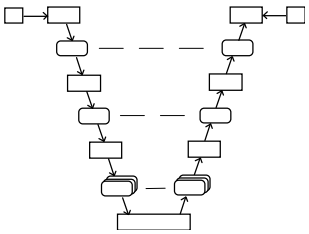
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Μεταγλωττιστές 2023

Θέμα εργασίας

Η γλώσσα Grace

Grace Murray Hopper (1906–1992), Rear Admiral, U.S. Navy



Μεταγλωττιστές

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

Διδάσκων: Κωστής Σαγώνας

Αθήνα, Φεβρουάριος 2023

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από κάθε ομάδα, το πολύ δυο σπουδαστών, ένας μεταγλωττιστής για τη γλώσσα Grace. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C/C++, 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 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:

- (50%) Βελτιστοποίηση ενδιάμεσου κώδικα, με ανάλυση ροής δεδομένων και ελέγχου.
- (50%) Δέσμευση καταχωρητών και βελτιστοποίηση τελικού κώδικα.

Περιεχόμενα

1 Περιγραφή της γλώσσας Grace

Η γλώσσα Grace είναι μια απλή γλώσσα προστακτικού προγραμματισμού. (Η αισθητική της συγχωρείται αν κανείς αναλογισθεί τον τρόπο σχεδίασής της.) Τα κύρια χαρακτηριστικά της εν συντομία είναι τα εξής:

- Απλή δομή και σύνταξη εντολών και εκφράσεων που μοιάζει με αυτές της Pascal και 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*, 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 πρέπει να αρχίζουν και να τελειώνουν στην ίδια γραμμή 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 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 (πιθανά) πολλών γραμμών*, τα οποία αρχίζουν και τελειώνουν με την ακολουθία χαρακτήρων `$$`. Τα σχόλια αυτής της μορφής δεν επιτρέπεται να είναι φωλιασμένα.
- *Σχόλια μιας γραμμής*, τα οποία αρχίζουν με το χαρακτήρα `$`, δεν ακολουθούνται από άλλο ένα `$` (διότι τότε θεωρείται ότι τότε αρχίζει ένα σχόλιο πιθανά πολλών γραμμών), και τερματίζονται με το τέλος της τρέχουσας γραμμής.

1.2 Τύποι δεδομένων

Η Grace υποστηρίζει δύο βασικούς τύπους δεδομένων:

- `int`: ακέραιοι αριθμοί μεγέθους τουλάχιστον 16 bit (-32768 έως 32767), και
- `char`: χαρακτήρες.

Εκτός από τους βασικούς τύπους, η Grace 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 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, 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, τις τοπικές δηλώσεις και το σώμα της. Η επικεφαλίδα αρχίζει με τη λέξη κλειδί `fun`. Σε αυτήν αναφέρεται το όνομα της δομικής μονάδας, οι τυπικές της παράμετροι (προαιρετικά) και ο τύπος επιστροφής. Ο τύπος επιστροφής των δομικών μονάδων που δεν επιστρέφουν αποτέλεσμα (πρβλ. διαδικασίες στην Pascal) είναι `nothing`. Ο τύπος επιστροφής δεν μπορεί να είναι τύπος πίνακα. Στη συνέχεια θα ονομάζουμε *διαδικασίες* τις δομικές μονάδες με τύπο επιστροφής `nothing` και *συναρτήσεις* τις υπόλοιπες δομικές μονάδες.

Η σύνταξη των τυπικών παραμέτρων μοιάζει με αυτήν της Pascal. Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομά της, τον τύπο της και τον τρόπο περάσματος. 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 δομικών μονάδων, 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 δηλώσεις ή το σώμα της) 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-value, οι οποίες περιγράφονται στην Ενότητα 1.4.1 και αυτές που δίνουν r-value, 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: μια l-value εμφανίζεται στο αριστερό μέλος της ανάθεσης ενώ μια r-value 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 και οι συνθήκες μπορούν να εμφανίζονται μέσα σε παρενθέσεις, που χρησιμοποιούνται για λόγους ομαδοποίησης.

1.4.1 L-value

Οι l-value 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 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-value. Έχουν τύπο `char [`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 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 στην Grace.
- 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-value 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 το 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. 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.

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
`= # > < <= >=`	Τελεστές σύγκρισης	2	infix, καμία
`not`	Λογική άρνηση	1	prefix
`and`	Λογική σύζευξη	2	infix, αριστερή
`or`	Λογική διάζευξη	2	infix, αριστερή

- Ο τελεστής με ένα τελούμενο `not` υλοποιεί τη λογική άρνηση. Το τελούμενό του πρέπει να είναι συνθήκη 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 αριθμών. Τα τελούμενα πρέπει να είναι εκφράσεις του ίδιου τύπου `int` ή `char` και το αποτέλεσμα είναι συνθήκη.
- 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 συναρτήσεων

Αν f είναι το όνομα μιας συνάρτησης με τύπο επιστροφής t (όχι `nothing`), 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 ή έκφραση e πρέπει να είναι του ίδιου τύπου t.
- Η σύνθετη εντολή, που αποτελείται από μια σειρά έγκυρων εντολών ανάμεσα σε άγκιστρα `{` και `}`. Οι εντολές αυτές εκτελούνται διαδοχικά.
- Η εντολή κλήσης διαδικασίας $f(e_1, \ldots, e_n)$`;` με τις ίδιες προϋποθέσεις όπως στην Ενότητα 1.4.4, με τη διαφορά όμως ότι ο τύπος επιστροφής της f πρέπει να είναι `nothing`.
- Η εντολή ελέγχου `if` c `then` s_1 `else` s_2, όπου c πρέπει να είναι μια έγκυρη συνθήκη και s_1, s_2 να είναι έγκυρες εντολές. Το τμήμα `else` είναι προαιρετικό. Η σημασιολογία αυτής της εντολής είναι όπως στη C.
- Η εντολή ελέγχου `while` c `do` s, όπου c πρέπει να είναι μια 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 την τιμή της e ως αποτέλεσμα. Αν η τρέχουσα δομική μονάδα έχει τύπο επιστροφής `nothing` τότε η έκφραση e πρέπει να παραλείπεται. Διαφορετικά, η έκφραση e πρέπει να έχει τύπο ίδιο με τον τύπο επιστροφής της τρέχουσας δομικής μονάδας.

1.6 Βιβλιοθήκη χρόνου εκτέλεσης

Η Grace 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.

1.6.1 Είσοδος και έξοδος

```
fun writeInteger (n : int)        : nothing;
fun writeChar    (c : char)       : nothing;
fun writeString  (ref s : char[]) : nothing;
```

Αυτές οι δομικές μονάδες χρησιμοποιούνται για την εκτύπωση τιμών που ανήκουν στους βασικούς τύπους της Grace, καθώς και για την εκτύπωση συμβολοσειρών.

```
fun readInteger ()                          : int;
fun readChar    ()                          : char;
fun readString  (n : int; ref s : char[]) : nothing;
```

Αντίστοιχα, οι παραπάνω δομικές μονάδες χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της Grace και για την εισαγωγή συμβολοσειρών. Η διαδικασία `readString` χρησιμοποιείται για την ανάγνωση μιας συμβολοσειράς μέχρι τον επόμενο χαρακτήρα αλλαγής γραμμής. Οι παράμετροί 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 ascii (c : char) : int;
fun chr   (n : int)  : char;
```

Η πρώτη επιστρέφει τον ASCII κωδικό ενός χαρακτήρα. Η δεύτερη μετατρέπει μια τιμή τύπου `int`, που πρέπει να περιέχει τον ASCII κωδικό ενός χαρακτήρα, σε αυτόν το χαρακτήρα.

1.6.3 Διαχείριση 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;
```

Αυτές οι δομικές μονάδες 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⟩

⟨func-decl⟩ ::= ⟨header⟩ "**;**"

⟨var-def⟩ ::= "**var**" ⟨id⟩ ("**,**" ⟨id⟩)* "**:**" ⟨type⟩ "**;**"

⟨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⟩ | "**(**" ⟨expr⟩ "**)**" | ⟨func-call⟩
| ("**+**" | "**-**") ⟨expr⟩ | ⟨expr⟩ ("**+**" | "**-**" | "*****" | "**div**" | "**mod**") ⟨expr⟩

⟨cond⟩ ::= "**(**" ⟨cond⟩ "**)**" | "**not**" ⟨cond⟩ | ⟨cond⟩ ("**and**" | "**or**") ⟨cond⟩
| ⟨expr⟩ ("**=**" | "**#**" | "**<**" | "**>**" | "**<=**" | "**>=**") ⟨expr⟩

3 Παραδείγματα

Στην παράγραφο αυτή δίνονται πέντε παραδείγματα προγραμμάτων στη γλώσσα Grace, 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 προγράμματα στη γλώσσα Grace που παράγει κάποιο αποτέλεσμα ορατό στο χρήστη. Το πρόγραμμα αυτό τυπώνει απλώς ένα μήνυμα.

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

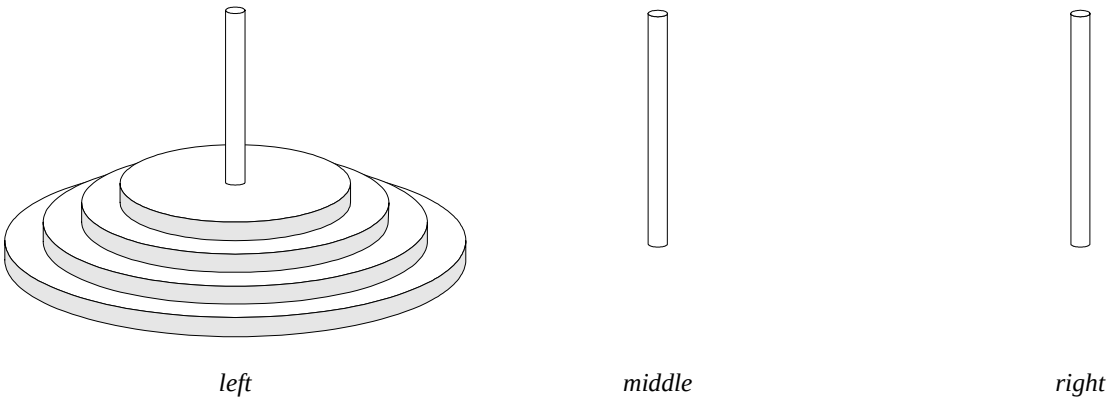
3.2 Οι πύργοι του Hanoi

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 στην αρχή της επόμενη σελίδας. Η διαδικασία `hanoi` είναι αναδρομική.



Σχήμα 1: Οι πύργοι του Hanoi.

```
fun solve () : nothing

   fun hanoi (rings : int; ref source, target, auxiliary : char[]) : nothing

      fun move (ref source, target : char[]) : nothing
      {
         writeString(”Move from ”);
         writeString(source);
         writeString(” to ”);
         writeString(target);
         writeString(”.\n”);
      }

   { $ hanoi
      if rings >= 1 then {
         hanoi(rings-1, source, auxiliary, target);
         move(source, target);
         hanoi(rings-1, auxiliary, target, source);
      }
   } $ hanoi

   var NumberOfRings : int;

{ $ solve
   writeString(”Please, give the number of rings: ”);
   NumberOfRings <- readInteger();
   writeString(”\nHere is the solution:\n\n”);
   hanoi(NumberOfRings, ”left”, ”right”, ”middle”);
} $ solve
```

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(-n);
      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;

{ $ main
   writeString(”Limit: ”);
   limit <- readInteger();
   writeString(”Primes:\n”);
   counter <- 0;
   if limit >= 2 then {
      counter <- counter + 1;
      writeString(”2\n”);
   }
   if limit >= 3 then {
      counter <- counter + 1;
      writeString(”3\n”);
   }
   number <- 6;
   while number <= limit do {
      if prime(number - 1) = 1 then {
         counter <- counter + 1;
         writeInteger(number - 1);
         writeString(”\n”);
      }
      if number # limit and prime(number + 1) = 1 then {
         counter <- counter + 1;
         writeInteger(number + 1);
         writeString(”\n”);
      }
      number <- number + 6;
   }

   writeString(”\nTotal: ”);
   writeInteger(counter);
   writeString(”\n”);
} $ main
```

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[20];

   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';
   }

{ $ main
   reverse("\n!dlrow olleH");
   writeString(r);
} $ main
```

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 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;
```

```
   { $ bsort
      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;
         }
      }
   } $ bsort

   fun writeArray (ref msg : char[]; n : int; ref x : int[]) : nothing
      var i : int;
   {
      writeString(msg);
      i <- 0;
      while i < n do {
         if i > 0 then writeString(", ");
         writeInteger(x[i]);
         i <- i+1;
      }
      writeString("\n");
   }

   var seed, i : int;
   var x       : int[16];

{ $ main
   seed <- 65;
   i <- 0;
   while i < 16 do {
      seed <- (seed * 137 + 221 + i) mod 101;
      x[i] <- seed;
      i <- i+1;
   }
   writeArray("Initial array: ", 16, x);
   bsort(16, x);
   writeArray("Sorted  array: ", 16, x);
} $ main
```

4 Οδηγίες για την παράδοση

Ο τελικός μεταγλωττιστής θα 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 (πχ. `*.grc`) 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.grc` θα παράγονται τα `/tmp/hello.imm` και `/tmp/hello.asm`.

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

`-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 `makeclean`, θα σβήνει όλα ανεξαιρέτως τα αρχεία που παράγονται αυτόματα (π.χ. αυτά που παράγουν `bison` και `flex`, τα object files, αλλά όχι το τελικό εκτελέσιμο).
- Με `makedistclean`, θα κάνει ό,τι και το `makeclean` αλλά θα σβήνει 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

compiler: lexer.o parser.o symbol.o general.o error.o symbol.o
        $(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 parser.c parser.h lexer.c core *~

distclean: clean
        $(RM) compiler
```

Η μορφή εμφάνισης των τετράδων και του τελικού κώδικα θα πρέπει να είναι αυτή που προτείνεται στα παραδείγματα. 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
  ```