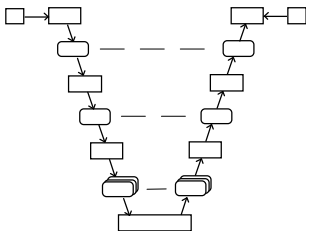
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Μεταγλωττιστές 2014

Θέμα εργασίας

Η γλώσσα Pazcal♭



Μεταγλωττιστές

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

Διδάσκοντες: *Νίκος Παπασπύρου*
Κωστής Σαγώνας

Βοηθοί: *Άγγελος Γιάντσιος*
Γιάννης Τσιούρης

Αθήνα, Απρίλιος 2014

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από κάθε ομάδα δυο σπουδαστών ένας μεταγλωττιστής για τη γλώσσα **Pazcal**$^\flat$. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C/C++, Java, SML, OCaml ή Haskell. Η επιλογή κάποιας άλλης γλώσσας υλοποίησης μπορεί να γίνει κατόπιν συνεννόησης με τους διδάσκοντες. Επιτρέπεται να χρησιμοποιηθούν και επίσης συνιστάται η χρήση εργαλείων, π.χ. `flex`/`ML-Lex`/`ocamllex`Alex, `bison`/`ML-Yacc`/`ocamlyacc`Happy, `JavaCC`, κ.λπ. Περισσότερες πληροφορίες σχετικές με αυτά τα εργαλεία θα δοθούν στις παραδόσεις.

Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή φαίνονται στον παρακάτω πίνακα. Τις αντίστοιχες ημερομηνίες παράδοσης θα βρείτε στο Moodle.

Τμήμα του μεταγλωττιστή	*Μονάδες*	*Bonus*
Λεκτικός αναλυτής	0.5	—
Συντακτικός αναλυτής	0.5	0.1
Σημασιολογικός αναλυτής	1.0	0.2
Ενδιάμεσος κώδικας	1.0	0.3
Βελτιστοποίηση	1.0	0.7
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**!

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 σε τμήματα της εργασίας που είναι προαιρετικά. Συγκεκριμένα:

- (20%) Πραγματικοί αριθμοί (`REAL`).
- (20%) Δομή `switch`.
- (30%) Βελτιστοποίηση ενδιάμεσου κώδικα.
  Οτιδήποτε απαιτεί ανάλυση ροής ελέγχου ή/και ανάλυση ροής δεδομένων.
- (30%) Δέσμευση καταχωρητών και βελτιστοποίηση τελικού κώδικα.

Περιεχόμενα

1 Περιγραφή της γλώσσας Pazcalᵇ

Η γλώσσα Pazcalᵇ είναι μια απλή γλώσσα προστακτικού προγραμματισμού — είναι αυτή που διδάσκεται στο πρώτο εξάμηνο της ΣΗΜΜΥ από φέτος. Βασίζεται σε ένα γνήσιο υποσύνολο της ISO C99 (ή της ISO C++11), με αρκετές απλοποιήσεις και μερικά macros που κατά σύμβαση γράφονται με κεφαλαία γράμματα για να ξεχωρίζουν και που εδώ εμφανίζονται ως ενσωματωμένα στη γλώσσα. Τα κύρια χαρακτηριστικά της Pazcalᵇ εν συντομία είναι τα εξής:

- Σύνταξη εντολών και εκφράσεων παρόμοια με αυτή της C.
- Συναρτήσεις με τους κανόνες εμβέλειας της C.
- Βασικοί τύποι δεδομένων για ακέραιους και πραγματικούς αριθμούς, λογικές τιμές και χαρακτήρες.
- Πίνακες με γνωστό ή άγνωστο μέγεθος.
- Βιβλιοθήκη συναρτήσεων.

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

1.1 Λεκτικές μονάδες

Οι λεκτικές μονάδες της γλώσσας Pazcalᵇ χωρίζονται στις παρακάτω κατηγορίες (προσοχή στη διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων):

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

`and`	`bool`	`break`	`case`	`char`	`const`
`continue`	`default`	`do`	`DOWNTO`	`else`	`false`
`FOR`	`FORM`	`FUNC`	`if`	`int`	`MOD`
`NEXT`	`not`	`or`	`PROC`	`PROGRAM`	`REAL`
`return`	`STEP`	`switch`	`TO`	`true`	`while`
`WRITE`	`WRITELN`	`WRITESP`	`WRITESPLN`		

- 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 πεζά γράμματα θεωρούνται διαφορετικά από τα αντίστοιχα κεφαλαία, επομένως τα ονόματα `foo`, `Foo` και `FOO` είναι όλα διαφορετικά. Επίσης, 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. Απαγορεύεται μία μη μηδενική ακέραια σταθερά να αρχίζει με μηδέν. Παραδείγματα ακέραιων σταθερών είναι τα ακόλουθα:

  `0` `42` `1284` `20404` αλλά όχι: `00200`.

- Τις *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`

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)

- 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'    '\"
```

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

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

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

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

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

```
&    ;    .    (    )    :    ,    [    ]    {    }
```

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα **Pazcal**$^\flat$ μπορεί επίσης να περιέχει τα παρακάτω, τα οποία 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 ακολουθία χαρακτήρων `*/`. Τα σχόλια αυτής της μορφής δεν επιτρέπεται να είναι φωλιασμένα.

1.2 Τύποι δεδομένων

Η **Pazcal**$^\flat$ υποστηρίζει τέσσερις βασικούς τύπους δεδομένων:

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

Εκτός από τους βασικούς τύπους, η **Pazcal**$^\flat$ υποστηρίζει επίσης τους παρακάτω σύνθετους τύπους, η κατασκευή των οποίων βασίζεται σε άλλους βασικούς ή σύνθετους τύπους:

- πίνακες (arrays), αποτελούμενοι από n στοιχεία τύπου t. Το n θα πρέπει να είναι ακέραια σταθερά με θετική τιμή και το t έγκυρος τύπος.
- πίνακες (arrays), αποτελούμενοι από άγνωστο αριθμό στοιχείων τύπου t. Το t πρέπει να είναι έγκυρος τύπος.

Οι τύποι `char` και `int` ονομάζονται *ακέραιοι* τύποι. Οι ακέραιοι τύποι και ο τύπος `REAL` ονομάζονται *αριθμητικοί* τύποι. Οι βασικοί τύποι και οι τύποι πινάκων δεδομένου μεγέθους ονομάζονται *πλήρεις* τύποι. Αντίθετα, οι τύποι πινάκων άγνωστου μεγέθους ονομάζονται *ημιτελείς* τύποι. Κατά το σχηματισμό ενός τύπου πίνακα, είτε πλήρους είτε ημιτελούς, ο τύπος του στοιχείου t θα πρέπει να είναι πλήρης. Η αρίθμηση των στοιχείων ενός πίνακα ακολουθεί τη σύμβαση της C: το πρώτο στοιχείο έχει δείκτη 0, το δεύτερο 1, ενώ το τελευταίο έχει δείκτη κατά ένα μικρότερο από την πραγματική διάσταση του πίνακα.

Ο αριθμός των bytes που καταλαμβάνουν τα δεδομένα κάθε πλήρους τύπου στη μνήμη του υπολογιστή, καθώς και ο ακριβής τρόπος παράστασης αυτών εξαρτώνται από την υλοποίηση της **Pazcal**$^\flat$. Κατ' ελάχιστον θα πρέπει να είναι:

- `int`: μέγεθος ≥ 2 bytes.
- `bool`: μέγεθος 1 byte, τιμές `false` $= 0$ και `true` $= 1$.
- `char`: μέγεθος 1 byte, παράσταση σύμφωνα με τον πίνακα ASCII.
- `REAL`: μέγεθος 10 bytes, παράσταση σύμφωνα με το πρότυπο IEEE 754.
- πίνακες n στοιχείων τύπου t: μέγεθος ίσο με n επί το μέγεθος του τύπου t, τοποθέτηση σε αύξουσα σειρά διευθύνσεων.

1.3 Δομή του προγράμματος

Ένα πρόγραμμα **Pazcal**$^\flat$ αποτελείται από μία ακολουθία δηλώσεων. Κάθε δήλωση μπορεί να είναι:

- Ορισμός σταθεράς.
- Ορισμός καθολικής μεταβλητής.
- Δήλωση ή ορισμός ρουτίνας.
- Ορισμός κύριου προγράμματος.

Η **Pazcal**$^\flat$ ακολουθεί τους κανόνες εμβέλειας της C, όσον αφορά στην ορατότητα των ονομάτων μεταβλητών, ρουτίνων και παραμέτρων. Ειδικότερα για τον ορισμό σταθερών και μεταβλητών, η εμβέλεια ενός ονόματος αρχίζει ακριβώς από το σημείο που αναγράφεται για πρώτη φορά το όνομα.

1.3.1 Σταθερές

Οι δηλώσεις σταθερών γίνονται με τη λέξη κλειδί `const`. Ακολουθεί ένας βασικός τύπος και ένα ή περισσότερα ονόματα σταθερών, καθένα από τα οποία συνοδεύεται από μία τιμή. Οι τιμές των σταθερών είναι σταθερές εκφράσεις, δηλαδή εκφράσεις που περιέχουν μόνο σταθερές και που μπορούν να αποτιμηθούν στο χρόνο μεταγλώττισης. Παραδείγματα δηλώσεων σταθερών είναι:

```
const int N = 1000, M = 100;
const int N2 = N*N;
const char EOLN = '\n';
```

1.3.2 Μεταβλητές

Οι δηλώσεις μεταβλητών γίνονται με την αναγραφή ενός βασικού τύπου, συνοδευόμενου από ένα ή περισσότερα ονόματα δηλωτών (declarators). Ένας δηλωτής μπορεί να είναι είτε ένα απλό αναγνωριστικό, είτε ένας δηλωτής πίνακα. Στη δεύτερη περίπτωση, το αναγνωριστικό του ακολουθείται από μία ή περισσότερες διαστάσεις, κάθε μία από τις οποίες περιέχει μία ακέραια έκφραση μέσα σε αγκύλες. Το πρώτο είδος δηλωτή χρησιμοποιείται για τη δήλωση μεταβλητών βασικών τύπων, ενώ το δεύτερο για τη δήλωση πινάκων.

Αν ο τύπος μίας μεταβλητής είναι βασικός, τότε το όνομά της μπορεί να συνοδεύεται από μία αρχικοποίηση. Οι αρχικές τιμές των καθολικών μεταβλητών, αν υπάρχουν, πρέπει να είναι σταθερές εκφράσεις. Αντίθετα, οι αρχικές τιμές των υπόλοιπων μεταβλητών, αν υπάρχουν, μπορούν να είναι οποιεσδήποτε εκφράσεις.

Οι καθολικές μεταβλητές που δε συνοδεύονται από αρχικοποίηση αυτόματα αρχικοποιούνται σε μηδενικές τιμές. Μηδενικές τιμές για τους τύπους `int`, `bool`, `char` και `REAL` θεωρούνται αντίστοιχα οι `0`, `false`, `'\0'` και `0.0`. Για τύπους πινάκων, μηδενική τιμή θεωρείται ένας πίνακας όλα τα στοιχεία του οποίου έχουν μηδενικές τιμές.

Παραδείγματα δηλώσεων μεταβλητών είναι:

```
int i = 42, j, k = 17;
REAL x, pi = 3.14, y;
char s[80], t[25][80];
```

1.3.3 Ρουτίνες

Οι ρουτίνες διακρίνονται σε *διαδικασίες* (procedures) και *συναρτήσεις* (functions). Κάθε ρουτίνα είναι μια δομική μονάδα και αποτελείται από την επικεφαλίδα της και το σώμα της. Στη *δήλωση* μίας ρουτίνας μπορεί να παραλείπεται το σώμα, αργότερα όμως η ρουτίνα πρέπει να *οριστεί* χρησιμοποιώντας την ίδια επικεφαλίδα.

Η επικεφαλίδα μίας ρουτίνας ξεκινά με μία από τις λέξεις κλειδιά `PROC` ή `FUNC`, ανάλογα αν η ρουτίνα είναι διαδικασία ή συνάρτηση. Στην περίπτωση της συνάρτησης, μετά τη λέξη κλειδί `FUNC` γράφεται ο τύπος επιστροφής, που πρέπει να είναι βασικός τύπος. Ακολουθεί το όνομα της ρουτίνας και οι τυπικές παράμετροι μέσα σε παρενθέσεις. Κάθε τυπική παράμετρος χαρακτηρίζεται από τον τύπο της και το όνομά της. Η δήλωση των παραμέτρων μοιάζει με τη δήλωση των μεταβλητών, χωρίς αρχικοποίηση. Τυπικές παράμετροι βασικών τύπων μπορούν να περνούν κατ' αξία (by value) ή κατ' αναφορά (by reference). Στη δεύτερη περίπτωση εμφανίζεται το σύμβολο `&` πριν από το όνομα της παραμέτρου. Τυπικές παράμετροι τύπων πίνακα περνούν υποχρεωτικά κατ' αναφορά, χωρίς αναγραφή του συμβόλου `&`. Στις τυπικές παραμέτρους τύπου πίνακα, το μέγεθος της πρώτης διάστασης μπορεί να παραλείπεται (ημιτελής τύπος).

Ακολουθούν παραδείγματα δηλώσεων ρουτίνων.

```
PROC p1 ();
PROC p2 (int n);
PROC p3 (int a, int b, bool &c);
FUNC REAL f1 (REAL x);
```

```
FUNC int f2 (char s[]);
FUNC REAL f3 (int a[10], REAL &x, REAL b[][10][20]);
```

1.4 Εκφράσεις

Κάθε έκφραση της **Pazcal**$^\flat$ διαθέτει ένα μοναδικό τύπο. Αν η αποτίμησή της ολοκληρωθεί, τότε προκύπτει 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 το όνομά τους από τη θέση τους σε μια εντολή ανάθεσης: οι l-values εμφανίζονται στο αριστερό μέλος της ανάθεσης ενώ οι r-values στο δεξιό. Οι 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 και οι παράμετροι των ρουτίνων. Συγκεκριμένα:

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

- Αν l είναι μια l-value τύπου πίνακα αποτελούμενου από στοιχεία τύπου t 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 i τότε αυτή η l-value αντιστοιχεί στο στοιχείο με δείκτη i του πίνακα που αντιστοιχεί στην l. Η τιμή του i δεν πρέπει να υπερβαίνει τα πραγματικά όρια του πίνακα.

Μια l-value βασικού τύπου μπορεί να χρησιμοποιηθεί ως έκφραση. Η τιμή αυτής της έκφρασης είναι ίση με την τιμή η οποία περιέχεται στο αντικείμενο που αντιστοιχεί στην l-value.

1.4.2 Σταθερές

Στις r-values της γλώσσας **Pazcal**$^\flat$ 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 λογικές σταθερές `true` και `false`, με τύπο `bool` και προφανείς τιμές.

- Οι πραγματικές σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `REAL` 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.

- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο πίνακα αποτελούμενου από $n + 1$ χαρακτήρες, όπου n είναι ο αριθμός χαρακτήρων που περιέχονται στη συμβολοσειρά. Κάθε τέτοια σταθερά αντιστοιχεί σε ένα σταθερό αντικείμενο τύπου πίνακα, 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. Οι σταθερές συμβολοσειρές είναι το μόνο είδος σταθεράς τύπου πίνακα που επιτρέπεται στην **Pazcal**$^\flat$. Δεν επιτρέπεται να μεταβληθούν τα περιεχόμενά τους κατά το χρόνο εκτέλεσης.

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της **Pazcal**$^\flat$.

Τελεστές	Περιγραφή	Αριθμός τελουμένων	Θέση και προσεταιριστικότητα
`+ - ! not`	Πρόσημα και λογική άρνηση	1	prefix
`* / % MOD`	Πολλαπλασιασμός και διαίρεση	2	infix, αριστερή
`+ -`	Πρόσθεση και αφαίρεση	2	infix, αριστερή
`> < <= >=`	Ανισότητα	2	infix, αριστερή
`== !=`	Ισότητα	2	infix, αριστερή
`&& and`	Λογική σύζευξη	2	infix, αριστερή
`|| or`	Λογική διάζευξη	2	infix, αριστερή

1.4.3 Τελεστές

Οι τελεστές της **Pazcal**$^\flat$ διακρίνονται σε τελεστές με ένα τελούμενο και τελεστές με δύο τελούμενα. Οι πρώτοι γράφονται πρίν το τελούμενο (prefix), ενώ οι δεύτεροι γράφονται μεταξύ των τελουμένων (infix). Η σειρά αποτίμησης των τελουμένων των τελεστών με δυο τελούμενα δεν καθορίζεται. Όλοι οι τελεστές της **Pazcal**$^\flat$ έχουν ως αποτέλεσμα r-value.

- 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.
- Οι τελεστές `!` και `not` είναι ισοδύναμοι και υλοποιούν τη λογική άρνηση. Το τελούμενό πρέπει να είναι τύπου `bool`, και τον ίδιο τύπο έχει και το αποτέλεσμα.
- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `/`, `%` και `MOD` υλοποιούν κατά σειρά τις αριθμητικές πράξεις της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού, της διαίρεσης (ακέραιας ή πραγματικής) και του υπόλοιπου της ακέραιας διαίρεσης. Οι τελεστές `%` και `MOD` είναι ισοδύναμοι. Τα τελούμενα όλων των παραπάνω τελεστών πρέπει να είναι εκφράσεις αριθμητικών τύπων. Στην περίπτωση των τελεστών `+`, `-`, `*` και `/`, αν και τα δύο τελούμενα είναι ακέραιου τύπου τότε το αποτέλεσμα είναι τύπου `int`, διαφορετικά το αποτέλεσμα είναι τύπου `REAL`. Στην περίπτωση των τελεστών `%` και `MOD`, τα τελούμενα πρέπει να είναι ακέραιου τύπου και το αποτέλεσμα είναι τύπου `int`.
- Οι τελεστές `==`, `!=`, `<`, `>`, `<=` και `>=` υλοποιούν τις σχέσεις ισότητας και ανισότητας μεταξύ αριθμών. Τα τελούμενα πρέπει να είναι και τα δύο αριθμητικά και το αποτέλεσμα είναι τύπου `bool`.
- Οι τελεστές `&&` και `and` είναι ισοδύναμοι και υλοποιούν την πράξη της λογικής σύζευξης. Ομοίως, οι τελεστές `||` και `or` είναι ισοδύναμοι και υλοποιούν την πράξη της λογικής 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.

Στον πίνακα 2 ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της **Pazcal**$^\flat$.

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. Η έννοια της *συμβατότητας για ανάθεση* περιγράφεται στην ενότητα 1.5.

- Αν η τυπική παράμετρος είναι τύπου t και περνά κατ' αξία, τότε ο τύπος t' της αντίστοιχης πραγματικής παραμέτρου πρέπει να είναι συμβατός για ανάθεση με τον τύπο t. Η αντιγραφή της τιμής της πραγματικής παραμέτρου στην τυπική παράμετρο γίνεται ακριβώς όπως στην ανάθεση.

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

  - Αν ο τύπος t είναι βασικός τύπος ή πλήρης τύπος πίνακα, τότε πρέπει $t' = t$.
  - Αν ο τύπος t είναι ημιτελής τύπος πίνακα, τότε πρέπει είτε $t' = t$, ή ο τύπος t' να είναι πλήρης τύπος πίνακα με τον ίδιο τύπο στοιχείου όπως ο t.

Κατά την κλήση μιας συνάρτησης, η σειρά με την οποία αποτιμώνται οι πραγματικές παράμετροι δεν καθορίζεται.

1.5 Εντολές

Οι εντολές που υποστηρίζει η γλώσσα **Pazcal**$^\flat$ είναι οι ακόλουθες:

- Η κενή εντολή, που δεν κάνει καμία ενέργεια.

- Η εντολή απλής ανάθεσης l `=` e, όπου l μια l-value τύπου t και e μια έκφραση τύπου t'. Ο τύπος t' πρέπει να είναι *συμβατός για ανάθεση* με τον τύπο t. Αυτή η σχέση συμβατότητας, που πρέπει να τονιστεί ότι δεν είναι συμμετρική, ορίζεται ως εξής:

  - Κάθε βασικός τύπος είναι συμβατός για ανάθεση με τον εαυτό του.
  - Ο τύπος `int` είναι συμβατός για ανάθεση με τον τύπο `REAL`. Στην περίπτωση αυτή, οι ακέραιες τιμές μετατρέπονται σε πραγματικές.
  - Ο τύπος `char` είναι συμβατός για ανάθεση με τον τύπο `int`. Οι τιμές χαρακτήρων μετατρέπονται σε ακέραιες τιμές στο διάστημα από 0 έως 255.
  - Ο τύπος `int` είναι συμβατός για ανάθεση με τον τύπο `char`. Οι ακέραιες τιμές μετατρέπονται σε τιμές χαρακτήρων κρατώντας μόνο τα 8 λιγότερο σημαντικά bits τους.

- Η εντολή σύνθετης ανάθεσης $l \diamond$`=` e, όπου $\diamond$`=` ένας από τους τελεστές ανάθεσης `+=`, `-=`, `*=`, `/=` και `%=`, l μια l-value αριθμητικού τύπου e μια έκφραση αριθμητικού τύπου. Η εντολή αυτή έχει ακριβώς την ίδια σημασιολογία, ως προς τον έλεγχο τύπων και ως προς την εκτέλεση, με την εντολή l `=` $l \diamond e$ με τη μόνη διαφορά ότι η l-value l αποτιμάται μόνο μία φορά.

- Οι εντολές αύξησης l `++` και μείωσης l `--`, που είναι ισοδύναμες με l `+= 1` και l `-= 1`, αντίστοιχα.

- Η κλήση μιας διαδικασίας. Συντακτικά και σημασιολογικά συμπίπτει με την κλήση μιας συνάρτησης, με τη διαφορά ότι δεν υπάρχει αποτέλεσμα.

- Η σύνθετη εντολή, που αποτελείται από μια σειρά έγκυρων τοπικών δηλώσεων (σταθερών ή μεταβλητών) ή εντολών που περικλείονται από τα σύμβολα `{` και `}`. Οι δηλώσεις και οι εντολές αυτές εκτελούνται διαδοχικά, εκτός αν κάποια από αυτές είναι εντολή άλματος. Η εμβέλεια των δηλώσεων περιορίζεται στο εσωτερικό της σύνθετης εντολής.

- 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 (` i `,` r `)` s. Η μεταβλητή ελέγχου i πρέπει να είναι τύπου `int`. Η περιοχή r πρέπει να είναι της μορφής:

  - *lower* `TO` *upper*
  - *upper* `DOWNTO` *lower*

– *lower* `TO` *upper* `STEP` *step*

– *upper* `DOWNTO` *lower* `STEP` *step*

όπου *lower*, *upper* και *step* είναι ακέραιες εκφράσεις. Αν παραλείπεται το `STEP`, εννοείται 1. Σε κάθε περίπτωση, τα όρια *lower* και *upper* καθώς και το βήμα *step* αποτιμώνται μία φορά (με σειρά που δεν καθορίζεται) πριν την εκτέλεση του βρόχου. Η τιμή του *step* πρέπει να είναι θετική.

Η εκτέλεση του βρόχου `FOR` γίνεται ως εξής. Η μεταβλητή ελέγχου παίρνει την αρχική τιμή, δηλαδή *lower* σε περίπτωση περιοχής `TO` ή *upper* σε περίπτωση περιοχής `DOWNTO`. Αν η τιμή της μεταβλητής ελέγχου i είναι εντός ορίων (δηλαδή $i \leq$ *upper* σε περίπτωση περιοχής `TO` ή $i \geq$ *lower* σε περίπτωση περιοχής `DOWNTO`), τότε εκτελείται το σώμα του βρόχου. Στη συνέχεια η μεταβλητή ελέγχου παίρνει την επόμενη τιμή και ο βρόχος επαναλαμβάνεται με τον έλεγχο ορίων. Η επόμενη τιμή είναι ίση με την προηγούμενη τιμή συν την τιμή του βήματος σε περίπτωση περιοχής `TO` ή με την προηγούμενη τιμή μείον την τιμή του βήματος σε περίπτωση περιοχής `DOWNTO`.

Για τον υπολογισμό της επόμενης τιμής της μεταβλητής ελέγχου δε λαμβάνεται υπόψη η τιμή της *στο τέλος* της προηγούμενης επανάληψης αλλά εκείνη *στην αρχή* της προηγούμενης επανάληψης. Αυτό είναι σημαντικό στην περίπτωση που η τιμή της μεταβλητής ελέγχου αλλάξει κατά την εκτέλεση του σώματος του βρόχου. Το παρακάτω παράδειγμα

```
FOR (i, 1 TO 10) {
  WRITELN(i);
  i += 100;
  WRITELN(i);
}
```

θα εκτυπώσει κατά σειρά τις τιμές 1, 101, 2, 102, 3, 103, ..., 10, 110. Μετά την εκτέλεση του βρόχου, η τιμή της μεταβλητής ελέγχου δεν καθορίζεται.

- Οι εντολές βρόχων `while (` e `)` s και `do` s `while (` e `)` με την ίδια σημασιολογία όπως στη C. Η έκφραση e πρέπει να έχει τύπο `bool` και το s να είναι έγκυρη εντολή.

- Η εντολή `switch (` e `) { ... }`, με την ίδια σημασιολογία όπως στη C. Η έκφραση e και οι σταθερές που εμφανίζονται μετά τα `case` πρέπει να είναι ακέραιου τύπου. Κάθε σκέλος του `switch` τελειώνει με μία από τις λέξεις κλειδιά `break` ή `NEXT`. Η πρώτη διακόπτει την εκτέλεση του `switch` (όπως στη C) ενώ η δεύτερη συνεχίζει την εκτέλεση και στο επόμενο σκέλος (όπως η απουσία του `break` στη C).

- Η εντολή άλματος `break`, η οποία προκαλεί την άμεση έξοδο από τον εσωτερικότερο βρόχο `FOR`, `while` ή `do...while`. Η εντολή άλματος `break` δεν πρέπει να είναι φωλιασμένη σε σκέλος εντολής `switch` εσωτερικότερης της εντολής βρόχου στην οποία αντιστοιχεί, π.χ.

```
switch (x) {
  case 1:
    while (true)
      if (something()) break;   // OK --- break the loop
    break;                      // exit the switch
}

while (true) {
  switch (x) {
    case 1:
      if (something()) break;   // WRONG --- break nested in switch!
    break;
  }
}
```

- Η εντολή άλματος `continue`, η οποία προκαλεί την άμεση μετακίνηση στην επόμενη επανάληψη του εσωτερικότερου βρόχου `FOR`, `while` ή `do...while`.
- Η εντολή `return`, που επιστρέφει τερματίζοντας την εκτέλεση της τρέχουσας ρουτίνας. Αν η ρουτίνα είναι συνάρτηση, η εντολή αυτή συνοδεύεται από μία έκφραση, η τιμή της οποίας θα επιστραφεί ως αποτέλεσμα της συνάρτησης.
- Οι εντολές εκτύπωσης `WRITE`, `WRITELN`, `WRITESP` και `WRITESPLN`. Η σύνταξή τους μοίαζει με κλήση διαδικασίας, όμως μπορούν να έχουν οσεσδήποτε πολλές παραμέτρους. Κάθε παράμετρος πρέπει να είναι βασικού τύπου ή τύπου πίνακα χαρακτήρων.

  Στη θέση οπιασδήποτε παραμέτρου των εντολών εκτύπωσης μπορούν να χρησιμοποιηθούν οι ειδικές μορφές `FORM(`x`,`w`)` και `FORM(`x`,`w`,`d`)` που αντιστοιχούν στα x`:`w και x`:`w`:`d της Pascal. Το w είναι μία έκφραση τύπου `int` που παριστάνει το εύρος του πεδίου (δηλαδή το ελάχιστο πλήθος χαρακτήρων που θα εκτυπωθούν, οι υπόλοιποι θα συμπληρωθούν από αριστερά με κενά διαστήματα). Το d είναι μία έκφραση τύπου `int` που παριστάνει το πλήθος των δεκαδικών ψηφίων που θα χρησιμοποιηθούν — αν ορίζεται, τότε ο τύπος του x πρέπει να είναι `REAL`.

  Η εντολή `WRITE` εκτυπώνει τις τιμές των παραμέτρων τη μία μετά την άλλη, χωρίς διαχωριστικό. Η εντολή `WRITELN` κάνει ό,τι η `WRITE` αλλά στο τέλος αλλάζει γραμμή. Η εντολή `WRITESP` κάνει ό,τι η `WRITE` αλλά εκτυπώνει ένα κενό διάστημα μεταξύ των τιμών των παραμέτρων της, ανά δύο. Τέλος, η εντολή `WRITESPLN` κάνει ό,τι η `WRITESP` αλλά στο τέλος αλλάζει γραμμή.

1.6 Προκαθορισμένες ρουτίνες

Η **Pazcal**$^\flat$ υποστηρίζει ένα σύνολο προκαθορισμένων ρουτίνων, οι οποίες βρίσκονται στη διάθεση του προγραμματιστή. Οι ρουτίνες αυτές είναι ορατές σε κάθε δομική μονάδα, εκτός αν επισκιάζονται από μεταβλητές, παραμέτρους ή άλλες ρουτίνες με το ίδιο όνομα. Παρακάτω δίνονται οι επικεφαλίδες τους, όπως θα γράφονταν αν τις ορίζαμε σε **Pazcal**$^\flat$. Επίσης, εξηγείται η λειτουργία τους.

1.6.1 Είσοδος και έξοδος

```
PROC putchar       (char c);
PROC puts          (char s[]);
PROC WRITE_INT     (int n, int w);
PROC WRITE_BOOL    (bool b, int w);
PROC WRITE_CHAR    (char c, int w);
PROC WRITE_REAL    (REAL r, int w, int d);
PROC WRITE_STRING  (char s[], int w);
```

Οι διαδικασίες `putchar` και `puts` έχουν την ίδια σημασιολογία όπως στη C. Οι υπόλοιπες διαδικασίες μπορούν να χρησιμοποιηθούν στο μεταγλωττιστή σας για την υλοποίηση των εντολών εκτύπωσης. Οι παράμετροι `w` και `d` έχουν την ίδια σημασία όπως στις ειδικές μορφές `FORM`.

```
FUNC int  READ_INT    ();
FUNC bool READ_BOOL   ();
FUNC int  getchar     ();
FUNC REAL READ_REAL   ();
PROC      READ_STRING (int size, char s[]);
```

Αντίστοιχα, τα παραπάνω υποπρογράμματα χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της **Pazcal**$^\flat$ και για την εισαγωγή συμβολοσειρών. Η διαδικασία `READ_STRING` 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. Η συνάρτηση `getchar` έχει την ίδια σημασιολογία όπως στη C.

1.6.2 Μαθηματικές συναρτήσεις

```
FUNC int  abs  (int n);
FUNC REAL fabs (REAL r);
```

Η απόλυτη τιμή ενός ακέραιου ή πραγματικού αριθμού.

```
FUNC REAL sqrt   (REAL r);
FUNC REAL sin    (REAL r);
FUNC REAL cos    (REAL r);
FUNC REAL tan    (REAL r);
FUNC REAL arctan (REAL r);
FUNC REAL exp    (REAL r);
FUNC REAL ln     (REAL r);
FUNC REAL pi     ();
```

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

1.6.3 Συναρτήσεις μετατροπής

```
FUNC REAL trunc (REAL r);
FUNC REAL round (REAL r);
FUNC INT  TRUNC (REAL r);
FUNC INT  ROUND (REAL r);
```

Η `trunc` επιστρέφει τον πλησιέστερο ακέραιο αριθμό, η απόλυτη τιμή του οποίου είναι μικρότερη από την απόλυτη τιμή του `r`. Η `round` επιστρέφει τον πλησιέστερο ακέραιο αριθμό. Σε περίπτωση αμφιβολίας, προτιμάται ο αριθμός με τη μεγαλύτερη απόλυτη τιμή. Οι αντίστοιχες συναρτήσεις με κεφαλαία επιστρέφουν ακέραιο αριθμό αντί πραγματικού.

1.6.4 Συναρτήσεις διαχείρισης συμβολοσειρών

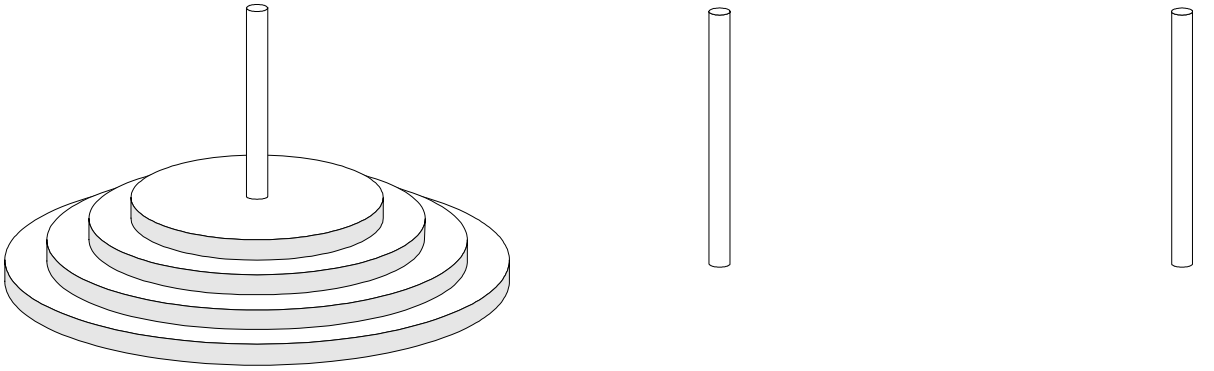
```
FUNC int strlen (char s[]);
FUNC int strcmp (char s1[], char s2[]);
PROC     strcpy (char trg[], char src[]);
PROC     strcat (char trg[], char src[]);
```

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

2 Πλήρης γραμματική της Pazcal♭

Η σύνταξη της γλώσσας Pazcal♭ 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 στον πίνακα 2. Τα σύμβολα ⟨id⟩, ⟨int-const⟩, ⟨float-const⟩, ⟨char-const⟩ και ⟨string-literal⟩ είναι τερματικά σύμβολα της γραμματικής και αντιστοιχούν στις λεκτικές μονάδες που περιγράφονται στη ενότητα 1.1.

```
1  ⟨module⟩          ::= ( ⟨declaration⟩ )*
2  ⟨declaration⟩     ::= ⟨const_def⟩ | ⟨var_def⟩ | ⟨routine⟩ | ⟨program⟩
3  ⟨const_def⟩       ::= "const" ⟨type⟩ ⟨id⟩ "=" ⟨const_expr⟩ ( "," ⟨id⟩ "=" ⟨const_expr⟩ )* ";"
4  ⟨var_def⟩         ::= ⟨type⟩ ⟨var_init⟩ ( "," ⟨var_init⟩ )* ";"
5  ⟨var_init⟩        ::= ⟨id⟩ [ "=" ⟨expr⟩ ] | ⟨id⟩ ( "[" ⟨const_expr⟩ "]" )+
6  ⟨routine_header⟩  ::= ( "PROC" | "FUNC" ⟨type⟩ ) ⟨id⟩ "(" [ ⟨type⟩ ⟨formal⟩ ( "," ⟨type⟩ ⟨formal⟩ )* ] ")"
7  ⟨formal⟩          ::= [ "&" ] ⟨id⟩ | ⟨id⟩ "[" [ ⟨const_expr⟩ ] "]" ( "[" ⟨const_expr⟩ "]" )*
8  ⟨routine⟩         ::= ⟨routine_header⟩ ( ";" | ⟨block⟩ )
9  ⟨program_header⟩  ::= "PROGRAM" ⟨id⟩ "(" ")"
10 ⟨program⟩         ::= ⟨program_header⟩ ⟨block⟩
11 ⟨type⟩            ::= "int" | "bool" | "char" | "REAL"
12 ⟨const_expr⟩      ::= ⟨expr⟩
13 ⟨expr⟩            ::= ⟨int-const⟩ | ⟨float-const⟩ | ⟨char-const⟩ | ⟨string-literal⟩ | "true" | "false"
14                     | "(" ⟨expr⟩ ")" | ⟨l_value⟩ | ⟨call⟩ | ⟨unop⟩ ⟨expr⟩ | ⟨expr⟩ ⟨binop⟩ ⟨expr⟩
15 ⟨l_value⟩         ::= ⟨id⟩ ( "[" ⟨expr⟩ "]" )*
16 ⟨unop⟩            ::= "+" | "-" | "!" | "not"
17 ⟨binop⟩           ::= "+" | "-" | "*" | "/" | "%" | "MOD" | "==" | "!=" | "<" | ">" | "<=" | ">="
18                     | "&&" | "and" | "||" | "or"
19 ⟨call⟩            ::= ⟨id⟩ "(" [ ⟨expr⟩ ( "," ⟨expr⟩ )* ] ")"
20 ⟨block⟩           ::= "{" ( ⟨local_def⟩ | ⟨stmt⟩ )* "}"
21 ⟨local_def⟩       ::= ⟨const_def⟩ | ⟨var_def⟩
22 ⟨stmt⟩            ::= ";" | ⟨l_value⟩ ⟨assign⟩ ⟨expr⟩ ";" | ⟨l_value⟩ ( "++" | "--" ) ";" | ⟨call⟩ ";"
23                     | "if" "(" ⟨expr⟩ ")" ⟨stmt⟩ [ "else" ⟨stmt⟩ ] | "while" "(" ⟨expr⟩ ")" ⟨stmt⟩
24                     | "FOR" "(" ⟨id⟩ "," ⟨range⟩ ")" ⟨stmt⟩ | "do" ⟨stmt⟩ "while" "(" ⟨expr⟩ ")" ";"
25                     | "switch" "(" ⟨expr⟩ ")" "{" ( ( "case" ⟨const_expr⟩ ":" )+ ⟨clause⟩ )*
26                         [ "default" ":" ⟨clause⟩ ] "}"
27                     | "break" ";" | "continue" ";" | "return" [ ⟨expr⟩ ] ";" | ⟨block⟩
28                     | ⟨write⟩ "(" [ ⟨format⟩ ( "," ⟨format⟩ )* ] ")" ";"
29 ⟨assign⟩          ::= "=" | "+=" | "-=" | "*=" | "/=" | "%="
30 ⟨range⟩           ::= ⟨expr⟩ ( "TO" | "DOWNTO" ) ⟨expr⟩ [ "STEP" ⟨expr⟩ ]
31 ⟨clause⟩          ::= ( ⟨stmt⟩ )* ( "break" ";" | "NEXT" ";" )
32 ⟨write⟩           ::= "WRITE" | "WRITELN" | "WRITESP" | "WRITESPLN"
33 ⟨format⟩          ::= ⟨expr⟩ | "FORM" "(" ⟨expr⟩ "," ⟨expr⟩ [ "," ⟨expr⟩ ] ")"
34
```



Σχήμα 1: Οι πύργοι του Hanoi.

3 Παραδείγματα

Στην παράγραφο αυτή δίνονται πέντε παραδείγματα προγραμμάτων στη γλώσσα **Pazcal**$^\flat$, 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, ή στο Moodle.

3.1 Πες γειά!

Το παρακάτω παράδειγμα είναι το απλούστερο πρόγραμμα στη γλώσσα **Pazcal**$^\flat$ που παράγει κάποιο αποτέλεσμα ορατό στο χρήστη. Το πρόγραμμα αυτό τυπώνει απλώς ένα μήνυμα.

```
1   PROGRAM hello ()
2   {
3     WRITELN(”Hello world!”);
4   }
```

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 **Pazcal**$^\flat$ που λύνει αυτό το πρόβλημα δίνεται παρακάτω. Η συνάρτηση `hanoi` είναι αναδρομική.

```
PROC hanoi (char source[], char target[], char auxiliary[], int rings)
{
  if (rings > 0) {
    hanoi(source, auxiliary, target, rings-1);
    WRITESPLN(”Move from”, source, ”to”, target);
    hanoi(auxiliary, target, source, rings-1);
  }
}

PROGRAM solve ()
{
  WRITE(”Please, give the number of rings: ”);
  int numberOfRings = READ_INT();
  WRITELN(”\nHere is the solution:\n”);
  hanoi(”left”, ”right”, ”middle”, numberOfRings);
}
```

3.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα **Pazcal**$^\flat$ 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
$t := 6$
όσο $t \leq n$ κάνε τα εξής:
    αν ο αριθμός $t - 1$ είναι πρώτος τότε τύπωσέ τον
    αν ο αριθμός $t + 1$ είναι πρώτος τότε τύπωσέ τον
    $t := t + 6$

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
$i := 3$
όσο $i \leq t/2$ κάνε τα εξής:
    αν ο t διαιρείται με τον i τότε ο t δεν είναι πρώτος
    $i := i + 2$
ο t είναι πρώτος

Το αντίστοιχο πρόγραμμα σε γλώσσα **Pazcal**$^\flat$ είναι το εξής:

```
FUNC bool prime (int n)
{
  if (n < 0)                return prime(-n);
  else if (n < 2)           return false;
  else if (n == 2)          return true;
  else if (n MOD 2 == 0)    return false;
  else {
    int i;
```

```
9         FOR (i, 3 TO n / 2 STEP 2)
10          if (n MOD i == 0) return false;
11        return true;
12      }
13    }
14
15    PROGRAM primes ()
16    {
17      WRITE("Please, give the upper limit: ");
18      int limit = READ_INT();
19      WRITESPLN("Prime numbers between 0 and", limit);
20      WRITELN();
21      int counter = 0;
22      if (limit >= 2) { WRITELN(2); counter++; }
23      if (limit >= 3) { WRITELN(3); counter++; }
24      int number;
25      FOR (number, 6 TO limit STEP 6) {
26        if (prime(number-1))                    { WRITELN(number-1); counter++; }
27        if (number != limit and prime(number+1)) { WRITELN(number+1); counter++; }
28      }
29      WRITELN();
30      WRITESPLN(counter, "prime number(s) were found.");
31    }
```

3.4 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 **Pazcal**$^\flat$ ταξινομεί έναν πίνακα ακεραίων αριθμών κατ' αύξουσα σειρά, χρησιμοποιώντας μια παραλλαγή αυτού του αλγορίθμου. Αν `x` είναι ο πίνακας που πρέπει να ταξινομηθεί και `n` είναι το μέγεθός του (θεωρούμε σύμφωνα με τη σύμβαση της **Pazcal**$^\flat$ 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 **Pazcal**$^\flat$ είναι το εξής:

```
1     PROC swap (int &x, int &y)
2     {
3       int t = x; x = y; y = t;
4     }
5
6     PROC bsort (int n, int x[])
7     {
8       bool changed = true;
9       while (changed) {
10        changed = false;
11        int i;
12        FOR (i, 0 TO n-2)
13          if (x[i] > x[i+1]) { swap(x[i], x[i+1]); changed = true; }
14      }
15    }
```

```
16
17    PROC print (char msg[], int n, int x[])
18    {
19      WRITE(msg);
20      int i;
21      FOR (i, 0 TO n-1) {
22        if (i > 0) WRITE(”, ”);
23        WRITE(x[i]);
24      }
25      WRITELN();
26    }
27
28    PROGRAM bubbles ()
29    {
30      int seed = 65;
31      int i, x[16];
32      FOR (i, 0 TO 15) {
33        seed = (seed * 137 + 221 + i) MOD 101;
34        x[i] = seed;
35      }
36      print(”Initial array: ”, 16, x);
37      bsort(16, x);
38      print(”Sorted array: ”, 16, x);
39    }
```

3.5 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.

```
1     PROGRAM mean ()
2     {
3       WRITE(”Give n: ”);
4       int n = READ_INT();
5       WRITE(”Give k: ”);
6       int k = READ_INT();
7       int i, seed = 65;
8       REAL sum = 0.0;
9
10      FOR (i, 0 TO k-1) {
11        seed = (seed * 137 + 221 + i) MOD n;
12        sum += seed;
13      }
14
15      if (k > 0) WRITESPLN(”Mean:”, sum / k);
16    }
```

Από την εκτέλεση του προγράμματος διαπιστώνει κανείς ότι η μέση τιμή που εκτυπώνεται 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. `*.pzc`) 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.pzc` 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.

`-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 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: compiler.lex.o compiler.tab.o symbol.o symbolc.o
        $(CC) $(CFLAGS) -o compiler compiler.lex.o compiler.tab.o \
                            symbol.o symbolc.o

compiler.lex.c: compiler.l compiler.tab.h
        flex -it 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 \
              compiler core
```

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

- Ενδιάμεσος κώδικας: να υιοθετηθεί μορφοποίηση ισοδύναμη με

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

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

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