# ΗΥ-252 Αντικειμενοστραφής Προγραμματισμός
# Προγραμματιστική Εργασία Χειμερινού Εξαμήνου 2004

## 1. Μηχανές Στοίβας

Ένα εναλλακτικό μοντέλο υπολογισμού σε αυτό των μηχανών καταχωρητών είναι οι μηχανές στοίβας. Στις μηχανές καταχωρητών, ως γνωστόν, η επεξεργασία των δεδομένων γίνεται αντιγράφοντας τα δεδομένα από την κύρια μνήμη σε κάποιον από τους καταχωρητές της μηχανής, εφαρμόζοντας σε αυτά κάποια λειτουργία (operation) και έπειτα αντιγράφοντας τα πάλι πίσω στην κύρια μνήμη όπου παραμένουν μέχρι να ξαναχρειαστούν. Μηχανές καταχωρητών είναι οι περισσότεροι από τους σύγχρονους υπολογιστές γενικού σκοπού.

Στις μηχανές στοίβας δεν υπάρχουν οι έννοιες της κύριας μνήμης και των καταχωρητών. Τα δεδομένα που χρησιμοποιεί ένα πρόγραμμα είναι τοποθετημένα σε μια στοίβα. Η επεξεργασία γίνεται αφαιρώντας δεδομένα από την κορυφή της στοίβας, εφαρμόζοντας πάνω τους κάποια λειτουργία και τοποθετώντας το αποτέλεσμα της λειτουργίας (αν αυτό υπάρχει) πάλι στην κορυφή της στοίβας. Αν οι μηχανές στοίβας με την πρώτη ματιά δείχνουν να έχουν μικρότερες υπολογιστικές δυνατότητες από τις μηχανές καταχωρητών, στην πραγματικότητα τα δύο μοντέλα είναι ισοδύναμα και μπορεί το ένα να προσομοιώσει το άλλο.

Οι βασικές λειτουργίες των μηχανών στοίβας είναι, όπως εύκολα μπορεί να φανταστεί κανείς, οι `push` και `pop`. Αυτές οι λειτουργίες όμως δεν είναι λειτουργίες υπολογισμού. Γι αυτό συνήθως υπάρχει και μια ποικιλία άλλων λειτουργιών που κάνουν την επεξεργασία των δεδομένων (για παράδειγμα, λειτουργίες αριθμητικών πράξεων). Στον πίνακα 1 παρουσιάζεται η λειτουργία ενός υπολογιστή στοίβας που μπορεί να κάνει αριθμητικές πράξεις καθώς υπολογίζει το αποτέλεσμα της παράστασης 5+(6 * (5 - 3)).

| Λειτουργία | Κατάσταση στοίβας |
|---|---|
| `init` | |
| `push 5` | 5 |
| `push 3` | 5 3 |
| `sub` | 2 |
| `push 6` | 2 6 |
| `mul` | 12 |
| `push 5` | 12 5 |
| `add` | 17 |

**Πίνακας 1 – Υπολογισμός του 5+(6 * (5 - 3))**

## 2. Η γλώσσα Postscript

Η Postscript είναι μια γλώσσα προγραμματισμού γενικού σκοπού. Παρόλα αυτά η συνήθης χρήση της είναι η περιγραφή εκτυπώσιμης σελίδας. Δηλαδή περιγράφουμε την σελίδα που θέλουμε να δούμε στην οθόνη μας ή να εκτυπώσουμε με ένα πρόγραμμα Postscript. Το πρόγραμμα αυτό το οποίο το δίνουμε σε κάποιο διερμηνέα Postscript (interpreter) ο οποίος παράγει την τελική σελίδα. Στην πρώτη περίπτωση (προβολή σε οθόνη) ο διερμηνέας Postscript είναι κάποιο πρόγραμμα (π.χ. ghostview) που τρέχει σε έναν υπολογιστή. Στη δεύτερη περίπτωση (εκτύπωση) ο διερμηνέας είναι υλοποιημένος σε hardware και ενσωματωμένος στον εκτυπωτή. Οι περισσότεροι laser εκτυπωτές σήμερα έχουν ενσωματωμένους Postscript interpreters.

Το μοντέλο υπολογισμού της Postscript είναι η μηχανή στοίβας. Οι λειτουργίες (operations), όπως μπορεί να φανταστεί κανείς, σε μεγάλο μέρος τους αφορούν τη διαχείριση γραφικών. Οι interpreters της είναι

μηχανές στοίβας που υλοποιούν τις λειτουργίες αυτές. Τα αρχεία Postscript είναι σε μορφή ascii text και περιέχουν τόσο δεδομένα όσο και τις λειτουργίες που θα εφαρμοστούν πάνω σε αυτά. Οι interpreters διαβάζουν από τα αρχεία Postscript δεδομένα και λειτουργίες και τροποποιούν την κατάσταση της στοίβας ανάλογα.

Όλα τα περιεχόμενα ενός αρχείου Postscript θεωρούνται σαν αντικείμενα Postscript. Ωστόσο τα αντικείμενα Postscript διακρίνονται σε literal αντικείμενα (π.χ. αριθμοί, strings, πίνακες κτλ) και εκτελέσιμα αντικείμενα – operators (π.χ. built-in τελεστές, συναρτήσεις κτλ). Τα αντικείμενα διαβάζονται ένα-ένα από το αρχείο Postscript. Το πρόγραμμα Postscript είναι ουσιαστικά μία ροή αντικειμένων. Αν είναι literals τοποθετούνται στη στοίβα εκτέλεσης. Αν είναι εκτελέσιμα, εκτελούνται, αλλάζοντας πιθανώς τα περιεχόμενα της στοίβας εκτέλεσης. Σημειώνουμε ότι η τοποθέτηση ενός αντικειμένου στην κορυφή της στοίβας εκτέλεσης (`push`) γίνεται «αυτόματα» και δε χρειάζεται να γίνει explicitly από τον προγραμματιστή.

Η Postscript «ζωγραφίζει» την σελίδα-αποτέλεσμα του κάθε προγράμματος χρησιμοποιώντας διαδρομές (paths). Μια διαδρομή Postscript είναι μια σειρά από γραμμές και καμπύλες που ορίζουν μια περιοχή για γέμισμα ή μια τροχιά προς «ζωγράφισμα». Κατά τη διάρκεια εκτέλεσης του προγράμματος η διαδρομή δημιουργείται, χωρίς όμως να ζωγραφίζεται πραγματικά. Η πραγματική εκτύπωση στην οθόνη ή στον εκτυπωτή γίνεται μόνο όταν το πρόγραμμα φτάσει σε κάποιο ειδικό τελεστή Postscript (π.χ. `stroke`,`show`), ο οποίος αναλαμβάνει την κατάλληλη εμφάνιση ενός αντικειμένου.

Στο επόμενο σχήμα φαίνεται ένα πρόγραμμα Postscript που ζωγραφίζει ένα σπιράλ. Τα σχόλια φαίνονται με μπλε γράμματα, οι operators με καφέ, και τα literals με ροζ. Παρατηρούμε πως εξαιτίας της δομής στοίβας που χρησιμοποιούν τα προγράμματα για να αποθηκεύσουν την κατάστασή τους, οι παράμετροι των operators εμφανίζονται πριν από τους ίδιους τους operators. Για παράδειγμα για το `for` loop, οι παράμετροι του, δηλαδή τα όρια του loop και το block εντολών του loop, εμφανίζονται πριν από το ίδιο το `for`, ενώ π.χ. στη C ή στη Java το `for` προηγείται των ορίων του loop και του block εντολών. Επίσης χαρακτηριστικό είναι πως κατά την εκτέλεσή τους πολλοί τελεστές «καταναλώνουν» literals από τη στοίβα. Έτσι πρέπει συχνά να χρησιμοποιούμε τον τελεστή `dup` (αντιγραφής) για να μην «χάσουμε» κάποιο literal της στοίβας το οποίο μας χρειάζεται και στη συνέχεια.

```
% spiral.ps
% prints a spiral
% operations tested:
%       for
%       if
%       moveto
%       rlineto

newpath

%move to center of spiral
202 202 moveto

%spiral drawing loop
%2 is the distance between two spires
%400 defines the number of spires
0 2 400{

        %compute the action selector (400/2) MOD 4
        %according to this number we choose which way to go
        %i is the length of the next line to be drawn
        dup %duplicate i
        2 idiv %do i/2
        4 mod %do (400/2) MOD 4

        %Draw the next line.
        %eq consumes the action selector, so we have to
```

```
        %duplicate the action selector before every eq.
        %
        %On the executable array the first thing we do
        %is to pop the duplicated action selector. The
        %action selector is pushed back again at the end
        %of the executable array.
        dup
        0 eq {pop 0 rlineto 0} if
        dup
        1 eq {pop 0 exch rlineto 1} if
        dup
        2 eq {pop -1 mul 0 rlineto 2} if
        dup
        3 eq {pop -1 mul 0 exch rlineto} if

} for


stroke
showpage
```

**Σχήμα 1 – Πρόγραμμα Postscript**

## 3. Ζητούμενα Προγραμματιστικής Εργασίας

Στην άσκηση αυτή θα πρέπει να υλοποιήσετε έναν interpreter για ένα μικρό υποσύνολο της γλώσσας Postscript. Το υποσύνολο αυτό θα επιτρέπει τον προγραμματισμό απλών σχεδίων που αποτελούνται από γραμμές, καμπύλες και κείμενο. Πιο συγκεκριμένα:

- Ο interpreter σας θα πρέπει να υποστηρίζει τα εκτελέσιμα αντικείμενα (λειτουργίες) που παρουσιάζονται στον πίνακα τελεστών. Από literals, ο interpreter σας θα πρέπει να υποστηρίζει αριθμητικά literals, λογικά literals, αλφαριθμητικά (string) literals και εκτελέσιμους πίνακες.
- Για κάθε τύπο Postscript Object που υποστηρίζει ο interpreter σας, θα υλοποιήσετε και μια ξεχωριστή κλάση. Οι κλάσεις αυτές θα πρέπει να ομαδοποιηθούν σύμφωνα με το τι είδους Postscript Object αντιπροσωπεύουν. Η ομαδοποίηση θα γίνει υλοποιώντας interfaces. Τα δύο κύρια interfaces, υπερκλάσεις των υπόλοιπων interfaces, θα αντιπροσωπεύουν προφανώς τους Postscript Operators και τα Postscript literals. Έστω `PSOperator` και `PSLiteral` τα ονόματα των δύο αυτών interfaces.

  Οι βασικές λειτουργίες που θα πρέπει να παρέχουν τα interfaces αυτά φαίνονται στον πίνακα 2.

```
public interface PSOperator{
    // returns the token identifying this operator
    public String getOperatorToken();

    // checks if it is valid to use this operator now
    public boolean isValid(PSStack s)
        throws PSSyntaxException;
}
public interface PSLiteral{
    // Make explicit that all literals should implement
    // toString(). This is needed to implement generic
    // stack printing methods.
    public String toString();
}
```

**Πίνακας 2 – Βασικά interfaces**

- Δεδομένου πως υποστηρίζονται μόνο 4 είδη literals (αριθμητικοί, λογικοί, αλφαριθμητικοί και εκτελέσιμοι πίνακες), δεν χρειάζεται να ορίσετε επιπλέον interfaces που να εξειδικεύουν το interfaces `PSLiteral`.
- Αντίθετα θα πρέπει να ορίσετε εξειδικεύσεις του interface `PSOperator`, για να ομαδοποιήσετε τους Postscript Operators σύμφωνα με την κατηγορία στην οποία έχουν καταταχτεί στον πίνακα τελεστών. Δηλαδή θα έχουμε τα interfaces `PSStackOperator`, `PSArithmeticOperator` κτλ, τα οποία θα επεκτείνουν το interface `PSOperator`. Η ομαδοποίηση αυτή των κλάσεων, θα σας επιτρέψει να γράψετε γενικές μεθόδους χειρισμού ομοειδών Postscript Operators. Κάθε νέο interface θα πρέπει να περιέχει μια μέθοδο action(), η οποία θα υλοποιεί τη λειτουργία του operator. Τα ορίσματα της μεθόδου θα είναι ανάλογα με τον τύπο του operator. Παραδείγματα ορισμού τριών εξειδικεύσεων του PSOperator φαίνονται στον πίνακα 3.

```
public interface PSStackOperator extends PSOperator{
    /* apply the stack operator on the stack */
    public void action(PSStack s);
}
public interface PSPathOperator extends PSOperator{
    /* apply the stack operator on the stack, and path */
    public void action(PSStack s, PSPath p);
}
public interface PSConsoleOperator extends PSOperator{
    /* apply the stack operator on the stack using os stream */
    public void action(PSStack s, OutputStream os);
}
```

**Πίνακας 3 – operator interfaces**

- Όσον αφορά τα literals που θα υλοποιήσετε, ιδιαιτερότητα παρουσιάζει η υλοποίηση των εκτελέσιμων πινάκων. Οι εκτελέσιμοι πίνακες είναι ο μηχανισμός με τον οποίο η Postscript ορίζει blocks αντικειμένων που θα πρέπει να χρησιμοποιούνται μαζί. Τα όρια ενός εκτελέσιμου πίνακα μέσα στο πρόγραμμα σημαδεύονται με τα '{' και '}'. Μια και οι εκτελέσιμοι πίνακες θεωρούνται και οι ίδιοι αντικείμενα, εκτός από άλλα Postscript αντικείμενα, μπορούν πιθανώς να περιέχουν και άλλους εκτελέσιμους πίνακες.

```
1 2 eq
{
    100 0
    100 200
}
{
    0 100
    200 100
}
ifelse
moveto
rlineto
stroke
```

**Πίνακας 4 – Χρήση εκτελέσιμων πινάκων**

Στον πίνακα 4 φαίνεται η χρήση των εκτελέσιμων πινάκων προκειμένου να επιλέξουμε ποια από τις δύο διαφορετικές γραμμές θα ζωγραφίσουμε, ανάλογα με το αποτέλεσμα μιας σύγκρισης. Αν δεν υπήρχαν οι εκτελέσιμοι πίνακες, δεν θα μπορούσαμε να ομαδοποιήσουμε τις δύο συνεχόμενες `moveto` και `rlineto`.

Η «εκτέλεση» των εκτελέσιμων πινάκων γίνεται είτε σαν αποτέλεσμα ενός τελεστή ροής, είτε σαν αποτέλεσμα του τελεστή exec, τον οποίο όμως είναι προαιρετικό να υλοποιήσετε. Για έναν εκτελέσιμο πίνακα που περιέχει ν Postscript Objects, το αποτέλεσμα της εκτέλεσής του είναι να διαβαστούν αυτά τα ν στοιχεία από τον πίνακα, προτού διαβαστεί το επόμενο Postscript Object από το αρχείο του προγράμματος και φυσικά να εκτελεστούν όσα από αυτά είναι operators. Για παράδειγμα, στον πίνακα 5 φαίνεται πώς εξελίσσεται η κατάσταση της στοίβας στο πρόγραμμα του πίνακα 4 καθώς και πληροφορίες για την προέλευση του κάθε Postscript Object.

| Action | PS Object | Κατάσταση στοίβας |
|---|---|---|
| Το αριθμητικό 1 εισάγεται στη στοίβα. | `1` | 1 |
| Το αριθμητικό 2 εισάγεται στη στοίβα. | `2` | 1 2 |
| Η `action()` της κλάσης που υλοποιεί την `eq` εκτελείται. Το λογικό literal `false` εισάγεται στη στοίβα. | `eq` | false |
| Μόλις διαβάζεται το '{', καλείται από τη μέθοδο που διαβάζει τα PS Objects ο constructor της κλάσης που υλοποιεί τον εκτελέσιμο πίνακα. Ο constructor διαβάζει συνεχώς νέα PS Objects μέχρι να συναντήσει το αντίστοιχο του '}'. Τελικά ο εκτελέσιμος πίνακας `EA1` επιστρέφεται και τοποθετείται στη στοίβα. | `{ 100 0 100 200 } (EA1)` | false EA1 |
| Όμοια με το προηγούμενο. | `{ 0 100 200 100 } (EA2)` | false EA1 EA2 |
| Η `action()` της `ifelse` διαβάζει τα 3 πρώτα objects της στοίβας. Αφού δει πως η συνθήκη είναι `false`, επιλέγει τον EA2 και τον δίνει στη μηχανή εκτέλεσης για να εκτελεστεί. | `ifelse` | 0 100 200 100 |
| Εκτέλεση της `moveto`. | `moveto` | 0 100 |
| Εκτέλεση της `rlineto`. | `rlineto` | |
| Εκτέλεση της `stroke`. | `stroke` | |

**Πίνακας 5 – Εκτέλεση του προγράμματος του πίνακα 4**

Σημειώνουμε πως η ίδια η `action()` της `ifelse` δεν μπορεί να εκτελέσει τον EA2. Αυτό, γιατί ο EA2 ίσως περιέχει operators που απαιτούν παραμέτρους που δεν είναι διαθέσιμες, π.χ. περιέχει κάποιον path operator, οπότε δεν είναι «γνωστό» στην `action()` της `ifelse` ποιο path να χρησιμοποιήσει σαν παράμετρο στην `action()` του path operator.

Οι εκτελέσιμοι πίνακες μπορούν να υλοποιηθούν πολύ εύκολα με τη χρήση των πινάκων της Java ή κάποιου Java container. Η διεπαφή της κλάσης των εκτελέσιμων πινάκων, θα πρέπει να επιτρέπει την ανάκτηση ενός προς ενός των περιεχομένων τους.

- Για τα αριθμητικά literals, θα πρέπει να υποστηρίζονται και ακέραιοι και πραγματικοί αριθμοί. Ωστόσο, σημειώστε ότι η PostScript δεν θεωρεί τους ακέραιους αριθμούς σαν διαφορετικό τύπο δεδομένων από τους πραγματικούς. Αναφέρεται και στα δύο ως number.

- Τα αλφαριθμητικά (strings) literals εμφανίζονται εγκλεισμένα σε παρενθέσεις. Ένα αλφαριθμητικό μπορεί να έχει οσοδήποτε μεγάλο μέγεθος αρχίζοντας με ( και τελειώνοντας με ) και μπορεί να περιέχει οποιοδήποτε χαρακτήρα εκτός από τους (, ) και \. Στην περίπτωση που χρειαζόμαστε κάποιον από τους προηγούμενους χαρακτήρες ως κανονικό κείμενο τότε πρέπει να χρησιμοποιήσουμε τον χαρακτήρα διαφυγής \. Δηλαδή οι \( \) και \\ εμφανίζουν αντίστοιχα την αριστερή παρένθεση, την δεξιά παρένθεση και την κάθετο. Επιπλέον, πρέπει να υποστηρίζονται οι χαρακτήρες διαφυγής line-feed \n και horizontal tab \t. Μερικά παραδείγματα νόμιμων αλφαριθμητικών και πως εμφανίζονται αυτά δίδονται παρακάτω:

  ```
  (This is a string)
  ```
  This is a string

  ```
  (Strings may contain newlines \n and such.)
  ```
  Strings may contain newlines
  and such.

  ```
  (Strings may have \( and \).)
  ```
  Strings may have ( and ).

  ```
  (Today the date is 12\\12\\2012)
  ```
  Today the date is 12\12\2012
- Αν κάποιες λειτουργίες δεν μπορούν να εκτελεστούν για τους τύπους αντικειμένων που βρίσκονται στην κορυφή της στοίβας εκτέλεσης, θα πρέπει να δημιουργούνται οι κατάλληλες runtime exceptions.
- Ο interpreter θα πρέπει να υποστηρίζει σχόλια στα προγράμματα. Τα σχόλια στην Postscript ξεκινούν με '%' και τελειώνουν στο τέλος της γραμμής του προγράμματος.
- Ο interpreter θα πρέπει να χειρίζεται τα γραφικά με τη χρήση διαδρομών (paths). Γενικά μια διαδρομή Postscript είναι μια σειρά από γραμμές και καμπύλες που ορίζουν μια περιοχή για γέμισμα ή μια τροχιά προς «ζωγράφισμα». Στο υποσύνολο της Postscript, το οποίο θα αναγνωρίζει ο interpreter σας, οι διαδρομές από γραμμές και τόξα είναι μόνο προς «ζωγράφισμα».

  Οι γραμμές που αποτελούν κάθε path θα πρέπει να αποθηκεύονται σε κατάλληλη δομή, ώστε να είναι διαθέσιμες για να ζωγραφιστούν όταν εμφανιστεί ο operator `stroke` ή `showpage`. Ειδικά για την τρέχουσα διαδρομή θα πρέπει επιπλέον να αποθηκεύεται και το σημείο όπου βρίσκεται η γραφίδα, καθώς και το τελευταίο σημείο προς το οποίο έγινε μια `moveto` ή `rmoveto`. Επίσης οι τελεστές `stroke` πρέπει να αποθηκεύονται για κάθε path, για να μη ζωγραφίζονται γραμμές για τις οποίες δεν έχει δοθεί ακόμα εντολή `stroke`.

  Στον πίνακα 6 φαίνεται η δημιουργία ενός path βήμα-βήμα. Μετά το `stroke` η γραμμή εμφανίζεται στο παράθυρο γραφικών.

| PS Objects | Path Info |
|---|---|
| `100 100 moveto` | `curp(100,100), lastmove(100,100), contents=()` |
| `100 100 rlineto` | `curp(200,200), lastmove(100,100),`<br>`contents=line(100,100,200,200)` |
| `0 1000 lineto` | `curp(0,1000), lastmove(100,100),`<br>`contents=line(100,100,200,200), line(200,200,0,1000)` |
| `stroke` | `curp(0,1000), lastmove(100,100),`<br>`contents=line(100,100,200,200), line(200,200,0,1000), stroke` |

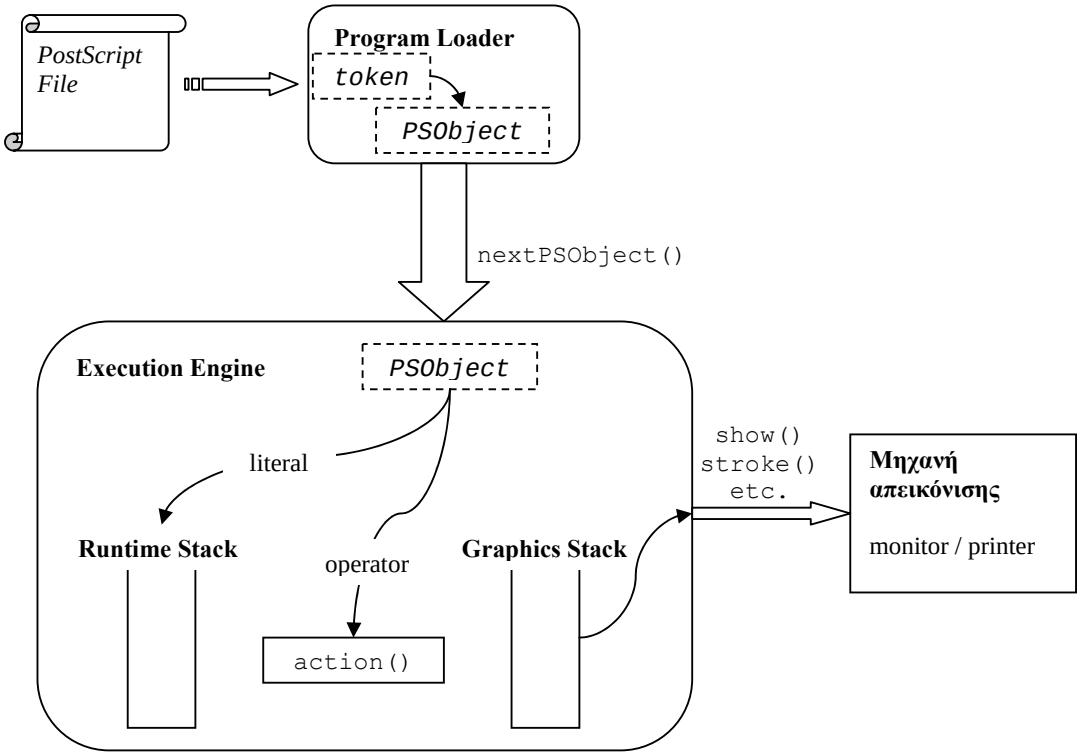
**Πίνακας 6 – Εκτέλεση του προγράμματος του πίνακα 4**

- Ο τελικός interpreter σας θα πρέπει να μπορεί να δέχεται σαν παράμετρο το αρχείο που περιέχει το πρόγραμμα. Αν αυτό δεν συμβαίνει, θα πρέπει αυτόματα να διαβάζει τις εντολές από το System.in.

Συνήθεις postscript interpreters όπου μπορείτε να ελέγξετε απλά προγράμματα πριν τα δοκιμάσετε στον interpreter σας, είναι ο gs (σε περιβάλλον X11-unix) και ο ghostscript σε περιβάλλον ms-windows.

## Βοήθεια υλοποίησης του διερμηνέα PostScript

Εκτός από τις κλάσεις που υλοποιούν τις διάφορες λειτουργίες, θα πρέπει να υλοποιήσετε και το σύστημα εκτέλεσης (runtime system) του διερμηνέα σας. Το σύστημα εκτέλεσης μπορεί να χωριστεί σε 5 κομμάτια.




**Σχήμα 2 – Σχηματική απεικόνιση συστήματος εκτέλεσης διερμηνέα PostScript.**

**a) Program Loader**

Ο program loader είναι υπεύθυνος να διαβάζει ascii tokens από το αρχείο εισόδου (πρόγραμμα postscript) και να τα μετατρέπει σε runtime objects. Επίσης πρέπει να αναγνωρίζει τα σχόλια του προγράμματος και να τα απορρίπτει. Αυτό μπορεί να γίνει εύκολα με την χρήση της κλάσης java.io.StreamTokenizer που παρέχει μεθόδους για την ένα προς ένα επιστροφή των tokens, την αναγνώριση αυτών (εάν δηλαδή πρόκειται για numbers ή strings κλπ), και τον ορισμό χαρακτήρων που δηλώνουν την αρχή σχολίων.

Στον program loader θα πρέπει να προσφέρεται η δυνατότητα να μπορούμε να προσθέτουμε νέους τελεστές στο υποσύνολο της Postscript που υποστηρίζει ο interpreter χωρίς να αλλάζουμε κάθε φορά τον κώδικά του. Αυτό μπορεί να γίνει εύκολα με τη χρήση του μηχανισμού reflection της Java. Με την χρήση του μηχανισμού αυτού ο κώδικας γίνεται πιο συμπαγής και κατανοητός, διότι αποφεύγεται η υλοποίηση με πολλαπλά if - else if, όταν θέλουμε να αναγνωρίσουμε, ποια κλάση θα πρέπει να εκτελέσουμε για κάθε τελεστή. Ο program loader κατά την αρχικοποίηση του θα διαβάζει ένα αρχείο με τα ονόματα των διαθέσιμων κλάσεων που αντιστοιχούν σε τελεστές. Έπειτα ο loader θα ελέγχει αν η κλάση με το συγκεκριμένο όνομα υπάρχει και αν αντιστοιχεί πραγματικά σε τελεστή (δηλαδή υλοποιεί τα ανάλογα interfaces). Αν ναι, τότε καλεί τη μέθοδο `getOperatorToken()` για να μάθει τη λέξη, την οποία, όταν θα την συναντά, θα πρέπει να επιστρέφει αντικείμενο της εν λόγω κλάσης.

**b) Execution Engine**

Η μηχανή εκτέλεσης ζητάει από τον program loader το επόμενο Postscript Object με τη χρήση της μεθόδου `nextPSObject()`. Ανάλογα με τον τύπο του Postscript Object, είτε το εισάγει στη

runtime stack, είτε καλεί τη μέθοδο, η οποία εκτελεί τη λειτουργία που αντιπροσωπεύει η κλάση. Η μηχανή εκτέλεσης πρέπει να παρέχει μια μέθοδο `execExecutableArray()`, η οποία θα μπορεί να καλείται από την action() των operators που χρησιμοποιούν εκτελέσιμους πίνακες. Η μέθοδος αυτή θα πρέπει να συμπεριφέρεται ακριβώς όπως αν τα περιεχόμενα του πίνακα προέρχονταν από την `nextPSObject()` και όχι από τον εκτελέσιμο πίνακα.

**c) Runtime Stack**

Η στοίβα εκτέλεσης περιέχει την κατάσταση του προγράμματος καθώς εκτελείται. Η κατάσταση αποτελείται από literals και εκτελέσιμους πίνακες. Η στοίβα εκτέλεσης πρέπει να παρέχει τις βασικές λειτουργίες `push()`, `pop()` και `peek(n)`. Οι πρώτες δύο είναι οι κλασσικές λειτουργίες στοίβας. Η τρίτη επιστρέφει τον τύπο του αντικειμένου που βρίσκεται στη θέση `n` από την κορυφή της στοίβας σαν ένα αντικείμενο τύπου `Class`. Με τη χρήση της `peek(n)` διευκολύνεται ο έλεγχος για το αν μπορεί κάποιος operator να εφαρμοστεί στην στοίβα όπως αυτή είναι τώρα.

**d) Graphics Stack**

Η στοίβα γραφικών κρατάει πληροφορίες για τις ολοκληρωμένες αλλά και την τρέχουσα διαδρομή γραφικών. Με βάση αυτήν την πληροφορία ζωγραφίζεται το τελικό αποτέλεσμα όταν συναντήσει η μηχανή εκτέλεσης τον ανάλογο τελεστή. Φυσικά μπορούν να χρησιμοποιηθούν και άλλες δομές εκτός της στοίβας για να κρατηθεί αυτή η πληροφορία.

**e) Μηχανή απεικόνισης**

Η μηχανή απεικόνισης εμφανίζει (στην περίπτωσή μας στην οθόνη) το γραφικό αποτέλεσμα του προγράμματος. Για την υλοποίησή του, χρησιμοποιήστε τις κλάσεις των πακέτων java.awt και javax.swing. Οι κλάσεις αυτές είναι επαρκείς για να καλύψουν τα απλά γραφικά που θα παράγει ο interpreter σας.

Σημειώνουμε πως 5 κομμάτια δε σημαίνει απαραίτητα και 5 κλάσεις.

| | Όνομα | Αλλαγές στην κορυφή της στοίβας | Περιγραφή |
|---|---|---|---|
| Τελεστές στοίβας | clear | ob1 … obn ⇒ - | Καθαρισμός στοίβας |
| | dup | ob ⇒ ob ob | «Κλωνοποίηση» |
| | index | obn … ob1 i ⇒ obn … ob1 obn | Αν i=n, αντιγράφει το n-ιοστό στοιχείο της στοίβας στην κορυφή της. |
| | exch | ob1 ob2 ⇒ ob2 ob1 | Αντιμετάθεση |
| | pop | ob1 ob2 ⇒ ob1 | Διαγραφή |
| Αριθμητικοί τελεστές | add | n1 n2 ⇒ n1+n2 | Πρόσθεση |
| | sub | n1 n2 ⇒ n1-n2 | Αφαίρεση |
| | div | n1 n2 ⇒ n1/n2 | Διαίρεση |
| | idiv | n1 n2 ⇒ (int)(n1/n2) | Ακέραια Διαίρεση |
| | mul | n1 n2 ⇒ n1*n2 | Πολλαπλασιασμός |
| | mod | n1 n2 ⇒ (n1 MOD n2) | Modulo |
| | abs | n1 ⇒ \|n1\| | Απόλυτο |
| Τελεστές κονσόλας | == | ob ⇒ – | Διαγραφή & εκτύπωση στην κονσόλα. |
| | pstack | ob1 obn ⇒ ob1 obn | Εκτύπωση όλης της στοίβας. |
| Τελεστές χειρισμού διαδρομής | newpath | - ⇒ – | Δημιουργία νέας διαδρομής. |
| | closepath | - ⇒ – | Κλείσιμο της τρέχουσας διαδρομής. Ισοδυναμεί με lineto προς το τελευταίο σημείο όπου έγινε moveto. |
| | stroke | - ⇒ – | Ζωγράφισμα της τρέχουσας διαδρομής. |
| | showpage | - ⇒ – | Ξαναζωγραφίζει όλες τις διαδρομές. |
| | lineto | x y ⇒ – | Γραμμή προς το σημείο x y και μετακίνηση προς αυτό. |
| | arc | x y r ang1 ang2 ⇒ - | Διαδρομή σχήματος τόξου κύκλου. Το τόξο έχει κέντρο το σημείο x y, ακτίνα r, ang1 είναι η γωνία από όπου αρχίζει το τόξο και ang2 η γωνία που καταλήγει, κινούμενο αντίθετα της φοράς των δεικτών του ρολογιού. Οι γωνίες υπολογίζονται με αρχή τον άξονα x. Εάν υπάρχει τρέχον σημείο στην διαδρομή, ο τελεστής arc περιλαμβάνει και μία ευθεία από το τρέχον σημείο προς την αρχή του τόξου. |
| | arct | x y r ang1 ang2 ⇒ - | Το ίδιο ισχύει με τον τελεστή arc, με την διαφορά ότι η φορά είναι ίδια με αυτήν των δειχτών του ρολογιού. |
| | moveto | x y ⇒ – | Μετακίνηση στο σημείο x y. |
| | rlineto | x y ⇒ – | Γραμμή προς το σημείο με συντεταγμένες x y ως προς το τρέχον σημείο και μετακίνηση προς αυτό. |
| | rmoveto | x y ⇒ – | Μετακίνηση στο σημείο με συντεταγμένες x y ως προς το τρέχον σημείο. |
| | currentpoint | ob ⇒ ob x y | Τοποθέτηση του τρέχοντος σημείου στην στοίβα. |

| | | | |
|---|---|---|---|
| Τελεστές σύγκρισης | eq | ob1 ob2 ⇒ bool | Ισότητα |
| | ne | ob1 ob2 ⇒ bool | μη ισότητα |
| | gt | ob1 ob2 ⇒ bool | Μεγαλύτερο |
| | ge | ob1 ob2 ⇒ bool | μεγαλύτερο-ίσο |
| | lt | ob1 ob2 ⇒ bool | Μικρότερο |
| | le | ob1 ob2 ⇒ bool | μικρότερο-ίσο |
| Τελεστές χειρισμού ροής | if | bool proc ⇒ – | if (bool) proc; |
| | ifelse | bool proct procf ⇒ – | if (bool) proct; else procf; |
| | for | j k l proc ⇒ ??? | for( i=j; i<=l; i+=k) proc;<br>Σε κάθε επανάληψη το i τοποθετείται στην κορυφή της στοίβας. |
| | loop | proc ⇒ – | Συνεχής επανάληψη του proc. |
| | exit | - ⇒ – | Τερματισμός του κοντινότερου loop ή for. |
| Τελεστές χειρισμού αλφαριθμητικών | string | n ⇒ string | Δημιουργία ενός νέου string μήκους n |
| | length | string ⇒ int | Επιστροφή του μήκους του αλφαριθμητικού |
| | get | string index ⇒ n | Επιστροφή του χαρακτήρα , στην ascii αναπαράστασή του, που βρίσκεται στην θέση index του αλφαριθμητικού str |
| | show | string ⇒ – | Εμφάνιση στην μηχανή απεικόνισης του string στην θέση x y. Η θέση πρέπει να έχει οριστεί προηγούμενα, π.χ. με την εντολή moveto. |

**Πίνακας 7 – Τελεστές / λειτουργίες προς υλοποίηση**

## 4. Σχεδιασμός και Υλοποίηση

Η εργασία χωρίζεται σε 2 φάσεις:

**Φάση 1η - Σχεδιασμός**

Σε αυτή τη φάση πρέπει να γίνει ο σχεδιασμός της εφαρμογής βάσει των ιδεών και των αρχών του αντικειμενοστραφούς προγραμματισμού που έχετε διδαχθεί. Αποτέλεσμα αυτής της φάσης είναι ο καθορισμός των αντικείμενων, των χαρακτηριστικών και της συμπεριφοράς τους που απαιτούνται για να αναπαραστήσουν τις τιμές (literals), τις λειτουργίες (βλέπε πίνακα 7) και τις συνιστώσες (βλέπε σχήμα 2) ενός διερμηνέα PostScript όπως έχουν περιγραφεί στις προηγούμενες σελίδες.

Παραδοτέα σε αυτή τη φάση είναι :

- μια αναφορά η οποία θα περιγράφει τα παραπάνω στοιχεία και θα παρουσιάζει το σχέδιο υλοποίησης του project έτσι ώστε να είναι έτοιμο το πέρασμα στην επόμενη φάση της υλοποίησης.
- τα Java interfaces και classes (πηγαίο κώδικα) συνοδευόμενα με τα απαραίτητα javadoc σχόλια, τα οποία θα καθοδηγήσουν την υλοποίηση στην επόμενη φάση.

Σε αυτή την φάση θα πρέπει να καθοριστεί επ'ακριβώς η λειτουργικότητα ενός PostScript διερμηνέα και να σκιαγραφηθεί σε επίπεδο classes και interfaces το πώς μπορεί να υλοποιηθεί. Επιγραμματικά, οι σημαντικότερες εργασίες που πρέπει να γίνουν σε αυτή τη φάση είναι:

- Αναγνώριση των κλάσεων και διεπαφών για κάθε λειτουργία και συνιστώσα του διερμηνέα. Αναγνώριση των ευθυνών κάθε κλάσης και των πιθανών σχέσεών της με άλλες. Ιδιαίτερα για τα PostScript Objects βρείτε την ιεραρχική δομή των κλάσεων (inheritance).

- Εύρεση της συμπεριφοράς (behaviour) κάθε κλάσης και διεπαφής του διερμηνέα, καθώς και της επικοινωνίας μέσω μηνυμάτων (method calls) που χρειάζεται να έχουν μεταξύ τους.
- Εύρεση των χαρακτηριστικών και των μεθόδων κάθε κλάσης.
- Για κάθε κλάση και διεπαφή δώστε τις υπογραφές (signatures) για όλες τις μεθόδους, το είδος τους (constructors, accessors, transformers) και τις εκ των προτέρων και εκ των υστέρων συνθήκες (preconditions, postconditions, invariants) που τις διέπουν.

**Φάση 2η - Υλοποίηση**

Σε αυτή τη φάση πρέπει να γίνει η υλοποίηση της εφαρμογής, βάσει της σχεδίασης που έχει προηγηθεί (φάση 1). Μολονότι δεν επιβάλλεται να χρησιμοποιηθεί αυτούσια η σχεδίαση της 1ης φάσης, καθότι κάποιες σχεδιαστικές επιλογές αποδεικνύονται στην πορεία άκυρες και χρειάζονται αναθεώρηση, εντούτοις η τελική βαθμολογία θα εξαρτηθεί από την συνέπεια της τελικής υλοποίησης ως προς την αρχική σχεδίαση.

Σε αυτή τη φάση, παραδοτέα είναι :

- ο πηγαίος κώδικας που υλοποιεί τον διερμηνέα PostScript και η δυνατότητα εκτέλεσης του διερμηνέα ως εφαρμογή Java.
- αναλυτικές οδηγίες για το πώς μεταγλωττίζεται και πώς τρέχει το πρόγραμμα (README, Makefile κ.λ.π.).
- αναφορά, στην οποία θα αναλύεται :
    - η -τελική- σχεδίαση της εφαρμογής,
    - ποιες αλλαγές έγιναν σε σχέση με τη σχεδίαση της 1ης φάσης (και γιατί),
    - οι αλγόριθμοι που χρησιμοποιήθηκαν,
    - τα προβλήματα που αντιμετωπίστηκαν,
    - οι σχεδιαστικές ή προγραμματιστικές αποφάσεις που ελήφθησαν και πώς αυτό ανταναικλάται στον τελικό χρήστη (π.χ. ευκολία/δυσκολία χειρισμού),
    - ... γενικά ό,τι άλλο κρίνετε απαραίτητο να αναφερθεί.

**Βαθμολογία εργασίας**

Για την βαθμολογία της εργασίας θα συνεκτιμηθούν

- εάν (και πόσο) η σχεδίαση της εφαρμογής εφαρμόζει τις έννοιες και τεχνικές του οντοκεντρικού προγραμματισμού που διδάχθηκαν στο μάθημα
- εάν (και πόσο) υλοποιήθηκαν οι ζητούμενες λειτουργίες της εφαρμογής
- η πληρότητα της τελικής αναφοράς, η οποία θα καταγράφει και θα τεκμηριώνει την σχεδίαση και υλοποίηση της εφαρμογής

*Παρατήρηση: Για την διευκόλυνση της εργασίας σας συνίσταται η σχεδίαση και υλοποίηση του διερμηνέα PostScript βήμα προς βήμα: μπορείτε σε μία πρώτη φάση να δείτε πως μπορείτε να αναπαραστήσετε μία λειτουργεία (operator) για κάθε κατηγορία, έπειτα να σχεδιάστε/υλοποιήστε μια-μια τις συνιστώσες του διερμηνέα και τέλος να ενώστε όλες τις λειτουργείες και όλες τις συνιστώσες σε μία ολοκληρωμένη εφαρμογή.*

Για διευκρινήσεις σχετικά με την παραπάνω εργασία, μπορείτε να στέλνετε ηλεκτρονικά μηνύματα με απορίες στη λίστα του μαθήματος *hy252-list*.