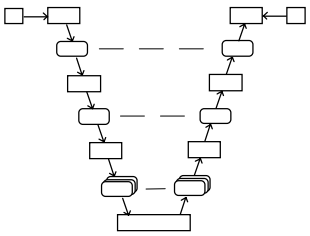
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Μεταγλωττιστές 2004
Θέμα εργασίας

Η γλώσσα Edsger



Μεταγλωττιστές

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

Διδάσκων: Νίκος Παπασπύρου

Βοηθοί: Κώστας Ταβερναράκης
Άγγελος Μανουσαρίδης
Κυριάκος Γκίνης
Μιχάλης Παπακυριάκου
Θοδωρής Τσόκος

Αθήνα, Μάρτιος 2004

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από κάθε ομάδα δυο σπουδαστών ένας μεταγλωττιστής για τη γλώσσα Edsger. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C++ (που συνιστάται), Java ή C. Η επιλογή κάποιας άλλης γλώσσας υλοποίησης μπορεί να γίνει κατόπιν συνεννόησης με το διδάσκοντα. Επιτρέπεται να χρησιμοποιηθούν και επίσης συνιστάται η χρήση εργαλείων, π.χ. `lex` / `flex`, `yacc` / `bison`, `ELI`, κ.λπ. Περισσότερες πληροφορίες σχετικές με αυτά τα εργαλεία θα δοθούν στις παραδόσεις.

Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή και οι αντίστοιχες ημερομηνίες παράδοσης φαίνονται στον παρακάτω πίνακα:

Τμήμα του μεταγλωττιστή	*Παράδοση*	*Μονάδες*	*Bonus*
Λεκτικός αναλυτής	21 Απριλίου	0.5	0.2
Συντακτικός αναλυτής	5 Μαΐου	0.5	0.1
Σημασιολογικός αναλυτής	19 Μαΐου	1.0	0.2
Ενδιάμεσος κώδικας	2 Ιουνίου	1.5	0.2
Βελτιστοποίηση	–	–	1.0
Τελικός κώδικας	16 Ιουνίου	1.5	0.3
Συνολική εργασία και έκθεση	ως την εξέταση	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. 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

Λόγω του βαθμού δυσκολίας της υλοποίησης του μεταγλωττιστή της Edsger, 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 σε τμήματα της εργασίας που είναι προαιρετικά. Συγκεκριμένα:

- Μία (1) μονάδα bonus αντιστοιχεί στην υλοποίηση φάσης βελτιστοποίησης ενδιάμεσου ή/και τελικού κώδικα.
- Μία (1) μονάδα bonus, που κατανέμεται στις φάσεις της λεκτικής ανάλυσης (20%), συντακτικής ανάλυσης (10%), σημασιολογικής ανάλυσης (20%), ενδιάμεσου κώδικα (20%) και τελικού κώδικα (30%), αντιστοιχούν στην υλοποίηση των παρακάτω χαρακτηριστικών της γλώσσας Edsger, τα οποία είναι **προαιρετικά**:
  1. (45%) Πραγματικοί αριθμοί (`double`).
  2. (35%) Μετατροπές τύπων (type casting).
  3. (20%) Οδηγία `#include` για αρχεία επικεφαλίδων.

Περιεχόμενα

1 Περιγραφή της γλώσσας Edsger

Η γλώσσα Edsger είναι μια απλή γλώσσα προστακτικού προγραμματισμού. Βασίζεται σε ένα γνήσιο υποσύνολο της ISO C99, είναι όμως εμπλουτισμένη με μερικές μικρές παραλλαγές. Τα κύρια χαρακτηριστικά της εν συντομία είναι τα εξής:

- Σύνταξη παρόμοια σε γενικές γραμμές με αυτή της C.
- Δομημένες συναρτήσεις με τους κανόνες εμβέλειας της Pascal.
- Βασικοί τύποι δεδομένων για ακέραιους αριθμούς, χαρακτήρες, λογικές τιμές και πραγματικούς αριθμούς.
- Τύποι δεικτών και (έμμεσα) πίνακες, συνδεδεμένοι σημασιολογικά όπως στη C.
- Βιβλιοθήκη συναρτήσεων.

Περισσότερες λεπτομέρειες της γλώσσας δίνονται στις παραγράφους που ακολουθούν. Λόγω των πολλών ομοιοτήτων της Edsger με τη γλώσσα C, τόσο από πλευράς σύνταξης όσο και από πλευράς σημασιολογίας, η περιγραφή θα είναι σύντομη με εξαίρεση τα σημεία όπου οι δυο γλώσσες διαφέρουν.

1.1 Λεκτικές μονάδες

Οι λεκτικές μονάδες της γλώσσας Edsger χωρίζονται στις παρακάτω κατηγορίες:

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

```
bool      break    byref    char     continue    delete
double    else     for      false    if          int
new       NULL     return   true     void
```

- Τα *ονόματα*, τα οποία αποτελούνται από ένα πεζό ή κεφαλαίο γράμμα του λατινικού αλφαβήτου, πιθανώς ακολουθούμενο από μια σειρά πεζών ή κεφαλαίων γραμμάτων, δεκαδικών ψηφίων ή χαρακτήρων υπογράμμισης (underscore). Τα ονόματα δεν πρέπει να συμπίπτουν με τις λέξεις κλειδιά που αναφέρθηκαν παραπάνω.
- Οι *ακέραιες σταθερές* χωρίς πρόσημο, που αποτελούνται από ένα ή περισσότερα δεκαδικά ψηφία. Παραδείγματα ακέραιων σταθερών είναι τα ακόλουθα:

```
0      42      1284      00200
```

- Οι *πραγματικές σταθερές* χωρίς πρόσημο, που αποτελούνται από ένα ακέραιο μέρος, ένα κλασματικό μέρος και ένα προαιρετικό εκθετικό μέρος. Το ακέραιο μέρος αποτελείται από ένα ή περισσότερα δεκαδικά ψηφία. Το κλασματικό μέρος αποτελείται από το χαρακτήρα `.` της υποδιαστολής, ακολουθούμενο από ένα ή περισσότερα δεκαδικά ψηφία. Τέλος, το εκθετικό μέρος αποτελείται από το πεζό ή κεφαλαίο γράμμα `E`, ένα προαιρετικό πρόσημο `+` ή `-` και ένα ή περισσότερα δεκαδικά ψηφία. Παραδείγματα πραγματικών σταθερών είναι τα ακόλουθα:

```
42.0      4.2e1      0.420e+2      42000.0e-3
```

- Οι *σταθεροί 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:

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

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

- Οι *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"          "Route 66"          "Hello world!\n"
     "Name:\t\"Douglas Adams\"\nValue:\t42\n"
```

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

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

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

```
;      (      )      [      ]      {      }
```

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα Edsger 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 την ακολουθία χαρακτήρων `//` και τερματίζονται με το τέλος της τρέχουσας γραμμής.

- *Σχόλια πολλών γραμμών*, τα οποία αρχίζουν με την ακολουθία χαρακτήρων `/*` και τερματίζονται με την πρώτη μετέπειτα εμφάνιση της ακολουθίας χαρακτήρων `*/`. Κατά συνέπεια, τα σχόλια δεν επιτρέπεται να είναι φωλιασμένα. Στο εσωτερικό τους επιτρέπεται η εμφάνιση οποιουδήποτε χαρακτήρα.

1.2 Τύποι δεδομένων

Η Edsger υποστηρίζει τέσσερις βασικούς τύπους δεδομένων:

- `int` : ακέραιοι αριθμοί,
- `char` : χαρακτήρες,
- `bool` : λογικές τιμές, και
- `double` : πραγματικοί αριθμοί.

Οι τύποι `int` και `double` ονομάζονται *αριθμητικοί* τύποι.

Εκτός από τους βασικούς τύπους, η Edsger υποστηρίζει επίσης τύπους δεικτών, που συμβολίζονται με $t*$, όπου ο t πρέπει να είναι έγκυρος τύπος. Κάθε δείκτης τύπου $t*$ μπορεί να χρησιμοποιηθεί ως ένας μονοδιάστατος πίνακας, αποτελούμενος από στοιχεία τύπου t. Στην περίπτωση αυτή, η αρίθμηση των στοιχείων του πίνακα ακολουθεί τη σύμβαση της C: το πρώτο στοιχείο βρίσκεται στη θέση 0, το δεύτερο στη θέση 1, ενώ το τελευταίο βρίσκεται σε θέση κατά ένα μικρότερη από το πλήθος των στοιχείων του πίνακα.

Ο αριθμός των bytes που καταλαμβάνουν τα δεδομένα κάθε τύπου στη μνήμη του υπολογιστή, καθώς και ο τρόπος παράστασης των δεδομένων περιγράφεται παρακάτω. Η περιγραφή διευκολύνει την υλοποίηση μεταγλωττιστών για υπολογιστές βασισμένους σε επεξεργαστές της σειράς x86 της Intel, με το μοντέλο προγραμμάτων COM.

- `int` : μέγεθος 2 bytes, παράσταση συμπληρώματος ως προς 2.
- `char` : μέγεθος 1 byte, παράσταση σύμφωνα με τον πίνακα ASCII.
- `bool` : μέγεθος 1 byte, τιμή 0 για `false` και $\neq 0$ για `true`.
- `double` : μέγεθος 10 bytes, παράσταση σύμφωνα με το πρότυπο IEEE 754.
- $t*$: μέγεθος 2 bytes.

1.3 Δομή του προγράμματος

Ένα πρόγραμμα Edsger αποτελείται από μια ή περισσότερες δηλώσεις. Οι δηλώσεις ανήκουν σε τρεις κατηγορίες:

- Δηλώσεις μεταβλητών (variable declarations),
- Ορισμοί συναρτήσεων (function definitions).
- Δηλώσεις συναρτήσεων (function declarations), οι ορισμοί των οποίων θα ακολουθήσουν.

Σε κάθε πρόγραμμα πρέπει να υπάρχει ο ορισμός μιας συνάρτησης με επικεφαλίδα:

```
void main ()
```

από την οποία ξεκινά η εκτέλεση του προγράμματος.

1.3.1 Μεταβλητές

Οι δηλώσεις μεταβλητών γίνονται με αναγραφή του τύπου, ακολουθούμενου από έναν ή περισσότερους *δηλωτές* (declarators) μεταβλητών. Η απλούστερη μορφή δηλωτή είναι ένα αναγνωριστικό (identifier), που αντιστοιχεί στο όνομα της μεταβλητής. Παραδείγματα τέτοιων δηλώσεων είναι:

```
int i;
double x, y;
int * p, q;
```

Ας σημειωθεί ότι, αντίθετα με τη γλώσσα C, στην τελευταία δήλωση και οι δύο μεταβλητές `p` και `q` έχουν τύπο `int *`.

Για τη διευκόλυνση της δήλωσης μεταβλητών τύπου δείκτη που χρησιμοποιούνται ως μονοδιάστατοι πίνακες, η γλώσσα Edsger υποστηρίζει ένα δεύτερο τύπο δηλωτή που αποτελείται από ένα αναγνωριστικό και την επιθυμητή διάσταση του πίνακα μέσα σε αγκύλες. Παραδείγματα τέτοιων δηλώσεων είναι:

```
int a[100];
double x[5], y[7];
int * p[12];
```

Στις παραπάνω δηλώσεις, η μεταβλητή `a` έχει τύπο `int *` (δείκτης σε ακέραιο, δηλαδή ισοδύναμα μονοδιάστατος πίνακας ακεραίων), οι μεταβλητές `x` και `y` έχουν τύπο `double *` ενώ η μεταβλητή `p` έχει τύπο `int **` (δείκτης σε δείκτη σε ακέραιο, δηλαδή ισοδύναμα μονοδιάστατος πίνακας δεικτών σε ακέραιο).

Οι μεταβλητές τύπου δείκτη που δηλώνονται με δηλωτές της δεύτερης μορφής ονομάζονται *μόνιμες* (immutable) και η τιμή τους δεν μπορεί να μεταβληθεί κατά την εκτέλεση του προγράμματος. Για παράδειγμα, το παρακάτω είναι σφάλμα:

```
int a[100], b[200];

a = b;  // wrong!
```

1.3.2 Συναρτήσεις

Κάθε συνάρτηση είναι μια δομική μονάδα και αποτελείται από την επικεφαλίδα της και το σώμα της. Στην επικεφαλίδα αναφέρεται ο τύπος επιστροφής της συνάρτησης, το όνομά της και οι τυπικές της παράμετροι μέσα σε παρενθέσεις. Οι παρενθέσεις είναι υποχρεωτικές ακόμα και μια συνάρτηση δεν έχει τυπικές παραμέτρους. Ο τύπος αποτελέσματος `void` υποδηλώνει ότι μια συνάρτηση δεν επιστρέφει αποτέλεσμα.

Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομά της, τον τύπο της και τον τρόπο περάσματος. Σε αντίθεση με τη C, η γλώσσα Edsger υποστηρίζει πέρασμα παραμέτρων κατ' αξία (by value) και κατ' αναφορά (by reference). Αν στη δήλωση μιας τυπικής παραμέτρου έχει προηγηθεί η λέξη κλειδί `byvar`, τότε αυτή περνά κατ' αναφορά, διαφορετικά περνά κατ' αξία.

Ακολουθούν παραδείγματα επικεφαλίδων συναρτήσεων.

```
void p1 ()
void p2 (int n)
void p3 (int a, int b, byref bool b)
double f1 (double x)
int  f2 (byref char * s)
double * f3 (double x)
```

Το σώμα μιας συνάρτησης περικλείεται μέσα σε άγκιστρα { και }. Μπορεί να περιέχει δηλώσεις, ακολουθούμενες από εντολές. Η Edsger ακολουθεί τους κανόνες εμβέλειας της Pascal, όσον αφορά στην ορατότητα των ονομάτων μεταβλητών, υποπρογραμμάτων και παραμέτρων.

1.4 Εκφράσεις

Κάθε έκφραση της Edsger διαθέτει ένα μοναδικό τύπο[1] και μπορεί να αποτιμηθεί δίνοντας ως αποτέλεσμα μια τιμή αυτού του τύπου. Οι εκφράσεις διακρίνονται σε δύο κατηγορίες: αυτές που προκύπτουν από l-values, οι οποίες περιγράφονται στην ενότητα 1.4.1, και αυτές που προκύπτουν από r-values, που περιγράφονται στις ενότητες 1.4.2 ως 1.4.4. Τα δυο αυτά είδη τιμών έχουν πάρει

[1] Εξαίρεση αποτελεί η σταθερά μηδενικού δείκτη `NULL`, η οποία δεν έχει μοναδικό τύπο. Περιγράφεται στην ενότητα 1.4.2.

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.

Τόσο οι l-values όσο και οι r-values 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 συναρτήσεων και οι μεταβλητές που κατασκευάζονται με δυναμική παραχώρηση μνήμης. Συγκεκριμένα:

- Το όνομα μιας μη μόνιμης 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 p είναι μια έκφραση τύπου t `*`, τότε `*`e είναι μια l-value με τύπο t, που αντιστοιχεί στο αντικείμενο όπου δείχνει ο δείκτης που παριστάνει η τιμή της p.

- Αν p είναι μια έκφραση τύπου t`*` και e είναι μια έκφραση τύπου `int`, τότε p`[`e`]` είναι μια l-value με τύπο t. Αυτή η l-value είναι ισοδύναμη με την l-value `*(`p`+`e`)`, όπως αυτή περιγράφεται στην ενότητα 1.4.3.

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 Edsger 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 λέξεις κλειδιά `true` και `false`, με τύπο `bool` και προφανείς τιμές.

- Οι πραγματικές σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `double` και η τιμή τους είναι ίση με τον μη αρνητικό πραγματικό 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.

- Η λέξη κλειδί `NULL` που έχει τύπο t`*`, για κάθε έγκυρο τύπο t, και η τιμή της οποίας είναι ο μηδενικός δείκτης. Είναι η μόνη έκφραση της Edsger που δεν έχει μοναδικό τύπο. Ο μηδενικός δείκτης απαγορεύεται να αποδεικτοδοτηθεί με χρήση του τελεστή `*`.

- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `char *`. Κάθε τέτοια r-value είναι ένας δείκτης σε έναν πίνακα αντικείμενων τύπου `char`, όπου 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.

1.4.3 Τελεστές

Οι τελεστές της Edsger διακρίνονται σε τελεστές με ένα, δύο και τρία τελούμενα. Από τους τελεστές με ένα τελούμενο, ορισμένοι γράφονται πρίν το τελούμενο (prefix) και ορισμένοι μετά (postfix). Οι τελεστές με δύο τελούμενα γράφονται πάντα μεταξύ των τελουμένων (infix) και η η αποτίμηση των τελουμένων γίνεται από αριστερά προς τα δεξιά. Ο μοναδικός τελεστής με τρία τελούμενα είναι ο `?:` και έχει ειδική σύνταξη και σημασιολογία, όπως θα εξηγηθεί παρακάτω.

Ήδη στην ενότητα 1.4.1 περιγράφηκε ο τελεστής αναφοράς σε στοιχείο πίνακα, η σύνταξη του οποίου αποτελεί εξαίρεση στα παραπάνω, και ο τελεστής αποδεικτοδότησης `*`. Οι δύο αυτοί τελεστές είναι οι μοναδικοί που έχουν ως αποτέλεσμα l-value. Στη συνέχεια περιγράφονται οι υπόλοιποι τελεστές της Edsger, που έχουν ως αποτέλεσμα r-value. Οι τελεστές αυτοί χωρίζονται σε δύο κατηγορίες: τους *τελεστές υπολογισμού* και τους *τελεστές ανάθεσης*. Οι πρώτοι χρησιμοποιούνται για τον υπολογισμό τιμών, ενώ οι δεύτεροι για την ανάθεση τιμών σε μεταβλητές. Η αποτίμηση μιας έκφραση που περιέχει έναν ή περισσότερους τελεστές ανάθεσης, όπως στη C, εκτός από το να υπολογίζει μια τιμή αποτελέσματος έχει ως *παρενέργεια* τη μεταβολή των τιμών κάποιων μεταβλητών.

Οι τελεστές υπολογισμού είναι οι ακόλουθοι:

- Ο τελεστής `&` επιστρέφει τη διεύθυνση ενός αντικειμένου. Αν l είναι μια l-value τύπου t, τότε `&`l είναι μια r-value τύπου t`*`. Η τιμή της είναι η διεύθυνση του αντικειμένου που αντιστοιχεί στην l και είναι πάντα διαφορετική του μηδενικού δείκτη.

- Οι τελεστές με ένα τελούμενο `+` και `-` υλοποιούν τους τελεστές προσήμου. Το τελούμενο πρέπει να είναι αριθμητικού τύπου και το αποτέλεσμα είναι του ίδιου τύπου με το τελούμενο.

- Ο τελεστής `!` υλοποιεί τη λογική άρνηση. Το τελούμενό του πρέπει να είναι τύπου `bool`, και τον ίδιο τύπο έχει και το αποτέλεσμα.

- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `/` και `%` υλοποιούν τις αριθμητικές πράξεις. Τα επιτρεπτά τελούμενα αυτών των τελεστών περιγράφονται παρακάτω:

  - Αν και τα δύο τελούμενα των `+`, `-`, `*`, `/` και `%` είναι τύπου `int` τότε και το αποτέλεσμα είναι τύπου `int`. Στην περίπτωση αυτή, οι τελεστές `/` και `%` υλοποιούν την ακέραια διαίρεση.
  - Αν τα τελούμενα των `+`, `-`, `*` και `/` είναι τύπου `double` τότε και το αποτέλεσμα είναι τύπου `double`. Στην περίπτωση αυτή, ο τελεστής `/` υλοποιεί την πραγματική διαίρεση.
  - Αν το πρώτο τελούμενο των `+` και `-` είναι τύπου t`*` και το δεύτερο τελούμενο είναι τύπου `int`, τότε το αποτέλεσμα είναι τύπου t`*`. Η τιμή του αποτελέσματος είναι ένας δείκτης μετατοπισμένος ως προς την τιμή του πρώτου τελούμενου κατά τόσες θέσεις αντικειμένων τύπου t όσες ορίζει η τιμή του δεύτερου τελουμένου (pointer arithmetic).

- Οι τελεστές `==` και `!=` υλοποιούν αντίστοιχα την ισότητα και την ανισότητα. Το αποτέλεσμα είναι τύπου `bool`. Τα τελούμενα πρέπει να είναι του ίδιου τύπου.

- Οι τελεστές `<`, `>`, `<=` και `>=` υλοποιούν τις σχέσεις ανισότητας μεταξύ αριθμών ή δεικτών. Το αποτέλεσμα είναι τύπου `bool`. Τα τελούμενα πρέπει να είναι του ίδιου τύπου. Αν είναι αριθμητικά, η σύγκριση γίνεται βάσει των αριθμητικών τους τιμών. Αν είναι τύπου `bool`, η σύγκριση γίνεται θεωρώντας ότι `false`$<$`true`. Τέλος, αν είναι δείκτες, η σύγκριση έχει νόημα μόνο όταν και οι δύο δείχνουν σε αντικείμενα του ίδιου μονοδιάστατου πίνακα και γίνεται βάσει της σχετικής θέσης αυτών των αντικειμένων μέσα στον πίνακα.

- Οι τελεστές `&&` και `||` υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης. Τα τελούμενα πρέπει να είναι τύπου `bool` και τον ίδιο τύπο έχει και το αποτέλεσμα. Η αποτίμηση εκφράσεων που χρησιμοποιούν αυτούς τους τελεστές γίνεται με *βραχυκύκλωση*

(short-circuit). Δηλαδή, αν το αποτέλεσμα της έκφρασης είναι γνωστό από την αποτίμηση και μόνο του πρώτου τελούμενου, το δεύτερο τελούμενο δεν αποτιμάται καθόλου.

- Ο τελεστής `,` (κόμμα) υπολογίζει τις τιμές των δύο τελουμένων και επιστρέφει την τιμή του δεύτερου, αγνοώντας αυτήν του πρώτου. Τα δύο τελούμενα μπορούν να είναι οποιουδήποτε έγκυρου τύπου. Το αποτέλεσμα είναι του ίδιου τύπου με αυτόν του δεύτερου τελούμενου.

- Ο τελεστής με τρία τελούμενα `?:` έχει την ειδική σύνταξη e `?` e_1 `:` e_2. Το πρώτο τελούμενο πρέπει να είναι τύπου `bool`, ενώ τα άλλα δύο τελούμενα πρέπει να είναι του ίδιου τύπου. Το αποτέλεσμα έχει τον ίδιο τύπο με τα δύο τελευταία τελούμενα. Η αποτίμηση της έκφρασης e `?` e_1 `:` e_2 γίνεται ξεκινώντας από την αποτίμηση της e. Αν η τιμή αυτής είναι `true`, τότε αποτιμάται η e_1 και η τιμή αυτής είναι το αποτέλεσμα. Διαφορετικά, αποτιμάται η e_2 και η τιμή αυτής είναι το αποτέλεσμα. Σε κάθε περίπτωση, αποτιμάται μόνο ένα από τα τελούμενα e_1 και e_2.

Οι τελεστές ανάθεσης είναι οι ακόλουθοι:

- Ο τελεστής `=` αναθέτει την τιμή του δεύτερου τελουμένου στη μεταβλητή που αντιστοιχεί στο πρώτο τελούμενο. Το πρώτο τελούμενο πρέπει να είναι μια l-value οποιουδήποτε έγκυρου τύπου t, ενώ το δεύτερο τελούμενο πρέπει να είναι του ίδιου τύπου t. Το αποτέλεσμα είναι και αυτό τύπου t και η τιμή του είναι η τιμή που ανατίθεται στη μεταβλητή.

- Ο τελεστής op`=`, όπου $op \in \{$`+`, `-`, `*`, `/`, `%`$\}$, συνδυάζει υπολογισμό και ανάθεση. Η έκφραση l op`=` e είναι σημασιολογικά ισοδύναμη με l `=` l op e, με τη διαφορά ότι το τελούμενο l θα υπολογιστεί μόνο μια φορά.

- Οι τελεστές `++` και `--` δέχονται ένα τελούμενο και υπάρχουν σε δύο μορφές, μια prefix και μια postfix. Το τελούμενό τους πρέπει να είναι μια l-value αριθμητικού τύπου ή τύπου δείκτη. Το αποτέλεσμα είναι του ίδιου τύπου με το τελούμενο. Οι τελεστές προκαλούν την αύξηση ή τη μείωση της τιμής της l-value. Στην περίπτωση της μορφής prefix το αποτέλεσμα που υπολογίζεται είναι η τιμή της l-value μετά την αύξηση ή τη μείωση, ενώ στην περίπτωση της μορφής postfix το αποτέλεσμα είναι η τιμή της l-value πριν την αύξηση ή τη μείωση.

Στον πίνακα 2 ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της Edsger. Οι γραμμές που βρίσκονται υψηλότερα στον πίνακα περιέχουν τελεστές μεγαλύτερης προτεραιότητας. Τελεστές που βρίσκονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα. Στον πίνακα συμπεριλαμβάνονται η κλήση συναρτήσεων, η μετατροπή τύπου και οι τελεστές δυναμικής παραχώρησης μνήμης, που περιγράφονται σε επόμενες ενότητες.

1.4.4 Κλήση συναρτήσεων

Αν f είναι το όνομα μιας συνάρτησης με αποτέλεσμα τύπου 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 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 συνάρτησης, οι πραγματικές παράμετροι αποτιμώνται από αριστερά προς τα δεξιά.

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της Edsger.

Τελεστές	Περιγραφή	Αριθμός τελουμένων	Θέση και προσεταιριστικότητα
`[ ]  ( )`	Στοιχείο πίνακα, κλήση συνάρτησης	2 ή >	ειδική
`++  --`	Αύξηση και μείωση	1	postfix
`&  *  +  -  !`	Διεύθυνση αντικειμένου, αποδεικτοδότηση, πρόσημα, λογική άρνηση	1	prefix
`new  delete`	Δυναμική παραχώρηση μνήμης	1	prefix
`++  --`	Αύξηση και μείωση	1	prefix
`( )`	Μετατροπή τύπου	2	ειδική
`*  /  %`	Πολλαπλασιαστικοί τελεστές	2	infix, αριστερή
`+  -`	Προσθετικοί τελεστές	2	infix, αριστερή
`==  !=  >  <  <=  >=`	Σχεσιακοί τελεστές	2	infix, καμία
`&&`	Λογική σύζευξη	2	infix, αριστερή
`||`	Λογική διάζευξη	2	infix, αριστερή
`?:`	Τελεστής συνθήκης	3	ειδική
`=  +=  -=  *=  /=  %=`	Τελεστές ανάθεσης	2	infix, δεξιά
`,`	Τελεστής παράθεσης	2	infix, αριστερή

1.4.5 Δυναμική διαχείριση μνήμης

Οι τελεστές `new` και `delete` χρησιμοποιούνται για τη δυναμική διαχείριση μνήμης.

- Η έκφραση `new` t `[`e`]`, όπου t οποιοσδήποτε έγκυρος τύπος και e έκφραση τύπου `int`, προκαλεί τη δυναμική παραχώρηση μνήμης. Η έκφραση e καθώς και οι αγκύλες είναι προαιρετικές. Αν δίνονται, τότε η τιμή της e πρέπει να είναι ένας θετικός ακέραιος αριθμός n. Διαφορετικά, θεωρείται ότι $n = 1$. Το αποτέλεσμα είναι τύπου t`*`. Ο δείκτης αυτός τοποθετείται να δείχνει προς το πρώτο από n νέα δυναμικά αντικείμενα τύπου t.

- Η έκφραση `delete` e, όπου e έκφραση τύπου t`*`, προκαλεί την αποδέσμευση της μνήμης στην οποία δείχνει η τιμή της e. Η μνήμη αυτή πρέπει να έχει προηγουμένως παραχωρηθεί δυναμικά με χρήση του τελεστή `new`. Το αποτέλεσμα είναι τύπου t`*` και η τιμή του είναι πάντοτε `NULL`.

1.5 Εντολές

Οι εντολές που υποστηρίζει η γλώσσα Edsger είναι οι ακόλουθες:

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

- Η εντολή έκφρασης e `;` που αποτιμά την έκφραση e και αγνοεί την τιμή του αποτελέσματος.

- Η σύνθετη εντολή, που αποτελείται από μια σειρά έγκυρων εντολών ανάμεσα σε άγκιστρα { και }. Οι εντολές αυτές εκτελούνται διαδοχικά, εκτός αν κάποια είναι εντολή άλματος.

- Η εντολή ελέγχου `if (`e`)` s_1 `else` s_2. Η έκφραση e πρέπει να έχει τύπο `bool` και τα s_1, s_2 να είναι έγκυρες εντολές. Το τμήμα `else` είναι προαιρετικό.

- Η εντολή βρόχου `for (`e_1 `;` e_2 `;` e_3`)` s. Οι εκφράσεις e_1, e_2 και e_3 είναι προαιρετικές. Αν δίνεται, η έκφραση e_2 πρέπει να έχει τύπο `bool`, διαφορετικά θεωρείται ίση με `true`. Το s πρέπει να είναι έγκυρη εντολή. Η σημασιολογία της εντολής βρόχου είναι η ίδια με αυτή

της C. Μπροστά από την εντολή βρόχου μπορεί να υπάρχει μια ετικέτα της μορφής I: το όνομα της οποίας πρέπει να είναι μοναδικό μέσα στην τρέχουσα συνάρτηση.

- Οι εντολές άλματος **break** I**;** και **continue** I**;**, όπου το I είναι προαιρετικό. Αν δίνεται, το I πρέπει να είναι το όνομα μιας ετικέτας εντολής βρόχου μέσα στο σώμα της οποίας βρίσκεται η εντολή άλματος. Αν το I παραλείπεται, οι εντολές αυτές έχουν την ίδια σημασιολογία με τη γλώσσα C, δηλαδή προκαλούν τον τερματισμό ή τη συνέχιση του βρόχου μέσα στον οποίο βρίσκονται. Αν το I δίνεται, προκαλείται ο τερματισμός ή η συνέχιση του βρόχου που φέρει την ετικέτα I.

- Η εντολή άλματος **return** e **;** που τερματίζει την εκτέλεση της τρέχουσας συνάρτησης και επιστρέφει την τιμή της e ως αποτέλεσμα της συνάρτησης. Αν η τρέχουσα συνάρτηση έχει τύπο αποτελέσματος **void** τότε η έκφραση e πρέπει να παραλείπεται. Διαφορετικά, η έκφραση e πρέπει να έχει τύπο ίδιο με τον τύπο αποτελέσματος της τρέχουσας συνάρτησης.

1.6 Οδηγίες προς το μεταγλωττιστή

Η μοναδική οδηγία, που επιτρέπει ο μεταγλωττιστής της γλώσσας Edsger είναι η οδηγία **#include**. Η οδηγία αυτή έχει την ίδια σημασιολογία όπως και στη γλώσσα C, δηλαδή επιτρέπει την ανάγνωση ενός εξωτερικού αρχείου σαν αυτό να ήταν τμήμα του προγράμματος. Πρέπει να βρίσκεται υποχρεωτικά στην αρχή της γραμμής (να μην προηγούνται κενά διαστήματα). Μετά την οδηγία **#include** ακολουθεί μια σταθερή συμβολοσειρά που περιέχει το όνομα του αρχείου το οποίο θα συμπεριληφθεί σε αυτό το σημείο.

Η οδηγία αυτή απευθύνεται ουσιαστικά στο λεκτικό αναλυτή. Σε περίπτωση που αυτός συναντήσει αυτή την οδηγία, θα πρέπει να σταματήσει την ανάγνωση του αρχείου προγράμματος και να συνεχίσει με την επεξεργασία του αρχείου που ζητείται να συμπεριληφθεί. Μετά το τέλος αυτού του αρχείου, ο λεκτικός αναλυτής πρέπει να συνεχίσει από το σημείο του αρχείου προγράμματος, στο οποίο είχε σταματήσει. Τα αρχεία που διαβάζονται με την οδηγία **#include** μπορούν να περιέχουν και αυτά τέτοιες οδηγίες. Φυσικά, μεμονωμένες λεκτικές μονάδες καθώς και σχόλια πρέπει να περιέχονται πλήρως σε ένα αρχείο προγράμματος (δεν επιτρέπεται να αρχίζουν σε ένα αρχείο προγράμματος και να τελειώνουν σε κάποιο άλλο).

1.7 Βιβλιοθήκη έτοιμων συναρτήσεων

Η Edsger υποστηρίζει ένα σύνολο προκαθορισμένων συναρτήσεων, οι οποίες έχουν υλοποιηθεί σε assembly του 80x86 ως μια *βιβλιοθήκη έτοιμων συναρτήσεων* (run-time library). Οι επικεφαλίδες αυτών των συναρτήσεων βρίσκονται σε κατάλληλα *αρχεία επικεφαλίδων* (header files), τα οποία διαβάζονται με χρήση της οδηγίας **#include**. Εφόσον το αντίστοιχο αρχείο επικεφαλίδας έχει συμπεριληφθεί σε ένα πρόγραμμα και η βιβλιοθήκη έτοιμων συναρτήσεων χρησιμοποιηθεί κατά τη συνένωση (linking) του τελικού κώδικα, οι συναρτήσεις που περιέχει βρίσκονται στη διάθεση του προγραμματιστή. Είναι ορατές σε κάθε δομική μονάδα, εκτός αν επισκιάζονται από μεταβλητές, παραμέτρους ή άλλες συναρτήσεις με το ίδιο όνομα.

Παρακάτω δίνονται οι δηλώσεις τους, όπως γράφονται στα αρχεία επικεφαλίδας της Edsger, και εξηγείται η λειτουργία τους.

1.7.1 Είσοδος και έξοδος — #include "stdio.h"

```
void writeInteger (int n);
void writeBoolean (bool b);
void writeChar    (char c);
void writeReal    (double d);
void writeString  (char * s);
```

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

```
int    readInteger ();
bool   readBoolean ();
char   readChar    ();
double readReal    ();
void   readString  (int size, char * s);
```

Αντίστοιχα, τα παραπάνω υποπρογράμματα χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της Edsger 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 Μαθηματικές συναρτήσεις — #include "math.h"

```
int    abs  (int n);
double fabs (double d);
```

Η απόλυτη τιμή ενός ακέραιου ή πραγματικού αριθμού.

```
double sqrt (double d);
double sin  (double d);
double cos  (double d);
double tan  (double d);
double atan (double d);
double exp  (double d);
double ln   (double d);
double pi   ();
```

Βασικές μαθηματικές συναρτήσεις: τετραγωνική ρίζα, τριγωνομετρικές συναρτήσεις, εκθετική συνάρτηση, φυσικός λογάριθμος, ο αριθμός π.

1.7.3 Συναρτήσεις μετατροπής — #include "stdlib.h"

```
int trunc (double d);
int round (double d);
```

Η **trunc** επιστρέφει τον πλησιέστερο ακέραιο αριθμό, η απόλυτη τιμή του οποίου είναι μικρότερη από την απόλυτη τιμή του **d**. Η **round** επιστρέφει τον πλησιέστερο ακέραιο αριθμό. Σε περίπτωση αμφιβολίας, προτιμάται ο αριθμός με τη μεγαλύτερη απόλυτη τιμή.

```
int  ord (char c);
char chr (int n);
```

Μετατρέπουν από ένα χαρακτήρα στον αντίστοιχο κωδικό ASCII και αντίστροφα.

1.7.4 Συναρτήσεις διαχείρισης συμβολοσειρών — #include "string.h"

```
int  strlen (char * s);
int  strcmp (char * s1, char * s2);
void strcpy (char * trg, char * src);
void strcat (char * trg, char * src);
```

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

2 Πλήρης γραμματική της Edsger

Η σύνταξη της γλώσσας Edsger δίνεται παρακάτω σε μορφή EBNF. Η γραμματική που ακολουθεί είναι *διφορούμενη*, οι αμφισημίες όμως μπορούν να ξεπερασθούν αν λάβει κανείς υπόψη τους κανόνες προτεραιότητας και προσεταιριστικότητας των τελεστών, όπως περιγράφονται στην ενότητα 1.4.3. Τα σύμβολα ⟨I⟩, ⟨int-const⟩, ⟨double-const⟩, ⟨char-const⟩ και ⟨string-literal⟩ είναι τερματικά σύμβολα της γραμματικής.

⟨program⟩ ::= (⟨declaration⟩)$^+$
⟨declaration⟩ ::= ⟨variable-declaration⟩ | ⟨function-declaration⟩ | ⟨function-definition⟩
⟨variable-declaration⟩ ::= ⟨type⟩ ⟨declarator⟩ (“`,`” ⟨declarator⟩)* “`;`”
⟨type⟩ ::= ⟨basic-type⟩ (“`*`”)*
⟨basic-type⟩ ::= “`int`” | “`char`” | “`bool`” | “`double`”
⟨declarator⟩ ::= ⟨I⟩ [“`[`” ⟨constant-expression⟩ “`]`”]
⟨function-declaration⟩ ::= ⟨result-type⟩ ⟨I⟩ “`(`” [⟨parameter-list⟩] “`)`” “`;`”
⟨result-type⟩ ::= ⟨type⟩ | “`void`”
⟨parameter-list⟩ ::= ⟨parameter⟩ (“`,`” ⟨parameter⟩)*
⟨parameter⟩ ::= [“`byref`”] ⟨type⟩ ⟨I⟩
⟨function-definition⟩ ::= ⟨result-type⟩ ⟨I⟩ “`(`” [⟨parameter-list⟩] “`)`”
“`{`” (⟨declaration⟩)* (⟨statement⟩)* “`}`”
⟨statement⟩ ::= “`;`” | ⟨expression⟩ “`;`” | “`{`” (⟨statement⟩)* “`}`”
| “`if`” “`(`” ⟨expression⟩ “`)`” ⟨statement⟩ [“`else`” ⟨statement⟩]
| [⟨I⟩ “`:`”] “`for`” “`(`” [⟨expression⟩] “`;`” [⟨expression⟩] “`;`”
[⟨expression⟩] “`)`” ⟨statement⟩
| “`continue`” [⟨I⟩] “`;`” | “`break`” [⟨I⟩] “`;`” | “`return`” [⟨expr⟩] “`;`”
⟨expression⟩ ::= ⟨I⟩ | “`(`” ⟨expression⟩ “`)`” | “`true`” | “`false`” | “`NULL`”
| ⟨int-const⟩ | ⟨char-const⟩ | ⟨double-const⟩ | ⟨string-literal⟩
| ⟨I⟩ “`(`” [⟨expression-list⟩] “`)`” | ⟨expression⟩ “`[`” ⟨expression⟩ “`]`”
| ⟨unary-operator⟩ ⟨expression⟩ | ⟨expression⟩ ⟨binary-operator⟩ ⟨expression⟩
| ⟨unary-assignment⟩ ⟨expression⟩ | ⟨expression⟩ ⟨unary-assignment⟩
| ⟨expression⟩ ⟨binary-assignment⟩ ⟨expression⟩ | “`(`” ⟨type⟩ “`)`” ⟨expression⟩
| ⟨expression⟩ “`?`” ⟨expression⟩ “`:`” ⟨expression⟩
| “`new`” ⟨type⟩ [“`[`” ⟨expression⟩ “`]`”] | “`delete`” ⟨expression⟩
⟨expression-list⟩ ::= ⟨expression⟩ (“`,`” ⟨expression⟩)*
⟨constant-expression⟩ ::= ⟨expression⟩
⟨unary-operator⟩ ::= “`&`” | “`*`” | “`+`” | “`-`” | “`!`”
⟨binary-operator⟩ ::= “`*`” | “`/`” | “`%`” | “`+`” | “`-`” | “`<`” | “`>`” | “`<=`” | “`>=`” | “`==`” | “`!=`”
| “`&&`” | “`||`” | “`,`”
⟨unary-assignment⟩ ::= “`++`” | “`--`”
⟨binary-assignment⟩ ::= “`=`” | “`*=`” | “`/=`” | “`%=`” | “`+=`” | “`-=`”

3 Παραδείγματα

Στην παράγραφο αυτή δίνονται έξι παραδείγματα προγραμμάτων στη γλώσσα Edsger. Για κάθε ένα δίνεται ο αρχικός κώδικας, ο ενδιάμεσος κώδικας και η μορφή των εγγραφημάτων δραστηριοποίησης των δομικών μονάδων, καθώς και ο τελικός κώδικας, αν το μέγεθος του παραδείγματος το επιτρέπει. Σε καμιά φάση της μεταγλώττισης των παραδειγμάτων δεν έχουν γίνει βελτιστοποιήσεις. Έχει γίνει η παραδοχή ότι οι μετατροπές τύπων γίνονται αυτόματα στη φάση της παραγωγής τελικού κώδικα. Στην περίπτωση των τελεστών αύξησης και μείωσης, των σύνθετων τελεστών ανάθεσης και της εντολής `for`, ο ενδιάμεσος κώδικας που έχει κατασκευαστεί μπορεί εύκολα

να βελτιωθεί σημαντικά. Συνεπώς, μη θεωρήσετε ότι ο μεταγλωττιστής σας πρέπει να παράγει ακριβώς τον ενδιάμεσο και τον τελικό κώδικα αυτής της ενότητας για τα παραδείγματα.

Επισημαίνεται ότι, λόγω έλλειψης διαθέσιμου μεταγλωττιστή της γλώσσας Edsger, τα παραδείγματα που ακολουθούν έχουν μεταγλωττισθεί χειρωνακτικά. Το γεγονός αυτό καθιστά εξαιρετικά πιθανή, αν όχι βέβαιη, την ύπαρξη ασυνεπειών ανάμεσα στον ενδιάμεσο ή τον τελικό κώδικα που προτείνεται και στον αντίστοιχο κώδικα, που θα έπρεπε να παράγεται με βάση το βιβλίο. Πάντως, ο τελικός κώδικας κάθε παραδείγματος έχει ελεγχθεί ως προς την ορθότητά του και έχει διαπιστωθεί ότι εκτελείται χωρίς λάθη.

3.1 Πες γεια!

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

```
#include "stdio.h"

void main ()
{
   writeString("Hello world!\n");
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:

```
1: unit, main, -, -
2: par, "Hello world!\n", V, -
3: call, -, -, writeString
4: endu, main, -, -
```

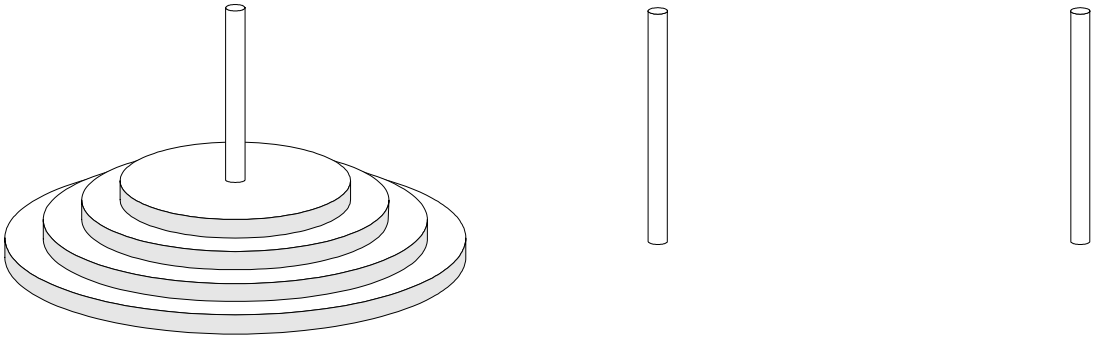
Το εγγράφημα δραστηριοποίησης του προγράμματος είναι εξαιρετικά απλό και δε θα δοθεί. Ο τελικός κώδικας είναι ο εξής:

```
xseg        segment public 'code'
            assume  cs:xseg, ds:xseg, ss:xseg
            org     100h

main        proc    near
            call    near ptr _main_1
            mov     ax, 4C00h
            int     21h
main        endp

@1:
_main_1     proc    near
            push    bp
            mov     bp, sp
            sub     sp, 0
@2:
            lea     ax, byte ptr @str_1
            push    ax
@3:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@4:
@main_1:    mov     sp, bp
            pop     bp
            ret
_main_1     endp

            extrn   _writeString : proc

@str_1      db      'Hello world!'
            db      10, 0

xseg        ends
            end     main
```

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:



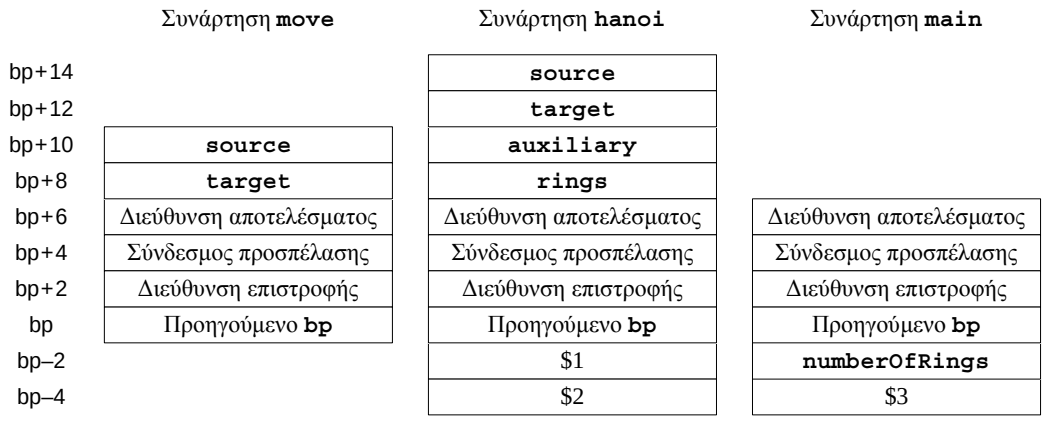
Σχήμα 1: Οι πύργοι του Hanoi.

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

```
#include "stdio.h"

void main ()
{
   int numberOfRings;

   void hanoi (char * source, char * target,
               char * auxiliary, int rings)
   {
      void move (char * source, char * target)
      {
         writeString("Move from ");
         writeString(source);
         writeString(" to ");
         writeString(target);
         writeString(".\n");
      }

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

   writeString("Please, give the number of rings: ");
   numberOfRings = readInteger();
   writeString("\nHere is the solution:\n\n");
   hanoi("left", "right", "middle", numberOfRings);
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:



Σχήμα 2: Εγγραφήματα δραστηριοποίησης για τους Πύργους του Hanoi.

```
 1: unit, move, -, -
 2: par, "Move from ", V, -
 3: call, -, -, writeString
 4: par, source, V, -
 5: call, -, -, writeString
 6: par, " to ", V, -
 7: call, -, -, writeString
 8: par, target, V, -
 9: call, -, -, writeString
10: par, ".\n", V, -
11: call, -, -, writeString
12: endu, move, -, -

13: unit, hanoi, -, -
14: >=, rings, 1, 16
15: jump, -, -, 31
16: par, source, V, -
17: par, auxiliary, V, -
18: par, target, V, -
19: -, rings, 1, $1
20: par, $1, V, -
21: call, -, -, hanoi
22: par, source, V, -
23: par, target, V, -
24: call, -, -, move
25: par, auxiliary, V, -
26: par, target, V, -
27: par, source, V, -
28: -, rings, 1, $2
29: par, $2, V, -
30: call, -, -, hanoi
31: endu, hanoi, -, -

32: unit, main, -, -
33: par, "Please, give the number of
      rings : ", V, -
34: call, -, -, writeString
35: par, $3, RET, -
36: call, -, -, readInteger
37: :=, $3, -, numberOfRings
38: par, "\nHere is the solution :\n\n",
      V, -
39: call, -, -, writeString
40: par, "left", V, -
41: par, "right", V, -
42: par, "middle", V, -
43: par, numberOfRings, V, -
44: call, -, -, hanoi
45: endu, main, -, -
```

Τα εγγραφήματα δραστηριοποίησης για το παραπάνω πρόγραμμα φαίνονται στο σχήμα 2. Ο τελικός κώδικας δίνεται παρακάτω.

```
xseg        segment public 'code'
            assume  cs:xseg, ds:xseg, ss:xseg
            org     100h

main        proc    near
            call    near ptr _main_1
            mov     ax, 4C00h
            int     21h
main        endp

@1:
_move_3     proc    near
            push    bp
            mov     bp, sp
            sub     sp, 0
@2:
            lea     ax, byte ptr @str_1
            push    ax
@3:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@4:
            mov     ax, word ptr [bp+10]
            push    ax
```

```
@5:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@6:
            lea     ax, byte ptr @str_2
            push    ax
@7:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@8:
            mov     ax, word ptr [bp+8]
            push    ax
@9:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@10:
            lea     ax, byte ptr @str_3
            push    ax
@11:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@12:
@move_3:    mov     sp, bp
            pop     bp
            ret
_move_3     endp

@13:
_hanoi_2    proc    near
            push    bp
            mov     bp, sp
            sub     sp, 4
@14:
            mov     ax, word ptr [bp+8]
            mov     dx, 1
            cmp     ax, dx
            jge     @16
@15:
            jmp     @31
@16:
            mov     ax, word ptr [bp+14]
            push    ax
@17:
            mov     ax, word ptr [bp+10]
            push    ax
@18:
            mov     ax, word ptr [bp+12]
            push    ax
@19:
            mov     ax, word ptr [bp+8]
            mov     dx, 1
            sub     ax, dx
            mov     word ptr [bp-2], ax
@20:
            mov     ax, word ptr [bp-2]
            push    ax
@21:
            sub     sp, 2
            push    word ptr [bp+4]
            call    near ptr _hanoi_2
            add     sp, 12
@22:
            mov     ax, word ptr [bp+14]
            push    ax
@23:
            mov     ax, word ptr [bp+12]
            push    ax
@24:
            sub     sp, 2
            push    bp
            call    near ptr _move_3
            add     sp, 8
@25:
            mov     ax, word ptr [bp+10]
            push    ax
@26:
            mov     ax, word ptr [bp+12]
            push    ax
@27:
            mov     ax, word ptr [bp+14]
            push    ax
@28:
            mov     ax, word ptr [bp+8]
            mov     dx, 1
            sub     ax, dx
            mov     word ptr [bp-4], ax
@29:
            mov     ax, word ptr [bp-4]
            push    ax
@30:
            sub     sp, 2
            push    word ptr [bp+4]
            call    near ptr _hanoi_2
            add     sp, 12
@31:
@hanoi_2:   mov     sp, bp
            pop     bp
            ret
_hanoi_2    endp

@32:
_main_1     proc    near
            push    bp
            mov     bp, sp
            sub     sp, 4
@33:
            lea     ax, byte ptr @str_4
            push    ax
@34:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@35:
            lea     si, word ptr [bp-4]
            push    si
@36:
            push    bp
```

```
            call    near ptr _readInteger
            add     sp, 4
@37:
            mov     ax, word ptr [bp-4]
            mov     word ptr [bp-2], ax
@38:
            lea     ax, byte ptr @str_5
            push    ax
@39:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6
@40:
            lea     ax, byte ptr @str_6
            push    ax
@41:
            lea     ax, byte ptr @str_7
            push    ax
@42:
            lea     ax, byte ptr @str_8
            push    ax
@43:
            mov     ax, word ptr [bp-2]
            push    ax
@44:
            sub     sp, 2
            push    bp
            call    near ptr _hanoi_2
            add     sp, 12
@45:
@main_1:    mov     sp, bp
            pop     bp
            ret
_main_1     endp

            extrn   _writeString : proc
            extrn   _readInteger : proc

@str_1      db      'Moving from '
            db      0

@str_2      db      ' to '
            db      0

@str_3      db      '.'
            db      10, 0

@str_4      db      'Please, give the '
            db      'number of rings : '
            db      0

@str_5      db      10
            db      'Here is the solution :'
            db      10, 10, 0

@str_6      db      'left'
            db      0

@str_7      db      'right'
            db      0

@str_8      db      'middle'
            db      0

xseg        ends
            end     main
```

3.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα Edsger 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

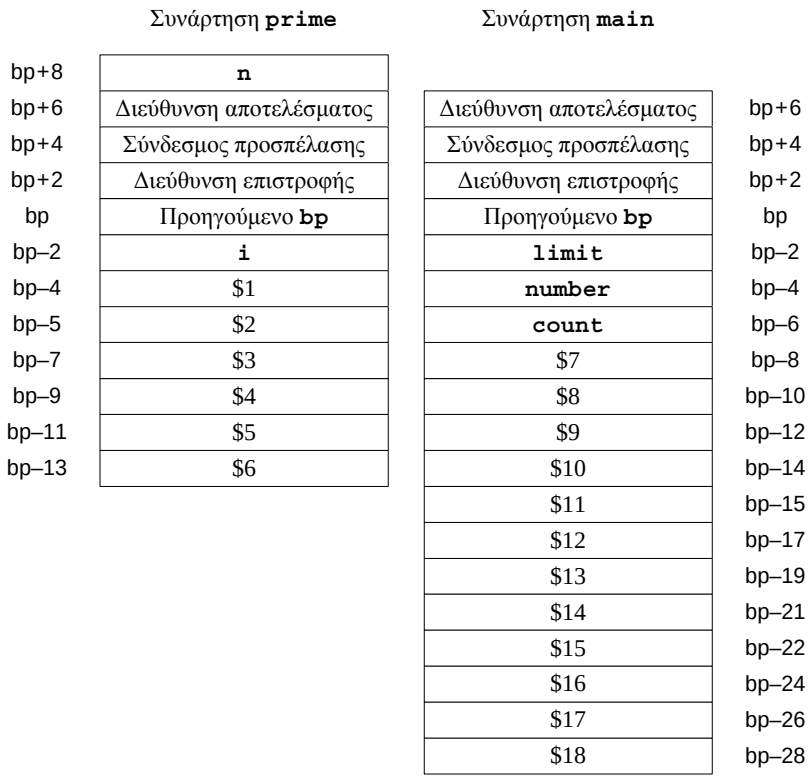
Το αντίστοιχο πρόγραμμα στη γλώσσα Edsger είναι το εξής:

```
#include "stdio.h"

void main ()
{
   int limit, number, counter;

   bool prime (int n)
   {
      int i;

      if (n < 0)
         return prime(-n);
      if (n < 2)
         return false;
      if (n == 2)
         return true;
      if (n % 2 == 0)
         return false;
      for (i = 3; i <= n / 2; i += 2)
         if (n % i == 0)
            return false;
      return true;
   }

   writeString("Please, give the upper limit: ");
   limit = readInteger();
   writeString("Prime numbers between 0 and ");
   writeInteger(limit);
   writeString("\n\n");
   counter = 0;
   if (limit >= 2) {
      counter++;
      writeString("2\n");
   }
   if (limit >= 3) {
      counter++;
      writeString("3\n");
   }
   for (number = 6; number <= limit; number += 6) {
      if (prime(number - 1)) {
         counter++;
         writeInteger(number - 1);
         writeString("\n");
      }
      if (number != limit && prime(number + 1)) {
         counter++;
         writeInteger(number + 1);
         writeString("\n");
      }
   }
   writeString("\n");
   writeInteger(counter);
   writeString(" prime number(s) were found.\n");
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:

```
 1: unit, prime, -, -
 2: <, n, 0, 4
 3: jump, -, -, 10
 4: -, n, -, $1
 5: par, $1, V, -
 6: par, $2, RET, -
 7: call, -, -, prime
 8: :=, $2, -, $$
 9: jump, -, -, 35
10: <, n, 2, 12
11: jump, -, -, 14
12: :=, false, -, $$
13: jump, -, -, 35
14: =, n, 2, 16
15: jump, -, -, 18
16: :=, true, -, $$
17: jump, -, -, 35
18: %, n, 2, $3
19: =, $3, 0, 21
20: jump, -, -, 23
21: :=, false, -, $$
22: jump, -, -, 35
23: :=, 3, -, i
24: /, n, 2, $4
25: <=, i, $4, 27
26: jump, -, -, 35
27: %, n, i, $5
28: =, $5, 0, 30
29: jump, -, -, 32
30: :=, false, -, $$
31: ret, -, -, -
32: +, i, 2, $6
33: :=, $6, -, i
34: jump, -, -, 24
35: :=, true, -, $$
36: endu, prime, -, -

37: unit, main, -, -
38: par, "Please, give the upper
      limit : ", V, -
39: call, -, -, writeString
40: par, $7, RET, -
41: call, -, -, readInteger
42: :=, $7, -, limit
43: par, "Prime numbers between 0 and ",
      V, -
44: call, -, -, writeString
45: par, limit, V, -
46: call, -, -, writeInteger
47: par, "\n\n", V, -
48: call, -, -, writeString
49: :=, 0, -, counter
50: >=, limit, 2, 52
51: jump, -, -, 56
52: +, counter, 1, $8
53: :=, $8, -, counter
54: par, "2\n", V, -
55: call, -, -, writeString
56: >=, limit, 3, 58
57: jump, -, -, 62
58: +, counter, 1, $9
59: :=, $9, -, counter
60: par, "3\n", V, -
61: call, -, -, writeString
62: :=, 6, -, number
63: <=, number, limit, 65
64: jump, -, -, 96
65: -, number, 1, $10
66: par, $10, V, -
67: par, $11, RET, -
68: call, -, -, prime
69: ifb, $11, -, 71
70: jump, -, -, 78
71: +, counter, 1, $12
72: :=, $12, -, counter
73: -, number, 1, $13
74: par, $13, V, -
75: call, -, -, writeInteger
76: par, "\n", V, -
77: call, -, -, writeString
78: <>, number, limit, 80
79: jump, -, -, 93
80: +, number, 1, $14
81: par, $14, V, -
82: par, $15, RET, -
83: call, -, -, prime
84: ifb, $15, -, 86
85: jump, -, -, 93
86: +, counter, 1, $16
87: :=, $16, -, counter
88: +, number, 1, $17
89: par, $17, V, -
90: call, -, -, writeInteger
91: par, "\n", V, -
92: call, -, -, writeString
93: +, number, 6, $18
94: :=, $18, -, number
95: jump, -, -, 63
96: par, "\n", V, -
97: call, -, -, writeString
98: par, counter, V, -
99: call, -, -, writeInteger
100: par, " prime number(s) were
       found.\n", V, -
101: call, -, -, writeString
102: endu, main, -, -
```

Τα εγγραφήματα δραστηριοποίησης για το παραπάνω πρόγραμμα φαίνονται στο σχήμα 3. Λόγω της πολυπλοκότητας του παραδείγματος, δε δίνεται τελικός κώδικας.



Σχήμα 3: Εγγραφήματα δραστηριοποίησης για τους πρώτους αριθμούς.

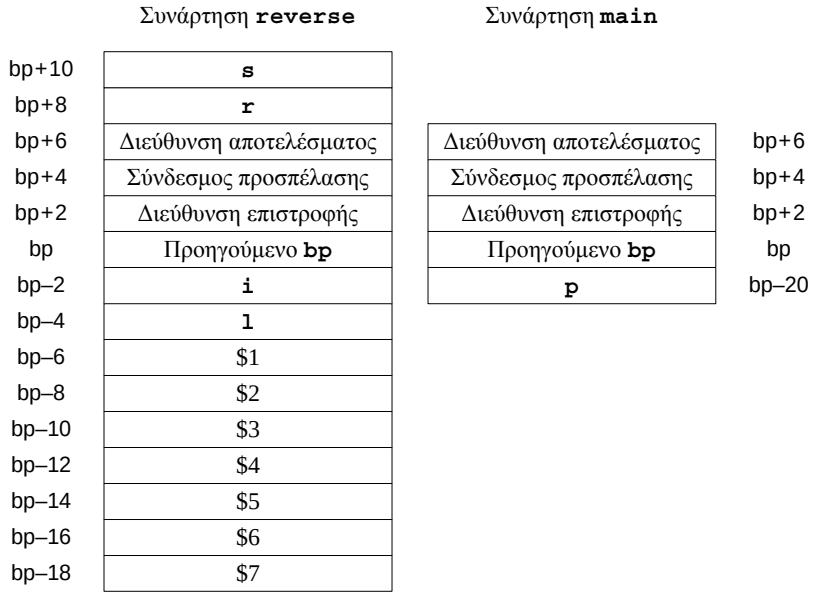
3.4 Αντιστροφή συμβολοσειράς

Το πρόγραμμα που ακολουθεί εκτυπώνει το μήνυμα "Hello world!" αντιστρέφοντας τη δοθείσα συμβολοσειρά. Το πρόγραμμα στη γλώσσα Edsger που λύνει αυτό το πρόβλημα δίνεται παρακάτω.

```
#include "stdio.h"
#include "string.h"

void main ()
{
   void reverse (char * s, char * r)
   {
      int i, l;

      for (i = 0, l = strlen(s); i < l; i++)
         r[i] = s[l-i-1];
      r[i] = '\0';
   }

   char p [20];

   reverse("\n!dlrow olleH", p);
   writeString(p);
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:

```
1: unit, reverse, -, -
2: :=, 0, -, i
3: par, s, V, -
4: par, $1, RET, -
5: call, -, -, strlen
6: :=, $1, -, l
```



Σχήμα 4: Εγγραφήματα δραστηριοποίησης για την αντιστροφή συμβολοσειράς.

```
 7: <, i, l, 9
 8: jump, -, -, 17
 9: array, r, i, $2
10: -, l, i, $3
11: -, $3, 1, $4
12: array, s, $4, $5
13: :=, [$5], -, [$2]
14: +, i, 1, $6
15: :=, $6, -, i
16: jump, -, -, 7
17: array, r, i, $7
18: :=, '\0', -, [$7]
19: endu, reverse, -, -

20: unit, main, -, -
21: par, "\n!dlrow olleH", V, -
22: par, p, V, -
23: call, -, -, reverse
24: par, p, V, -
25: call, -, -, writeString
26: endu, main, -, -
```

Τα εγγραφήματα δραστηριοποίησης για το παραπάνω πρόγραμμα φαίνονται στο σχήμα 4. Ο τελικός κώδικας δίνεται παρακάτω.

```
xseg        segment public 'code'
            assume  cs:xseg, ds:xseg, ss:xseg
            org     100h

main        proc    near
            call    near ptr _main_1
            mov     ax, 4C00h
            int     21h
main        endp

@1:
_reverse_2  proc    near
            push    bp
            mov     bp, sp
            sub     sp, 18

@2:
            mov     ax, 0
            mov     word ptr [bp-2], ax

@3:
            mov     ax, word ptr [bp+10]
            push    ax

@4:
            lea     si, word ptr [bp-6]
            push    si

@5:
            push    bp
            call    near ptr _strlen
            add     sp, 6

@6:
            mov     ax, word ptr [bp-6]
            mov     word ptr [bp-4], ax

@7:
            mov     ax, word ptr [bp-2]
            mov     dx, word ptr [bp-4]
            cmp     ax, dx
            jl      @9

@8:
            jmp     @17
```

```
@9:
            mov     ax, word ptr [bp-2]
            mov     cx, 1
            imul    cx
            mov     cx, word ptr [bp+8]
            add     ax, cx
            mov     word ptr [bp-8], ax

@10:
            mov     ax, word ptr [bp-4]
            mov     dx, word ptr [bp-2]
            sub     ax, dx
            mov     word ptr [bp-10], ax

@11:
            mov     ax, word ptr [bp-10]
            mov     dx, 1
            sub     ax, dx
            mov     word ptr [bp-12], ax

@12:
            mov     ax, word ptr [bp-12]
            mov     cx, 1
            imul    cx
            mov     cx, word ptr [bp+10]
            add     ax, cx
            mov     word ptr [bp-14], ax

@13:
            mov     di, word ptr [bp-14]
            mov     al, byte ptr [di]
            mov     di, word ptr [bp-8]
            mov     byte ptr [di], al

@14:
            mov     ax, word ptr [bp-2]
            mov     dx, 1
            add     ax, dx
            mov     word ptr [bp-16], ax

@15:
            mov     ax, word ptr [bp-16]
            mov     word ptr [bp-2], ax

@16:
            jmp     @7

@17:
            mov     ax, word ptr [bp-2]
            mov     cx, 1
            imul    cx
            mov     cx, word ptr [bp+8]
            add     ax, cx
            mov     word ptr [bp-18], ax

@18:
            mov     al, 0
            mov     di, word ptr [bp-18]
            mov     byte ptr [di], al

@19:
@reverse_2: mov     sp, bp
            pop     bp
            ret
_reverse_2  endp

@20:
_main_1     proc    near
            push    bp
            mov     bp, sp
            sub     sp, 20

@21:
            lea     si, byte ptr @str_1
            push    si

@22:
            lea     si, byte ptr [bp-20]
            push    si

@23:
            sub     sp, 2
            push    bp
            call    near ptr _reverse_2
            add     sp, 8

@24:
            lea     si, byte ptr [bp-20]
            push    si

@25:
            sub     sp, 2
            push    bp
            call    near ptr _writeString
            add     sp, 6

@26:
@main_1:    mov     sp, bp
            pop     bp
            ret
_main_1     endp

@str_1      db      10, '!dlrow olleH'
            db      0

            extrn   _strlen : proc
            extrn   _writeString : proc

xseg        ends
            end     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 Edsger 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 Edsger ότι τα στοιχεία του

είναι τα `x[0]`, `x[1]`, ..., `x[`$n-1$`]`), ο αλγόριθμος 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 Edsger είναι το εξής:

```
#include "stdio.h"

void bsort (int * x, int n)
{
   int  i;
   bool changed;

   void swap (byref int x, byref int y)
   {
      int t;

      t = x;
      x = y;
      y = t;
   }

   for (changed = true; changed;)
      for (i = 0, changed = false; i < n-1; i++)
         if (x[i] > x[i+1]) {
            swap(x[i], x[i+1]);
            changed = true;
         }
}

void main ()
{
   void printArray (char * msg, int * x, int n)
   {
      int i;

      writeString(msg);
      for (i = 0; i < n; i++) {
         if (i > 0)
            writeString(", ");
         writeInteger(x[i]);
      }
      writeString("\n");
   }

   int i, x[16], seed;

   for (i = 0, seed = 65; i < 16; i++)
      x[i] = seed = (seed * 137 + 221 + i) % 101;
   printArray("Initial array: ", x, 16);
   bsort(x, 16);
   printArray("Sorted array: ", x, 16);
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:

```
 1: unit, swap, -, -
 2: :=, x, -, t
 3: :=, y, -, x
 4: :=, t, -, y
 5: endu, swap, -, -

 6: unit, bsort, -, -
 7: :=, true, -, changed
 8: ifb, changed, -, 10
 9: jump, -, -, 31
10: :=, 0, -, i
11: :=, false, -, changed
12: -, n, 1, $1
13: <, i, $1, 15
14: jump, -, -, 8
15: array, x, i, $2
16: +, i, 1, $3
17: array, x, $3, $4
18: >, [$2], [$4], 20
19: jump, -, -, 27
20: array, x, i, $5
21: par, [$5], R, -
22: +, i, 1, $6
23: array, x, $6, $7
24: par, [$7], R, -
25: call, -, -, swap
26: :=, true, -, changed
27: +, i, 1, $8
28: :=, $8, -, i
29: jump, -, -, 12
30: jump, -, -, 8
31: endu, bsort, -, -

32: unit, printArray, -, -
33: par, msg, V, -
34: call, -, -, writeString
35: :=, 0, -, i
36: <, i, n, 38
37: jump, -, -, 48
38: >, i, 0, 40
39: jump, -, -, 42
40: par, ", ", V, -
41: call, -, -, writeString
42: array, x, i, $9
43: par, [$9], V, -
44: call, -, -, writeInteger
45: +, i, 1, $10
46: :=, $10, -, i
47: jump, -, -, 36
48: par, "\n", V, -
49: call, -, -, writeString
50: endu, printArray, -, -

51: unit, main, -, -
52: :=, 0, -, i
53: :=, 65, -, seed
54: <, i, 16, 56
55: jump, -, -, 66
56: *, seed, 137, $11
57: +, $11, 221, $12
58: +, $12, i, $13
59: %, $13, 101, $14
60: :=, $14, -, seed
61: array, x, i, $15
62: :=, seed, -, [$15]
63: +, i, 1, $16
64: :=, $16, -, i
65: jump, -, -, 54
66: par, "Initial array: ", V, -
67: par, x, V, -
68: par, 16, V, -
69: call, -, -, printArray
70: par, x, V, -
71: par, 16, V, -
72: call, -, -, bsort
73: par, "Sorted array: ", V, -
74: par, x, V, -
75: par, 16, V, -
76: call, -, -, printArray
77: endu, main, -, -
```

Τα εγγραφήματα δραστηριοποίησης για το παραπάνω πρόγραμμα φαίνονται στο σχήμα 5. Λόγω της πολυπλοκότητας του παραδείγματος, δίνεται τελικός κώδικας μόνο για λίγες από τις τετράδες του ενδιάμεσου κώδικα.

```
@15:
        mov     ax, word ptr [bp-2]
        mov     cx, 2
        imul    cx
        mov     cx, word ptr [bp+10]
        add     ax, cx
        mov     word ptr [bp-7], ax
...

@18:
        mov     di, word ptr [bp-7]
        mov     ax, word ptr [di]
        mov     di, word ptr [bp-11]
        mov     dx, word ptr [di]
        cmp     ax, dx
        jg      @20
...

@21:
        mov     si, word ptr [bp-13]
        push    si
...
```

Συνάρτηση **swap**

bp+10	**x**
bp+8	**y**
bp+6	Διεύθυνση αποτελέσματος
bp+4	Σύνδεσμος προσπέλασης
bp+2	Διεύθυνση επιστροφής
bp	Προηγούμενο **bp**
bp–2	**t**

Συνάρτηση **bsort**

x	bp+10
n	bp+8
Διεύθυνση αποτελέσματος	bp+6
Σύνδεσμος προσπέλασης	bp+4
Διεύθυνση επιστροφής	bp+2
Προηγούμενο **bp**	bp
i	bp–2
changed	bp–3
$1	bp–5
$2	bp–7
$3	bp–9
$4	bp–11
$5	bp–13
$6	bp–15
$7	bp–17
$8	bp–19

Συνάρτηση **printArray**

bp+12	**msg**
bp+10	**x**
bp+8	**n**
bp+6	Διεύθυνση αποτελέσματος
bp+4	Σύνδεσμος προσπέλασης
bp+2	Διεύθυνση επιστροφής
bp	Προηγούμενο **bp**
bp–2	**i**
bp–4	$9
bp–6	$10

Συνάρτηση **main**

Διεύθυνση αποτελέσματος	bp+6
Σύνδεσμος προσπέλασης	bp+4
Διεύθυνση επιστροφής	bp+2
Προηγούμενο **bp**	bp
i	bp–2
x	bp–34
seed	bp–36
$11	bp–38
$12	bp–40
$13	bp–42
$14	bp–44
$15	bp–46
$16	bp–48

Σχήμα 5: Εγγραφήματα δραστηριοποίησης για την ταξινόμηση με τη μέθοδο της φυσαλίδας.

```
@43:
        mov     di, word ptr [bp-4]
        mov     ax, word ptr [di]
        push    ax
```

```
...

@67:
        lea     si, word ptr [bp-34]
        push    si
```

3.6 Μέση τιμή τυχαίας μεταβλητής

Το πρόγραμμα που ακολουθεί υπολογίζει τη μέση τιμή μιας ακέραιας τυχαίας μεταβλητής, που μεταβάλλεται ομοιόμορφα στο διάστημα από 0 έως $n-1$. Η τιμή του n καθορίζεται από το χρήστη, όπως και το πλήθος k των δειγμάτων που χρησιμοποιούνται για τον υπολογισμό.

```
#include "stdio.h"

void main ()
{
   int    n, k, i, seed;
   double sum;

   writeString("Give n: ");
   n = readInteger();
   writeString("Give k: ");
   k = readInteger();

   for (i = 0, sum = 0.0, seed = 65; i < k; i++)
      sum += (double) (seed = (seed * 137 + 221 + i) % n);

   if (k > 0) {
      writeString("Mean: ");
      writeReal(sum / (double) k);
      writeString("\n");
   }
}
```

Ο ενδιάμεσος κώδικας που παράγει ο μεταγλωττιστής της Edsger είναι ο εξής:

```
 1: unit, main, -, -
 2: par, "Give n: ", V, -
 3: call, -, -, writeString
 4: par, $1, RET, -
 5: call, -, -, readInteger
 6: :=, $1, -, n
 7: par, "Give k: ", V, -
 8: call, -, -, writeString
 9: par, $2, RET, -
10: call, -, -, readInteger
11: :=, $2, -, k
12: :=, 0, -, i
13: :=, 0.0, -, sum
14: :=, 65, -, seed
15: <, i, k, 17
16: jump, -, -, 27
17: *, seed, 137, $3
18: +, $3, 221, $4
19: +, $4, i, $5
20: %, $5, n, $6
21: :=, $6, -, seed
22: +, sum, seed, $7
23: :=, $7, -, sum
24: +, i, 1, $8
25: :=, $8, -, i
26: jump, -, -, 15
27: >, k, 0, 29
28: jump, -, -, 37
29: par, "Mean: ", V, -
30: call, -, -, writeString
31: /, sum, k, $9
32: par, $9, V, -
33: call, -, -, writeReal
34: par, "\n", V, -
35: call, -, -, writeString
36: endu, main, -, -
```

Το εγγράφημα δραστηριοποίησης για το παραπάνω πρόγραμμα φαίνεται στο σχήμα 6. Ενδεικτικά δίνονται τμήματα του τελικού κώδικα για το παραπάνω πρόγραμμα. Έχουν παραλειφθεί όσα τμήματα δε σχετίζονται με πράξεις πραγματικών αριθμών. Επίσης, έχει γίνει η υπόθεση ότι η εκτέλεση θα γίνει σε επεξεργαστή 80286 ή νεώτερο, με μαθηματικό συνεπεξεργαστή.

```
xseg      segment public 'code'
          assume  cs:xseg, ds:xseg, ss:xseg
          org     100h

          .286
          .8087

main      proc    near
          call    near ptr _mean_1
          mov     ax, 4C00h
          int     21h
main      endp

@1:
_main_1   proc    near
          push    bp
          mov     bp, sp
          sub     sp, 52
...
```

Συνάρτηση **main**

bp+6	Διεύθυνση αποτελέσματος
bp+4	Σύνδεσμος προσπέλασης
bp+2	Διεύθυνση επιστροφής
bp	Προηγούμενο **bp**
bp–2	**n**
bp–4	**k**
bp–6	**i**
bp–8	**seed**
bp–18	**sum**
bp–20	$1
bp–22	$2
bp–24	$3
bp–26	$4
bp–28	$5
bp–30	$6
bp–40	$7
bp–42	$8
bp–52	$9

Σχήμα 6: Εγγράφημα δραστηριοποίησης για τη μέση τιμή τυχαίας μεταβλητής.

```
@13:
        fld     tbyte ptr @real_1
        fstp    tbyte ptr [bp-18]
...

@22:
        fld     tbyte ptr [bp-18]
        fild    word ptr [bp-8]
        faddp   ST(1), ST(0)
        fstp    tbyte ptr [bp-40]
@23:
        fld     tbyte ptr [bp-40]
        fstp    tbyte ptr [bp-18]
...

@31:
        fld     tbyte ptr [bp-18]
        fild    word ptr [bp-4]
        fdivp   ST(1), ST(0)
        fstp    tbyte ptr [bp-52]
@32:
        fld     tbyte ptr [bp-52]
        sub     sp, 10
        mov     si, sp
        fstp    tbyte ptr [si]
@33:
        sub     sp, 2
        push    bp
        call    near ptr _writeReal
        add     sp, 14
...

@36:
@main_1:  mov     sp, bp
          pop     bp
          ret
_main_1   endp

          extrn   _writeString : proc
          extrn   _readInteger : proc
          extrn   _writeReal : proc

@str_1    db      'Give n: '
          db      0

@str_2    db      'Give k: '
          db      0

@str_3    db      'Mean: '
          db      0

@str_4    db      10, 0

@real_1   dt      0.0

xseg      ends
          end     main
```

Από την εκτέλεση του προγράμματος διαπιστώνει κανείς ότι η μέση τιμή που προκύπτει διαφέρει σημαντικά από τη θεωρητική μέση τιμή $(n-1)/2$, που θα έπρεπε να προκύπτει όταν το k γίνει αρκετά μεγάλο. Αυτό οφείλεται στον απλοϊκό αλγόριθμο που έχει χρησιμοποιηθεί για τη γέννηση ψευδοτυχαίων αριθμών, λόγω του οποίου η μεταβλητή απέχει αρκετά από το να είναι τυχαία.

4 Οδηγίες για την παράδοση

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 (πχ. `*.eds`) 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.eds` 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 `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` είναι το παρακάτω.

```
CFLAGS=-Wall -g

compiler: compiler.lex.o compiler.tab.o symbol.o symbolc.o

$(CC) $(CFLAGS) -o compiler compiler.lex.o compiler.tab.o \
                           symbol.o interm.o final.o

compiler.lex.c: compiler.l compiler.tab.h
        flex -st compiler.l > compiler.lex.c

compiler.tab.c compiler.tab.h: compiler.y
        bison -dv compiler.y

clean:
        $(RM) *.o compiler.tab.c compiler.tab.h compiler.lex.c

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
  ```