



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

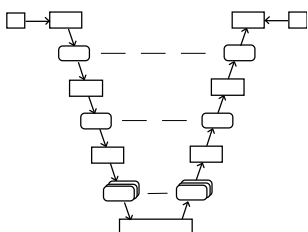
Μεταγλωττιστές 2026

Θέμα εργασίας

# Η γλώσσα PCL



Niklaus Wirth (1934–2024)



**Μεταγλωττιστές**

<https://courses.softlab.ntua.gr/compilers/>

Διδάσκων: Κωστής Σαγώνας

Αθήνα, Φεβρουάριος 2026



## ΘΕΜΑ:

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

### Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή και η κατανομή μονάδων φαίνονται στον παρακάτω πίνακα. Οι καταληκτικές ημερομηνίες παράδοσης της εργασίας θα ανακοινωθούν μετά από συνεννόηση.

<i>Τμήμα του μεταγλωττιστή</i>	<i>Μονάδες</i>	<i>Bonus χωρίς χρήση LLVM</i>
Λεκτικός αναλυτής	0.5	–
Συντακτικός αναλυτής	1.0	–
Σημασιολογικός αναλυτής	2.0	–
Ενδιάμεσος κώδικας	2.0	–
Βελτιστοποίηση	0.5	+0.5
Τελικός κώδικας	–	+0.5
<i>Βαθμός συνολικής εργασίας</i>	6.0	+1.0

### Προαιρετικά τμήματα και μονάδες bonus

Το σύνολο των μονάδων της παρούσας εργασίας είναι 6 αν η υλοποίηση του μεταγλωττιστή γίνει με χρήση LLVM, όπως φαίνεται στην δεύτερη στήλη του παραπάνω πίνακα. Αν γίνει με χρήση κάποιας δικής σας ενδιάμεσης γλώσσας και παραγωγής και βελτιστοποίησης κώδικα από αυτή, υπάρχει μια επιπλέον μονάδα bonus (που σημαίνει ότι το σύνολο μονάδων του μαθήματος είναι 11 ή 12).



# Περιεχόμενα

<b>1 Περιγραφή της γλώσσας PCL</b>	<b>5</b>
1.1 Λεκτικές μονάδες	5
1.2 Τύποι δεδομένων	6
1.3 Δομή του προγράμματος	7
1.3.1 Μεταβλητές	7
1.3.2 Υποπρογράμματα	8
1.4 Εκφράσεις	8
1.4.1 L-values	8
1.4.2 Σταθερές	9
1.4.3 Τελεστές	9
1.4.4 Κλήση συναρτήσεων	10
1.5 Εντολές	11
1.6 Προκαθορισμένα υποπρογράμματα	12
1.6.1 Είσοδος και έξοδος	12
1.6.2 Μαθηματικές συναρτήσεις	12
1.6.3 Συναρτήσεις μετατροπής	12
<b>2 Πλήρης γραμματική της PCL</b>	<b>13</b>
<b>3 Παραδείγματα</b>	<b>13</b>
3.1 Πες γεια!	14
3.2 Οι πύργοι του Hanoi	14
3.3 Πρώτοι αριθμοί	15
3.4 Αντιστροφή συμβολοσειράς	17
3.5 Ταξινόμηση με τη μέθοδο της φουσαλίδας	17
3.6 Μέση τιμή τυχαίας μεταβλητής	19
<b>4 Οδηγίες για την παράδοση</b>	<b>19</b>

# 1 Περιγραφή της γλώσσας PCL

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

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

Περαιτέρω λεπτομέρειες της γλώσσας δίνονται στις παραγράφους που ακολουθούν. Λόγω των πολλών ομοιοτήτων της PCL με την Pascal, 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 PCL χωρίζονται στις παρακάτω κατηγορίες:

- Τις λέξεις κλειδιά, οι οποίες είναι οι παρακάτω:

and	array	begin	boolean	char	dispose	div	do
else	end	false	forward	function	goto	if	integer
label	mod	new	nil	not	of	or	procedure
program	real	result	return	then	true	var	while

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

0      42      1284      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

- Τις *σταθερούς χαρακτήρες*, που αποτελούνται από έναν 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 τον χαρακτήρα \ και περιγράφονται στον πίνακα 1. Παραδείγματα σταθερών χαρακτήρων είναι τα εξής:

'a'      '1'      '\n'      '\"'

Πίνακας 1: Ακολουθίες διαφυγής (escape sequences).

Ακολουθία διαφυγής	Περιγραφή
<code>\n</code>	ο χαρακτήρας αλλαγής γραμμής (line feed)
<code>\t</code>	ο χαρακτήρας στηλοθέτησης (tab)
<code>\r</code>	ο χαρακτήρας επιστροφής στην αρχή της γραμμής (carriage return)
<code>\0</code>	ο χαρακτήρας με ASCII κωδικό 0
<code>\\</code>	ο χαρακτήρας <code>\</code> (backslash)
<code>\'</code>	ο χαρακτήρας <code>'</code> (απλό εισαγωγικό)
<code>\"</code>	ο χαρακτήρας <code>"</code> (διπλό εισαγωγικό)

- 1 • 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:

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

- 4  
5 • Τους *τελεστές*, οι οποίοι είναι οι παρακάτω:

6 `= > < <> >= <= + - * / ^ @`

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

8 `:= ; . ( ) : , [ ]`

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

- 11 • *Κενούς χαρακτήρες*, δηλαδή ακολουθίες αποτελούμενες από κενά διαστήματα (space), χαρακτήρες  
12 στηλοθέτησης (tab), χαρακτήρες αλλαγής γραμμής (line feed) ή χαρακτήρες επιστροφής στην αρχή  
13 της γραμμής (carriage return).  
14 • *Σχόλια*, τα οποία αρχίζουν με την ακολουθία χαρακτήρων (`*` και τερματίζονται με την πρώτη με-  
15 τέπειτα εμφάνιση της ακολουθίας χαρακτήρων `*`). Κατά συνέπεια, τα σχόλια δεν επιτρέπεται να  
16 είναι φωλιασμένα. Στο εσωτερικό τους επιτρέπεται η εμφάνιση οποιουδήποτε χαρακτήρα.

## 17 1.2 Τύποι δεδομένων

18 Η PCL υποστηρίζει τέσσερις βασικούς τύπους δεδομένων:

- 19 • `integer` : ακέραιοι αριθμοί,  
20 • `boolean` : λογικές τιμές,  
21 • `char` : χαρακτήρες, και  
22 • `real` : πραγματικοί αριθμοί.

23 Εκτός από τους βασικούς τύπους, η PCL υποστηρίζει επίσης τους παρακάτω σύνθετους τύπους, η κατα-  
24 σκευή των οποίων βασίζεται σε άλλους βασικούς ή σύνθετους τύπους:

- 25 • `array [n] of t` : πίνακες (arrays), αποτελούμενοι από  $n$  στοιχεία τύπου  $t$ . Το  $n$  θα πρέπει να είναι  
26 ακέραιο σταθερά με θετική τιμή και το  $t$  έγκυρος τύπος.  
27 • `array of t` : πίνακες (arrays), αποτελούμενοι από άγνωστο αριθμό στοιχείων τύπου  $t$ . Το  $t$  πρέπει  
28 να είναι έγκυρος τύπος.  
29 • `^t` : δείκτες (pointers) σε ένα στοιχείο τύπου  $t$ . Το  $t$  πρέπει να είναι έγκυρος τύπος.

1 Παραδείγματα σύνθετων τύπων είναι:

```
array [10] of integer
array of ^integer
2 ^array of array [10] of boolean
```

3 Οι τύποι `integer` και `real` ονομάζονται *αριθμητικοί* τύποι. Οι βασικοί τύποι, οι τύποι πινάκων δε-  
4 δομένου μεγέθους και οι τύποι δεικτών ονομάζονται *πλήρεις* τύποι. Αντίθετα, οι τύποι πινάκων άγνω-  
5 στου μεγέθους ονομάζονται *ημιτελείς* τύποι. Κατά το σχηματισμό ενός τύπου πίνακα, είτε πλήρους είτε  
6 ημιτελούς, ο τύπος του στοιχείου *t* θα πρέπει να είναι πλήρης. Η αρίθμηση των στοιχείων ενός πίνακα  
7 ακολουθεί τη σύμβαση της C: το πρώτο στοιχείο έχει δείκτη 0, το δεύτερο 1, ενώ το τελευταίο έχει δείκτη  
8 κατά ένα μικρότερο από την πραγματική διάσταση του πίνακα.

9 Ο αριθμός των bytes που καταλαμβάνουν τα δεδομένα κάθε πλήρους τύπου στη μνήμη του υπολο-  
10 γιστή, καθώς και ο ακριβής τρόπος παράστασης αυτών εξαρτώνται από την υλοποίηση της PCL. Στη  
11 συνέχεια κάνουμε τις ακόλουθες παραδοχές που διευκολύνουν την κατασκευή μεταγλωττιστών της PCL  
12 για υπολογιστές βασισμένους στους επεξεργαστές x86\_64.

- 13 • `integer` : μέγεθος τουλάχιστον 4 bytes, παράσταση συμπληρώματος ως προς 2.
- 14 • `boolean` : μέγεθος 1 byte, τιμές `false` = 0 και `true` = 1.
- 15 • `char` : μέγεθος 1 byte, παράσταση σύμφωνα με τον πίνακα ASCII.
- 16 • `real` : μέγεθος τουλάχιστον 8 bytes, παράσταση σύμφωνα με το πρότυπο IEEE 754.
- 17 • `array [ n ] of t` : μέγεθος ίσο με *n* επί το μέγεθος του τύπου *t*, τοποθέτηση σε αύξουσα σειρά  
18 διευθύνσεων.
- 19 • `^t` : μέγεθος 8 bytes.

### 20 1.3 Δομή του προγράμματος

21 Ένα πρόγραμμα PCL αποτελείται από την επικεφαλίδα και το σώμα της κύριας δομικής μονάδας. Η  
22 επικεφαλίδα αυτής είναι της μορφής:

```
23 program p;
```

24 όπου *p* το όνομα του προγράμματος. Το σώμα κάθε δομικής μονάδας μπορεί να περιέχει προαιρετικά:

- 25 • Δηλώσεις μεταβλητών.
- 26 • Δηλώσεις ετικετών.
- 27 • Ορισμούς υποπρογραμμάτων.
- 28 • Δηλώσεις υποπρογραμμάτων, οι ορισμοί των οποίων θα ακολουθήσουν.

29 Τελευταίο συστατικό στον ορισμό μιας δομικής μονάδας είναι το σώμα της, δηλαδή μια σύνθετη εντολή,  
30 η οποία καθορίζει τη λειτουργία της δομικής μονάδας. Στην περίπτωση της κυρίας δομικής μονάδας, η  
31 εκτέλεση του προγράμματος ξεκινά από αυτή τη σύνθετη εντολή.

32 Η PCL ακολουθεί τους κανόνες εμβέλειας της ISO Pascal, όσον αφορά στην ορατότητα των ονομάτων  
33 μεταβλητών, υποπρογραμμάτων και παραμέτρων.

#### 34 1.3.1 Μεταβλητές

35 Οι δηλώσεις μεταβλητών γίνονται με τη λέξη κλειδί `var`. Ακολουθούν ένα ή περισσότερα ονόματα μετα-  
36 βλητών και ένας τύπος δεδομένων, ο οποίος πρέπει να είναι πλήρης. Περισσότερες συνεχόμενες δηλώσεις  
37 μεταβλητών μπορούν να γίνουν παραλείποντας τη λέξη κλειδί `var`. Παραδείγματα δηλώσεων είναι:

```
38 var i    : integer;
39     x, y : real;
40 var s    : array [80] of char;
```

## 1.3.2 Υποπρογράμματα

Τα υποπρογράμματα διακρίνονται σε *διαδικασίες* (procedures) και *συναρτήσεις* (functions). Κάθε υπο-  
πρόγραμμα είναι μια δομική μονάδα και αποτελείται από την επικεφαλίδα του και το σώμα του. Η δομή  
του σώματος έχει ήδη περιγραφεί. Στην επικεφαλίδα αναφέρεται κατ' αρχήν αν το υποπρόγραμμα είναι  
διαδικασία ή συνάρτηση, το όνομά του, οι τυπικές του παράμετροι μέσα σε παρενθέσεις και, αν πρόκειται  
για συνάρτηση, ο τύπος του αποτελέσματος, ο οποίος δεν μπορεί να είναι τύπος πίνακα. Οι παρενθέσεις  
είναι υποχρεωτικές ακόμα και αν ένα υποπρόγραμμα δεν έχει τυπικές παραμέτρους.

Κάθε τυπική παράμετρος χαρακτηρίζεται από το όνομα, τον τύπο της και τον τρόπο περάσματος. Η  
PCL υποστηρίζει πέρασμα παραμέτρων κατ' αξία (by value) και κατ' αναφορά (by reference). Μία τυπική  
παράμετρος περνά κατ' αναφορά αν στη δήλωσή της έχει προηγηθεί η λέξη κλειδί var, διαφορετικά  
περνά κατ' αξία. Οι τυπικές παράμετροι που περνούν κατ' αξία δεν μπορούν να έχουν τύπο πίνακα.

Ακολουθούν παραδείγματα επικεφαλίδων υποπρογραμμάτων.

```
procedure p1 ();  
procedure p2 (n : integer);  
procedure p3 (a, b : integer; var b : boolean);  
function f1 (x : real) : real;  
function f2 (var s : array of char) : integer;  
function f3 (n : integer; x : real) : ^array of real;
```

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

## 1.4 Εκφράσεις

Κάθε έκφραση της PCL έχει μοναδικό τύπο<sup>1</sup> και μπορεί να αποτιμηθεί δίνοντας ως αποτέλεσμα μια τιμή  
αυτού του τύπου. Οι εκφράσεις διακρίνονται σε δύο κατηγορίες: αυτές που προκύπτουν από l-values,  
που 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.

Τόσο οι l-values όσο και οι r-values μπορούν να εμφανίζονται μέσα σε παρενθέσεις, που χρησιμο-  
ποιούνται για λόγους ομαδοποίησης.

### 1.4.1 L-values

Οι l-values αντιπροσωπεύουν αντικείμενα που καταλαμβάνουν χώρο στη μνήμη κατά την εκτέλεση του  
προγράμματος και τα οποία μπορούν να περιέχουν τιμές. Τέτοια αντικείμενα είναι οι μεταβλητές, οι πα-  
ράμετροι των υποπρογραμμάτων, οι μεταβλητές που κατασκευάζονται με δυναμική παραχώρηση μνήμης  
και οι θέσεις όπου φυλάσσονται τα αποτελέσματα των συναρτήσεων. Συγκεκριμένα:

- Το όνομα μιας μεταβλητής ή μιας παραμέτρου υποπρογράμματος είναι l-value και αντιστοιχεί στο  
εν λόγω αντικείμενο. Ο τύπος της l-value είναι ο τύπος του αντίστοιχου αντικειμένου.
- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1 είναι l-values. Έχουν τύπο  
array [n] of char, όπου n είναι ο αριθμός χαρακτήρων που περιέχονται στη συμβολοσειρά προ-  
σαυξημένος κατά ένα. Κάθε τέτοια l-value αντιστοιχεί σε ένα 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 PCL.

<sup>1</sup>Εξάιρεση αποτελεί η σταθερά μηδενικού δείκτη nil, η οποία δεν έχει μοναδικό τύπο. Περιγράφεται στην ενότητα 1.4.2.

- Αν  $l$  είναι μια l-value τύπου `array [n] of t` ή τύπου `array of t`, και  $e$  είναι μια έκφραση τύπου `integer`, τότε  $l[e]$  είναι μια l-value με τύπο  $t$ . Αν η τιμή της έκφρασης  $e$  είναι ο μη αρνητικός ακέραιος  $n$  τότε αυτή η l-value αντιστοιχεί στο στοιχείο με δείκτη  $n$  του πίνακα που αντιστοιχεί στην  $l$ . Η τιμή του  $n$  δεν πρέπει να υπερβαίνει τα πραγματικά όρια του πίνακα.
- Αν  $e$  είναι μια έκφραση τύπου  $t$ , τότε  $e^{\wedge}$  είναι μια l-value με τύπο  $t$ , που αντιστοιχεί στο αντικείμενο όπου δείχνει η τιμή της  $e$ .
- Στο σώμα μιας συνάρτησης που επιστρέφει αποτέλεσμα τύπου  $t$ , η λέξη κλειδί `result` είναι μια l-value με τύπο  $t$ , που αντιστοιχεί στο αντικείμενο όπου φυλάσσεται το αποτέλεσμα της συνάρτησης. Το αντικείμενο αυτό είναι προσωρινό και δεν πρέπει να χρησιμοποιείται μετά την επιστροφή από τη συνάρτηση.

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 PCL συγκαταλέγονται οι ακόλουθες σταθερές:

- Οι ακέραιες σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `integer` και η τιμή τους είναι ίση με τον μη αρνητικό ακέραιο αριθμό που παριστάνουν.
- Οι λογικές σταθερές `true` και `false`, με τύπο `boolean` 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.
- Η λέξη κλειδί `nil` που έχει τύπο  $t$ , για κάθε έγκυρο τύπο  $t$ , και η τιμή της οποίας είναι ο μηδενικός δείκτης. Είναι η μόνη έκφραση της PCL που δεν έχει μοναδικό τύπο. Ο μηδενικός δείκτης απαγορεύεται να αποδεικτοδοτηθεί με χρήση του τελεστή  $^{\wedge}$ .

### 1.4.3 Τελεστές

Οι τελεστές της PCL διακρίνονται σε τελεστές με ένα τελούμενο και τελεστές με δύο τελούμενα. Από τους πρώτους, ορισμένοι γράφονται πριν το τελούμενο (`prefix`) και ορισμένοι μετά (`postfix`), ενώ οι δεύτεροι γράφονται πάντα μεταξύ των τελουμένων (`infix`). Η αποτίμηση των τελουμένων των τελεστών με δυο τελούμενα γίνεται από αριστερά προς τα δεξιά.

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  $^{\wedge}$ . Οι δύο αυτοί τελεστές είναι οι μοναδικοί που έχουν ως αποτέλεσμα l-value. Στη συνέχεια περιγράφονται οι υπόλοιποι τελεστές της PCL, που έχουν ως αποτέλεσμα r-value.

- 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 `not` υλοποιεί τη λογική άρνηση. Το τελούμενό του πρέπει να είναι τύπου `boolean`, και τον ίδιο τύπο έχει και το αποτέλεσμα.
- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `/`, `div` και `mod` υλοποιούν αντίστοιχα τις αριθμητικές πράξεις της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού, της πραγματικής διαίρεσης, του ποιλίκου

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της PCL.

Τελεστές	Περιγραφή	Αριθμός τελούμενων	Θέση και προσεταιριστικότητα
[ ]	Αναφορά σε στοιχείο πίνακα	2	ειδική
@	Διεύθυνση αντικειμένου	1	prefix
^	Αποδεικτοδότηση	1	postfix
+ -	Πρόσημα	1	prefix
not	Λογική άρνηση	1	prefix
* / div mod and	Πολλαπλασιαστικοί τελεστές	2	infix, αριστερή
+ - or	Προσθετικοί τελεστές	2	infix, αριστερή
= > < <= >= <>	Σχισιακοί τελεστές	2	infix, καμία

και του υπόλοιπου της ακεραίας διαίρεσης. Τα τελούμενά τους πρέπει να είναι εκφράσεις αριθμητικών τύπων. Στην περίπτωση των τελεστών +, - και \*, αν και τα δύο τελούμενα είναι τύπου integer τότε και το αποτέλεσμα είναι τύπου integer, διαφορετικά το αποτέλεσμα είναι τύπου real. Στην περίπτωση του τελεστή /, το αποτέλεσμα είναι πάντα τύπου real. Τέλος, στην περίπτωση των τελεστών div και mod, τα τελούμενα πρέπει να είναι τύπου integer και το αποτέλεσμα είναι επίσης τύπου integer.

- Οι τελεστές = και <> υλοποιούν αντίστοιχα την ισότητα και την ανισότητα. Το αποτέλεσμα είναι τύπου boolean. Τα τελούμενα πρέπει να είναι είτε και τα δύο αριθμητικά, οπότε συγκρίνονται οι αριθμητικές τιμές τους, είτε του ίδιου τύπου, ο οποίος δεν πρέπει να είναι τύπος πίνακα. Στη δεύτερη περίπτωση, συγκρίνονται οι δυαδικές αναπαραστάσεις των τιμών των τελούμενων.
- Οι τελεστές <, >, <= και >= υλοποιούν τις σχέσεις ανισότητας μεταξύ αριθμών. Τα τελούμενα πρέπει να είναι και τα δύο αριθμητικά και το αποτέλεσμα είναι τύπου boolean.
- Οι τελεστές and και or υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης. Τα τελούμενα πρέπει να είναι τύπου boolean 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 PCL.

#### 1.4.4 Κλήση συναρτήσεων

Αν  $f$  είναι το όνομα μιας συνάρτησης με αποτέλεσμα τύπου  $t$ , τότε η έκφραση  $f(e_1, \dots, e_n)$  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  $t$ .
- Αν η τυπική παράμετρος είναι τύπου  $t$  και περνά κατ' αναφορά, τότε η αντίστοιχη πραγματική παράμετρος πρέπει να είναι l-value τύπου  $t'$ , όπου ο τύπος  $\wedge t'$  πρέπει να είναι συμβατός για ανάθεση με τον τύπο  $\wedge t$ .

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

## 1 1.5 Εντολές

2 Οι εντολές που υποστηρίζει η γλώσσα PCL είναι οι ακόλουθες:

- 3 • Η κενή εντολή, που δεν κάνει καμία ενέργεια.
- 4 • Η εντολή ανάθεσης  $l := e$ , όπου  $l$  μια l-value τύπου  $t$  και  $e$  μια έκφραση τύπου  $t'$ . Ο τύπος  $t'$  πρέπει  
5 να είναι συμβατός για ανάθεση με τον τύπο  $t$ . Αυτή η σχέση συμβατότητας, που πρέπει να τονιστεί  
6 ότι δεν είναι συμμετρική, ορίζεται ως εξής:
  - 7 – Κάθε πλήρης τύπος είναι συμβατός για ανάθεση με τον εαυτό του.
  - 8 – Ο τύπος `integer` είναι συμβατός για ανάθεση με τον τύπο `real`.
  - 9 – Ο τύπος `^array [n] of t` είναι συμβατός για ανάθεση με τον τύπο `^array of t`.
- 10 • Η σύνθετη εντολή, που αποτελείται από μια σειρά έγκυρων εντολών χωρισμένων με το διαχωριστή  
11 `;`, ανάμεσα στις λέξεις κλειδιά `begin` και `end`. Οι εντολές αυτές εκτελούνται διαδοχικά, εκτός αν  
12 κάποια από αυτές είναι εντολή άλματος.
- 13 • Η εντολή ελέγχου `if e then s1 else s2`. Η έκφραση  $e$  πρέπει να έχει τύπο `boolean` και τα  $s_1, s_2$   
14 να είναι έγκυρες εντολές. Το τμήμα `else` είναι προαιρετικό.
- 15 • Η εντολή βρόχου `while e do s`. Η έκφραση  $e$  πρέπει να έχει τύπο `boolean` και το  $s$  να είναι έγκυρη  
16 εντολή.
- 17 • Η εντολή με ετικέτα `I : s`, όπου  $I$  το όνομα μιας ετικέτας και  $s$  μια έγκυρη εντολή. Κάθε ετικέτα  
18 πρέπει να δηλώνεται στο τμήμα δηλώσεων της δομικής μονάδας και να ορίζεται το πολύ μια φορά  
19 στη σύνθετη εντολή που ορίζει το σώμα αυτής.
- 20 • Η εντολή άλματος `goto I`, όπου  $I$  το όνομα μιας ετικέτας που πρέπει να εμφανίζεται στην ίδια  
21 δομική μονάδα. Πέραν αυτού, δεν υπάρχουν άλλοι περιορισμοί ως προς τη θέση των εντολών  
22 αλμάτων ή των ετικετών όπου αυτά οδηγούν.
- 23 • Η εντολή `return`, που επιστρέφει τερματίζοντας την εκτέλεση της δομικής μονάδας.
- 24 • Η κλήση μιας διαδικασίας. Συντακτικά και σημασιολογικά συμπίπτει με την κλήση μιας συνάρτη-  
25 σης, με τη διαφορά ότι δεν επιστρέφεται αποτέλεσμα.
- 26 • Η εντολή `new` με την οποία γίνεται δυναμική παραχώρηση μνήμης, και η οποία εμφανίζεται σε δυο  
27 μορφές:
  - 28 – `new l`, όπου  $l$  πρέπει να είναι μια l-value τύπου `^t` και  $t$  πρέπει να είναι πλήρης τύπος. Μετά  
29 την εκτέλεση της εντολής, ο δείκτης που αντιστοιχεί στην  $l$  τοποθετείται να δείχνει προς ένα  
30 νέο δυναμικό αντικείμενο τύπου  $t$ .
  - 31 – `new [e] l`, όπου  $l$  πρέπει να είναι μια l-value τύπου `^array of t` και  $e$  μια έκφραση τύ-  
32 που `integer`. Η τιμή της  $e$  πρέπει να είναι ένας θετικός αριθμός  $n$ . Μετά την εκτέλεση της  
33 εντολής, ο δείκτης που αντιστοιχεί στην  $l$  τοποθετείται να δείχνει προς ένα νέο δυναμικό  
34 αντικείμενο τύπου `array [n] of t`.
- 35 • Η εντολή `dispose` με την οποία γίνεται η αποδέσμευση της μνήμης που έχει παραχωρηθεί δυνα-  
36 μικά με την εντολή `new`. Εμφανίζεται και αυτή σε δυο μορφές:
  - 37 – `dispose l`, όπου  $l$  πρέπει να είναι μια l-value τύπου `^t` και  $t$  πρέπει να είναι πλήρης τύπος.  
38 Η τρέχουσα τιμή της  $l$  πρέπει να είναι ένας δείκτης προς ένα δυναμικό αντικείμενο που έχει  
39 κατασκευαστεί με χρήση της εντολής `new`. Μετά την εκτέλεση της εντολής, η  $l$  περιέχει το  
40 μηδενικό δείκτη.
  - 41 – `dispose [] l`, όπου  $l$  πρέπει να είναι μια l-value τύπου `^array of t`. Η τρέχουσα τιμή της  
42  $l$  πρέπει να είναι ένας δείκτης προς ένα δυναμικό αντικείμενο που έχει κατασκευαστεί με  
43 χρήση της εντολής `new`. Μετά την εκτέλεση της εντολής, η  $l$  περιέχει το μηδενικό δείκτη.

## 1 1.6 Προκαθορισμένα υποπρογράμματα

2 Η PCL υποστηρίζει ένα σύνολο προκαθορισμένων υποπρογραμμάτων (διαδικασίες και συναρτήσεις βιβλιοθήκης), τα οποία είναι ορατά σε κάθε δομική μονάδα, εκτός αν επισκιάζονται από μεταβλητές, παραμέτρους ή υποπρογράμματα με το ίδιο όνομα. Παρακάτω δίνονται οι επικεφαλίδες τους, όπως θα γράφονταν αν τα ορίζαμε σε ένα πρόγραμμα PCL, και εξηγείται η λειτουργία τους.

### 6 1.6.1 Είσοδος και έξοδος

```
7 procedure writeInteger (n : integer);  
8 procedure writeBoolean (b : boolean);  
9 procedure writeChar (c : char);  
10 procedure writeReal (r : real);  
11 procedure writeString (var s : array of char);
```

12 Οι διαδικασίες αυτές χρησιμοποιούνται για την εκτύπωση τιμών που ανήκουν στους βασικούς τύπους της PCL, καθώς και για την εκτύπωση συμβολοσειρών.

```
14 function readInteger () : integer;  
15 function readBoolean () : boolean;  
16 function readChar () : char;  
17 function readReal () : real;  
18 procedure readString (size : integer; var s : array of char);
```

19 Αντίστοιχα, τα παραπάνω υποπρογράμματα χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της PCL και για την εισαγωγή συμβολοσειρών. Η διαδικασία `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.

### 26 1.6.2 Μαθηματικές συναρτήσεις

```
27 function abs (n : integer) : integer;  
28 function fabs (r : real) : real;
```

29 Η απόλυτη τιμή ενός ακέραιου ή πραγματικού αριθμού.

```
30 function sqrt (r : real) : real;  
31 function sin (r : real) : real;  
32 function cos (r : real) : real;  
33 function tan (r : real) : real;  
34 function arctan (r : real) : real;  
35 function exp (r : real) : real;  
36 function ln (r : real) : real;  
37 function pi () : real;
```

38 Βασικές μαθηματικές συναρτήσεις: τετραγωνική ρίζα, τριγωνομετρικές συναρτήσεις, εκθετική συνάρτηση, φυσικός λογάριθμος, ο αριθμός  $\pi$ .

### 40 1.6.3 Συναρτήσεις μετατροπής

```
41 function trunc (r : real) : integer;  
42 function round (r : real) : integer;
```

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

```
4 function ord (c : char) : integer;
5 function chr (n : integer) : char;
```

6 Μετατρέπουν από ένα χαρακτήρα στον αντίστοιχο κωδικό ASCII και αντίστροφα.

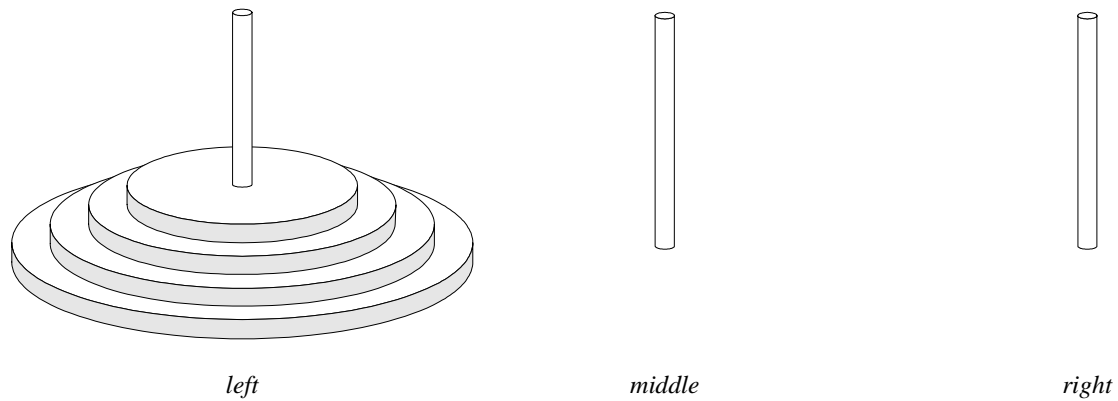
## 2 Πλήρης γραμματική της PCL

Η σύνταξη της γλώσσας PCL δίνεται παρακάτω σε μορφή EBNF. Η γραμματική που ακολουθεί είναι *διφορούμενη*, οι αμφισημίες όμως μπορούν να ξεπεραστούν αν λάβει κανείς υπόψη τους κανόνες προτεραιότητας και προσεταιριστικότητας των τελεστών, όπως περιγράφονται στην ενότητα 1.4.3. Τα σύμβολα  $\langle id \rangle$ ,  $\langle integer-const \rangle$ ,  $\langle real-const \rangle$ ,  $\langle char-const \rangle$  και  $\langle string-literal \rangle$  είναι τερματικά σύμβολα της γραμματικής.

```
1 <program> ::= "program" <id> ";" <body> "."
2 <body> ::= ( <local> )* <block>
3 <local> ::= "var" ( <id> ( "," <id> )* ":" <type> ";" )+ | "label" <id> ( "," <id> )* ";"
4 | <header> ";" <body> ";" | "forward" <header> ";"
5 <header> ::= "procedure" <id> "(" [ <formal> ( ";" <formal> )* ] ")"
6 | "function" <id> "(" [ <formal> ( ";" <formal> )* ] ")" ":" <type>
7 <formal> ::= [ "var" ] <id> ( "," <id> )* ":" <type>
8 <type> ::= "integer" | "real" | "boolean" | "char"
9 | "array" [ "[" <integer-const> "]" ] "of" <type> | "^" <type>
10 <block> ::= "begin" <stmt> ( ";" <stmt> )* "end"
11 <stmt> ::= ε | <l-value> "!=" <expr> | <block> | <call>
12 | "if" <expr> "then" <stmt> [ "else" <stmt> ] | "while" <expr> "do" <stmt>
13 | <id> ":" <stmt> | "goto" <id> | "return"
14 | "new" [ "[" <expr> "]" ] <l-value> | "dispose" [ "[" "]" ] <l-value>
15 <expr> ::= <l-value> | <r-value>
16 <l-value> ::= <id> | "result" | <string-literal> | <l-value> "[" <expr> "]"
17 | <expr> "^" | "(" <l-value> ")"
18 <r-value> ::= <integer-const> | "true" | "false" | <real-const> | <char-const>
19 | "(" <r-value> ")" | "nil" | <call> | "@" <l-value>
20 | <unop> <expr> | <expr> <binop> <expr>
21 <call> ::= <id> "(" [ <expr> ( "," <expr> )* ] ")"
22 <unop> ::= "not" | "+" | "-"
23 <binop> ::= "+" | "-" | "*" | "/" | "div" | "mod" | "or" | "and"
24 | "=" | "<>" | "<" | "<=" | ">" | ">="
25
```

## 3 Παραδείγματα

Στην ενότητα αυτή δίνονται πέντε παραδείγματα προγραμμάτων στη γλώσσα PCL.



Σχήμα 1: Οι πύργοι του Hanoi.

### 3.1 Πες γεια!

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

```

1  program hello;
2  begin
3      writeString("Hello world!\n")
4  end.
```

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

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

```

1  program solve;
2
3      var numberOfRings : integer;
4
5      procedure hanoi (var source, target, auxiliary : array of char; rings : integer);
6
7          procedure move (var source, target : array of char);
8              begin
9                  writeString("Move from ");
10                 writeString(source);
11                 writeString(" to ");
12                 writeString(target);
13                 writeString(".\n")
14             end;
```

```

15
16     begin
17         if rings >= 1 then
18             begin
19                 hanoi(source, auxiliary, target, rings-1);
20                 move(source, target);
21                 hanoi(auxiliary, target, source, rings-1)
22             end
23         end;
24
25     begin
26         writeString("Please, give the number of rings : ");
27         numberOfRings := readInteger();
28         writeString("\nHere is the solution :\n\n");
29         hanoi("left", "right", "middle", numberOfRings)
30     end.

```

### 3.3 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα PCL 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 τύπωσε τον  
    αν ο αριθμός  $t + 1$  είναι πρώτος τότε τύπωσε 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 PCL είναι το ακόλουθο.

```

1  program primes;
2
3      var limit, number, counter : integer;
4
5      function prime (n : integer) : boolean;
6          var i : integer;
7      begin
8          if n < 0 then
9              result := prime(-n)
10         else if n < 2 then
11             result := false
12         else if n = 2 then
13             result := true

```

```

14     else if n mod 2 = 0 then
15         result := false
16     else
17         begin
18             i := 3;
19             while i <= n div 2 do
20                 begin
21                     if n mod i = 0 then
22                         begin
23                             result := false;
24                             return
25                         end;
26                     i := i+2
27                 end
28             end;
29             result := true
30         end;
31
32     begin
33         writeString("Please, give the upper limit : ");
34         limit := readInteger();
35         writeString("Prime numbers between 0 and ");
36         writeInteger(limit);
37         writeString("\n\n");
38         counter := 0;
39         if limit >= 2 then
40             begin
41                 counter := counter + 1;
42                 writeString("2\n")
43             end;
44         if limit >= 3 then
45             begin
46                 counter := counter + 1;
47                 writeString("3\n");
48             end;
49         number := 6;
50         while number <= limit do
51             begin
52                 if prime(number-1) then
53                     begin
54                         counter := counter + 1;
55                         writeInteger(number-1);
56                         writeString("\n")
57                     end;
58                 if (number <> limit) and prime(number+1) then
59                     begin
60                         counter := counter + 1;
61                         writeInteger(number+1);
62                         writeString("\n");
63                     end;
64                 number := number + 6
65             end;
66         writeString("\n");
67         writeInteger(counter);
68         writeString(" prime number(s) were found.\n")
69     end.

```

### 3.4 Αντιστροφή συμβολοσειράς

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

```
1   program reverse;
2
3   function strlen (var s : array of char) : integer;
4   begin
5       result := 0;
6       while s[result] <> '\0' do result := result + 1
7   end;
8
9   var r : array [32] of char;
10
11  procedure reverse (var s : array of char);
12  var i, l : integer;
13  begin
14      l := strlen(s);
15      i := 0;
16      while i < l do
17          begin
18              r[i] := s[l-i-1];
19              i := i+1
20          end;
21      r[i] := '\0'
22  end;
23
24  begin
25      reverse("\n!dlrow olleH");
26      writeString(r)
27  end.
```

### 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 PCL 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 PCL ότι τα στοιχεία του είναι τα  $x[0], x[1], \dots, x[n-1]$ ), 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 στη γλώσσα PCL είναι το εξής:

```
1   program bsort;
2
3   procedure BubbleSort (var x : array of integer; n : integer);
4   var i : integer;
5       changed : boolean;
6
7   procedure swap (var x, y : integer);
```

```

8         var t : integer;
9     begin
10        t := x;
11        x := y;
12        y := t
13    end;
14
15    begin
16        changed := true;
17        while changed do
18            begin
19                i := 0;
20                changed := false;
21                while i < n-1 do
22                    begin
23                        if x[i] > x[i+1] then
24                            begin
25                                swap(x[i], x[i+1]);
26                                changed := true
27                            end;
28                            i := i+1
29                    end
30                end
31            end;
32
33        procedure PrintArray (var msg : array of char;
34                               var x  : array of integer;
35                               n    : integer);
36
37            var i : integer;
38            begin
39                writeString(msg);
40                i := 0;
41                while i < n do
42                    begin
43                        if i > 0 then
44                            writeString(", ");
45                            writeInteger(x[i]);
46                            i := i+1
47                        end;
48                        writeString("\n");
49                    end;
50
51            var i    : integer;
52                x    : array [16] of integer;
53                seed : integer;
54            begin
55                i := 0;
56                seed := 65;
57                while i < 16 do
58                    begin
59                        seed := (seed * 137 + 221 + i) mod 101;
60                        x[i] := seed;
61                        i := i + 1
62                    end;
63                PrintArray("Initial array: ", x, 16);
64                BubbleSort(x, 16);
65                PrintArray("Sorted array: ", x, 16)
66            end.

```

### 3.6 Μέση τιμή τυχαίας μεταβλητής

Το πρόγραμμα που ακολουθεί υπολογίζει τη μέση τιμή μιας αέραςιας 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   var n, k, i, seed : integer;
4       sum           : real;
5
6   begin
7       writeString("Give n: ");
8       n := readInteger();
9       writeString("Give k: ");
10      k := readInteger();
11
12      i := 0;
13      sum := 0.0;
14      seed := 65;
15      while i < k do
16          begin
17              seed := (seed * 137 + 221 + i) mod n;
18              sum := sum + seed;
19              i := i + 1
20          end;
21
22      if k > 0 then
23          begin
24              writeString("Mean: ");
25              writeReal(sum / k);
26              writeString("\n")
27          end
28      end.
```

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

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

Ο τελικός μεταγλωττιστής πρέπει να μπορεί να εξάγει κατά βούληση ενδιάμεσο και τελικό κώδικα. Εφόσον δεν έχουν καθορισθεί οι παράμετροι λειτουργίας `-f` ή `-i`, που εξηγούνται παρακάτω, ο μεταγλωττιστής θα δέχεται το πηγαίο πρόγραμμα από ένα αρχείο με οποιαδήποτε κατάληξη (πχ. `*.pc1`) 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.pc1` θα παράγονται τα `/tmp/hello.imm` και `/tmp/hello.asm`.

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

- o file το εκτελέσιμο που παράγεται θα αποθηκεύεται στο file. Αν η παράμετρος αυτή δεν έχει δοθεί το εκτελέσιμο θα πρέπει να βρίσκεται στο αρχείο ./a.out (δηλ. στο τρέχον directory από το οποίο καλείται ο μεταγλωττιστής σας).
- O σημαία βελτιστοποίησης (προαιρετική).
- f πρόγραμμα στο standard input, έξοδος τελικού κώδικα στο standard output.
- i πρόγραμμα στο standard input, έξοδος ενδιάμεσου κώδικα στο standard output.

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

Την ίδια τιμή επιστροφής (μηδέν) θα πρέπει να επιστρέφει και το παραγόμενο εκτελέσιμο για το πρόγραμμα που μεταγλωττίστηκε, αν η εκτέλεσή του είναι επιτυχής. Αυτό σημαίνει ότι η εκτέλεση από τον φλοιό του λειτουργικού της εντολής ./a.out && ./a.out όπου a.out είναι το εκτελέσιμο για το πρόγραμμα hello.rc1 θα πρέπει να τυπώνει το "Hello world!" δύο φορές (όχι μία μόνο).

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

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

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

```
CC=gcc
CFLAGS=-Wall

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

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

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

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

distclean: clean
    $(RM) pclc
```

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
```