχετε σαφείς οδηγίες και εκτελέσιμο πρόγραμμα που θα μπορεί να επιδεικνύει την λειτουργία των υλοποιημένων τμημάτων του μεταγλωττιστή σας.

Ελέγξτε την λειτουργία του μεταγλωττιστή σας σε περισσότερα παραδείγματα προγραμμάτων από αυτά που δίνονται στο παρόν έγγραφο. Ο μεταγλωττιστής σας πρέπει να επιτυγχάνει σε έγκυρα προγράμματα της Grace και να αποτυγχάνει σε μη έγκυρα.

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. Σε περίπτωση που επιλέξετε να διερμηνεύετε τον ενδιάμεσο κώδικα, προνοήστε να υπάρχει επιλογή μόνο μεταγλώττιση και εμφάνιση του ενδιάμεσου κώδικα και όχι για εκτέλεση.

Η εργασία σας θα διορθωθεί σε σύστημα linux με 32-bit x86 επεξεργαστή. Αν χρειάζονται επιπλέον εργαλεία για την μεταγλώττιση και εκτέλεση της εργασίας θα πρέπει να περιγράφονται στο αρχείο `README` και να είναι ευρέως διαθέσιμα σε διανομές linux. Σε περίπτωση που ο μεταγλωττιστής σας βασίζεται σε επιπλέον βιβλιοθήκες είτε θα τις παρέχετε μαζί με τον πηγαίο κώδικα, είτε θα πρέπει να τις χειρίζεται το εργάλειο μεταγλώττισης που θα χρησιμοποιήσετε (βλέπε ενότητα 4.2).

4.1 Δομή της εργασίας

- Θα πρέπει να συμπεριλάβετε ένα UTF-8 αρχείο `README` με λίγα λόγια για το συγκεκριμένο πρόγραμμα, οδηγίες για την μεταγλώττιση και την εκτέλεση κάποιον παραδειγμάτων. Προτείνεται να ακολουθείτε τον φορμαλισμό Markdown.
- Θα πρέπει να συμπεριλάβετε ένα UTF-8 αρχείο `AUTHORS` στον κατάλογο ρίζα της εργασίας σας που θα περιέχει τους φοιτητές που ασχολήθηκαν με την συγκεκριμένη εργασία. Κάθε γραμμή θα περιέχει το όνομα και το πανεπιστημιακό email του φοιτητή μεταξύ των συμβόλων < και >. Για παράδειγμα κάθε γραμμή θα είναι της μορφής

  ```
  John Doe <sdi1234@di.uoa.gr>
  ```

- Θα πρέπει να συμπεριλάβετε έναν τρόπο μεταγλώττισης συνολικά της εργασίας. Λεπτομέρειες στην ενότητα 4.2.

4.2 Μεταγλώττιση της εργασίας

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 και το τελικό εκτελέσιμο).

Σε περίπτωση που υλοποιήσετε την εργασία σας σε Java μπορείτε να χρησιμοποιήσετε το εργαλείο `maven` ή `gradle`.

Για την εύκολη συνεργασία μεταξύ των μελών της ομάδας προτείνεται και ενθαρρύνεται η χρήση κάποιου εργαλείου version control (π.χ. `git`, `svn`, `mercurial`).