Αρχές Γλωσσών Προγραμματισμου

Χρήστος Νομικός

Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Ιωαννίνων

2015

Περιεχόμενα

1. Διάρκεια Ζωής και Διαχείριση Μνήμης
2. Εμβέλεια
3. Παραστάσεις

Περιεχόμενα

1. Διάρκεια Ζωής και Διαχείριση Μνήμης
2. Εμβέλεια
3. Παραστάσεις

Διάρκεια Ζωής και Διαχείριση Μνήμης

Ονομάζουμε μπλοκ μία ενότητα του προγράμματος, η οποία μπορεί να περιέχει δηλώσεις μεταβλητών (και γενικότερα ονομάτων). Για παράδειγμα ένα υποπρόγραμμα αποτελεί μπλόκ.

Διάρκεια ζωής ενός αντικειμένου ονομάζεται το χρονικό διάστημα κατά την εκτέλεση του προγράμματος κατά το οποίο το αντικείμενο είναι αποθηκευμένο σε κάποιες θέσεις μνήμης που χρησιμοποιούνται από το πρόγραμμα.

Όταν μία μεταβλητή συνδέεται με ένα αντικείμενο, ενδέχεται η διάρκεια ζωής της σύνδεσης μεταβλητής - αντικειμένου να διαφέρει από την διάρκεια ζωής του ίδιου του αντικειμένου.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Η διάρκεια ζωής ενός αντικειμένου γενικά διαφέρει από τη διάρκεια εκτέλεσης του προγράμματος. Αυτό είναι χρήσιμο για αρκετούς λόγους:

- Ενδέχεται το σύνολο των αντικειμένων που θα πρέπει να αποθηκευτούν στη μνήμη κατα τη διάρκεια της εκτέλεσης του προγράμματος να απαιτεί χώρο που υπερβαίνει το μέγεθος της διαθέσιμης μνήμης.
- Ενδέχεται να πρέπει να κατασκευαστεί μία λίστα από αντικείμενα που το πλήθος τους δεν είναι γνωστό εκ των προτέρων.
- Ένα αναδρομικό υποπρόγραμμα δημιουργεί αντικείμενα σε κάθε κλήση του και το βάθος της αναδρομής καθορίζεται κατά τη διάρκεια της εκτέλεσης του προγράμματος.
- ...

Διάρκεια Ζωής και Διαχείριση Μνήμης

Μπορούμε να διακρίνουμε τις παρακάτω τρεις κατηγορίες αντικειμένων, ανάλογα με τη διάρκεια ζωής τους. Τα αντικείμενα κάθε κατηγορίας αποθηκεύονται σε ένα ξεχωριστό τμήμα της μνήμης:

- Στατικά αντικείμενα: έχουν διάρκεια ζωής ίση με ολόκληρο το χρόνο εκτέλεσης του προγράμματος. Η αναφορά τους μπορεί να καθοριστεί στατικά (σε χρόνο φόρτωσης).
- Αντικείμενα στοίβας: έχουν ως διάρκεια ζωής τη διάρκεια εκτέλεσης ενός υποπρογράμματος (η γενικότερα ενός μπλοκ του προγράμματος).
  Τα αντικείμενα αυτά αποθηκεύονται σε ένα τμήμα της μνήμης που ονομάζεται στοίβα (η αποθήκευση αυτών των αντικειμένων μπορεί να γίνει στατικά μόνο σε γλώσσες που δεν υποστηρίζουν αναδρομή).

Διάρκεια Ζωής και Διαχείριση Μνήμης

- Αντικείμενα σωρού: δημιουργούνται και καταστρέφονται σε αυθαίρετες χρονικές στιγμές. Τα αντικείμενα αυτά:
  - δημιουργούνται με κατάλληλη εντολή του προγράμματος και η διάρκεια ζωής τους ενδέχεται να υπερβαίνει τη διάρκεια εκτέλεσης του μπλόκ κατά την εκτέλεση του οποίου δημιουργούνται.
  - καταστρέφονται με έναν από τους παρακάτω τρόπους:
    - με κατάλληλη εντολή του προγράμματος
    - αυτόματα κατά τη συλλογή σκουπιδιών, αν διαπιστωθεί ότι δεν υπάρχει πρόσβαση από το πρόγραμμα προς αυτά
    - όταν ολοκληρωθεί το πρόγραμμα.
  - αποθηκεύονται σε ένα τμήμα της μνήμης που ονομάζεται σωρός.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Παράδειγμα

Εστω το παρακάτω πρόγραμμα Pascal:

```
program p;
var q:^integer;

    procedure create(n:integer);
       begin;
          new(q); q^:=n;
       end;

    procedure delete;
       begin;
          dispose(q)
       end;

begin (* of program p *)
   create(5);
   delete;
end.
```

Διάρκεια Ζωής και Διαχείριση Μνήμης

- Το αντικείμενο τύπου `^integer` (δείκτης σε ακέραιο) που συνδέεται με τη μεταβλητή q είναι στατικό αντικείμενο. Η διάρκεια ζωής του ταυτίζεται με τη διάρκεια εκτέλεσης του προγράμματος.
- Το αντικείμενο τύπου integer που συνδέεται με την τυπική παράμετρο n του υποπρογράμματος create είναι αντικείμενο στοίβας. Η διάρκεια ζωής του ταυτίζεται με τη διάρκεια εκτέλεσης του υποπρογράμματος create.
- Το αντικείμενο που κατασκευάζεται στο υποπρόγραμμα create με την χρηση του new είναι αντικείμενο σωρού. Το αντικείμενο αυτό δεν έχει δικό του όνομα. Είναι ωστόσο προσπελάσιμο μέσω της μεταβλητής q η τιμή της οποίας είναι η θέση μνήμης στην οποία είναι αποθηκευμένο αυτό το αντικείμενο.
  Το αντικείμενο αυτό εξακολουθεί να υπάρχει και μετά την ολοκλήρωση του υποπρογράμματος create και καταστρέφεται κατά τη διάρκεια εκτέλεσης της delete με χρήση του dispose.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Τα στατικά αντικείμενα αποθηκεύονται σε ένα ξεχωριστό τμήμα της μνήμης και οι αναφορές τους μπορούν να καθοριστούν κατά τη φόρτωση του προγράμματος. Η μνήμη στην οποία αποθηκεύονται μπορεί να βρίσκεται στην αρχή της στοίβας, χωρίς αυτό να είναι απαραίτητο.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Η στοίβα χρησιμοποιείται για αντικείμενα που δημιουργούνται κατά την αρχή της εκτέλεσης ενός μπλόκ και καταστρέφονται κατά την ολοκλήρωσή του. Η δέσμευση μνήμης για αυτά τα αντικείμενα γίνεται δυναμικά (σε χρόνο εκτέλεσης).

Παρατηρούμε ότι οι κλήσεις των υποπρογραμμάτων ακολουθούν την λογική LIFO (last in first out): το πρώτο υποπρόγραμμα που καλείται είναι το τελευταίο που ολοκληρώνεται.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Η διαχείριση της στοίβας γίνεται με τον παρακάτω τρόπο:

- Αρχικά η στοίβα είναι κενή (ο χώρος στη μνήμη που διατίθεται για την αποθήκευση της στοίβας είναι αχρησιμοποίητος)
- Κάθε φορά που καλείται ένα υποπρόγραμμα δημιουργείται στην κορυφή της στοίβας ένα εγγράφημα δραστηριοποίησης, στο οποίο αποθηκεύονται (μαζί με άλλες πληροφορίες που σχετίζονται με το υποπρόγραμμα) τα αντικείμενα στοίβας που δημιουργούνται με την κλήση του υποπρογράμματος.
  Παρότι η θέση μνήμης ενός αντικειμένου στοίβας δεν μπορεί να προβλεφθεί στατικά, η σχετική θέση του μέσα στο εγγράφημα δραστηριοποίησης μπορεί να καθοριστεί στατικά.

Διάρκεια Ζωής και Διαχείριση Μνήμης

- Όταν ολοκληρώνεται η εκτέλεση ένος υποπρογράμματος, το εγγράφημα δραστηριοποίησης του διαγράφεται από τη στοίβα και οι αντίστοιχες θέσεις μνήμης ελευθερώνονται.
  Κατ΄ αυτό τον τρόπο τα αντικείμενα που είχαν δημιουργηθεί κατά την κλήση του καταστρέφονται.
  Η στοίβα συρρικνώνεται στο μέγεθος που είχε ακριβώς πριν την κλήση.

Παρατηρούμε ότι στο τμήμα της μνήμης που διατίθεται για αποθήκευση των αντικειμένων στοίβας, η στοίβα όπως επίσης και ο αχρησιμοποίητος χώρος αποτελούνται απο συνεχόμενες θέσεις μνήμης.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Παράδειγμα

Έστω ο παρακάτω σκελετός προγράμματος σε Pascal:

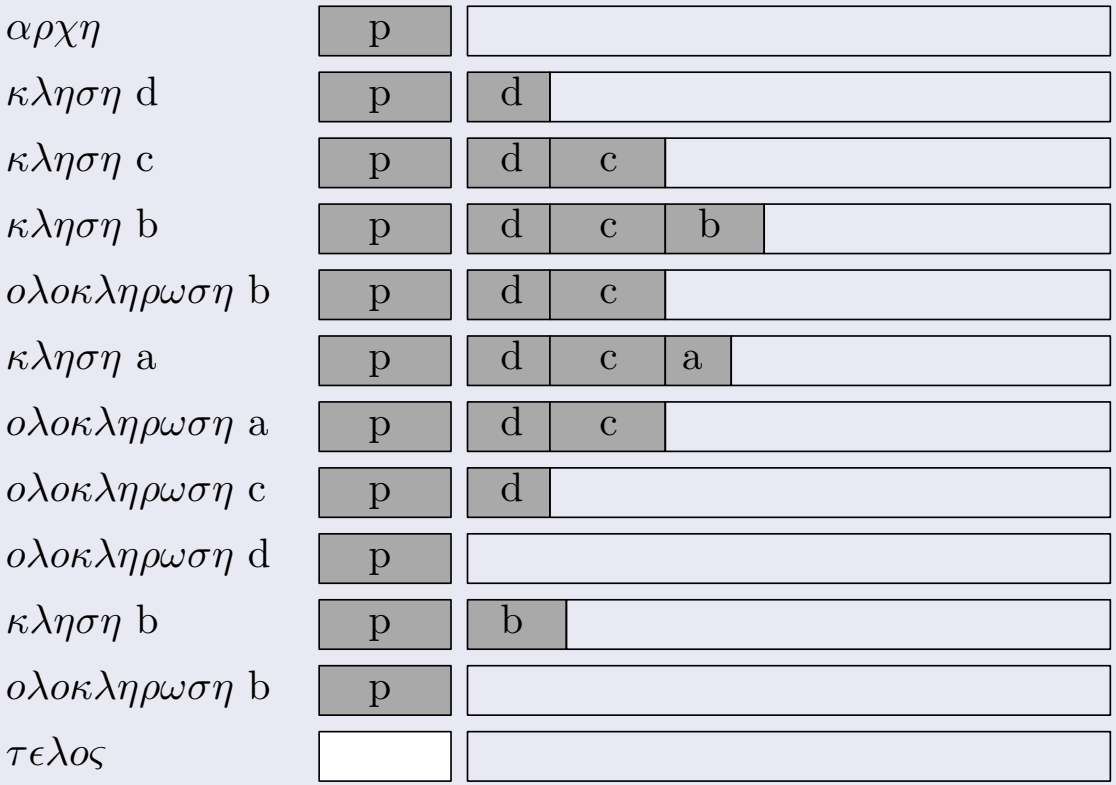
```
program p(...);
var ...;
   procedure a(...);
      var ...;
      begin ... end;
   procedure b(...);
      var ...;
      begin ... end;
   procedure c(...);
      var ...;
      begin ... b(...); ... a(...); ... end;
   procedure d(...);
      var ...;
      begin ... c(...); ... end;
begin (* of program p *)
     d(...); b(...)
end. (* of program p *)
```

Διάρκεια Ζωής και Διαχείριση Μνήμης

Στο παρακάτω σχήμα φαίνεται πως τροποποιείται το περιεχόμενο της στοίβας κατά την κλήση και την ολοκλήρωση κάθε υποπρογράμματος.

- Το κυρίως πρόγραμμα p καλεί αρχικά το υποπρόγραμμα d, το d καλεί το c και το c διαδοχικά τα υποπρογράμματα b και a. Στη συνέχεια το κυρίως πρόγραμμα p καλεί το b.
- Η στοίβα μεγαλώνει από αριστερά προς τα δεξιά (η κορυφή της στοίβας είναι το δεξιότερο σημείο της), ενώ ο χρόνος τρέχει από πάνω προς τα κάτω.
- Οι στατικές μεταβλητές αποθηκεύονται σε ξεχωριστό τμήμα της μνήμης, το οποίο εικονίζεται ακριβώς πρίν από την αρχή της στοίβας.

Διάρκεια Ζωής και Διαχείριση Μνήμης



Διάρκεια Ζωής και Διαχείριση Μνήμης

- Παρατηρούμε ότι εγγραφήματα δραστηριοποίησης διαφορετικών υποπρογραμμάτων (π.χ. των b και a) χρησιμοποιούν τις ίδιες θέσεις της στοίβας σε διαφορετικές χρονικές στιγμές.
- Επίσης τα εγγραφήματα δραστηριοποίησης διαφορετικών κλήσεων του ίδιου υποπρογράμματος (π.χ. του b) χρησιμοποιούν διαφορετικές θέσεις της στοίβας.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Ο σωρός χρησιμοποιείται για αντικείμενα που σχηματίζονται και καταστρέφονται με ακαθόριστο τρόπο. Η δέσμευση μνήμης για αυτά τα αντικείμενα γίνεται δυναμικά (σε χρόνο εκτέλεσης).

Επειδή η διάρκεια ζωής αυτών των αντικειμένων δεν μπορεί να προβλεφθεί τη στιγμή που δεσμεύεται η μνήμη για την αποθήκευσή τους, ενδέχεται σε κάποια χρονική στιγμή ο διαθέσιμος χώρος στο σωρό να αποτελείται από πολλά μικρά τμήματα τα οποία δεν είναι συνεχόμενα στη μνήμη. Αυτό σημαίνει ότι ένα (σχετικά μεγάλο) αντικείμενο ενδέχεται να μην χωράει στο σωρό, παρότι ο συνολικός διαθέσιμος χώρος υπερβαίνει το μέγεθός του.

Το φαινόμενο αυτό ονομάζεται εξωτερική κατάτμηση.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Η διαχείριση της διαθέσιμης μνήμης στο σωρό μπορεί να γίνει με διάφορους αλγόριθμους. Μία κατηγορία αλγορίθμων βασίζεται στη διατήρηση μίας συνδεδεμένης λίστας με όλα τα ελεύθερα τμήματα του σωρού. Η λειτουργία αυτών των αλγορίθμων συνοψίζεται παρακάτω:

- Αρχικά η λίστα περιέχει ένα μοναδικό τμήμα που καλύπτει ολόκληρο το σωρό, καθώς ολόκληρη η μνήμη του σωρού είναι διαθέσιμη.
- Όταν υπάρχει αίτημα για την κατασκευή ενός αντικειμένου, αναζητείται στη λίστα ένα τμήμα με μέγεθος επαρκές ώστε να αποθηκευτεί το αντικείμενο. Αν το τμήμα της μνήμης που επιλέγεται είναι μεγαλύτερο από το μέγεθος του αντικειμένου, τότε το αχρησιμοποίητο τμήμα αποθηκεύεται στη λίστα στη θέση του αρχικού.

Διάρκεια Ζωής και Διαχείριση Μνήμης

- Όταν καταστραφεί ένα αντικείμενο, το τμήμα της μνήμης που καταλάμβανε αποδεσμεύεται και εισάγεται στην κατάλληλη θέση της λίστας. Στη συνέχεια γίνεται έλεγχος για το αν κάποιο από τα γειτονικά τμήματα του σωρού είναι ελεύθερο. Στην περίπτωση αυτή τα δύο (ή τρία) τμήματα συνενώνονται σε ένα μοναδικό τμήμα.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Υπάρχουν διάφορες παραλλαγές του παραπάνω γενικού αλγόριθμου διαχείρισης του σωρού:

- Ο αλγόριθμος πρώτου ταιριάσματος ικανοποιεί το αίτημα για μνήμη χρησιμοποιώντας το πρώτο τμήμα που θα βρεί στη λίστα και το οποίο είναι αρκετά μεγάλο ώστε να ικανοποιήσει το αίτημα.
- Ο αλγόριθμος καλύτερου ταιριάσματος, ανάμεσα στα τμήματα της λίστας, επιλέγει το μικρότερο από αυτά που είναι αρκετά μεγάλα ώστε να μπορούν να ικανοποιήσουν το αίτημα.

Ο αλγόριθμος καλύτερου ταιριάσματος κάνει καλύτερη διαχείριση του σωρού, ωστόσο είναι πιο αργός καθώς σε κάθε περίπτωση πρέπει να εξετάσει ολόκληρη της λίστα.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Για τη μείωση του χρόνου αναζήτησης, έχουν αναπτυχθεί αλγόριθμοι που χρησιμοποιούν περισσότερες από μία λίστες, κάθε μία από τις οποίες περιέχει τμήματα διαφορετικού μεγέθους, και πιο πολύπλοκα κριτήρια επιλογής του τμήματος μνήμης που θα δεσμεύσουν.

Στην περίπτωση που για την ικανοποίηση ενός αιτήματος πρόκειται να χρησιμοποιηθεί ένα τμήμα του σωρού λίγο μεγαλύτερο από το μέγεθος που απαιτείται, ορισμένοι αλγόριθμοι επιλέγουν να δεσμέυσουν ολόκληρο το διαθέσιμο τμήμα και να μην αφήσουν το υπόλοιπο ως διαθέσιμο.

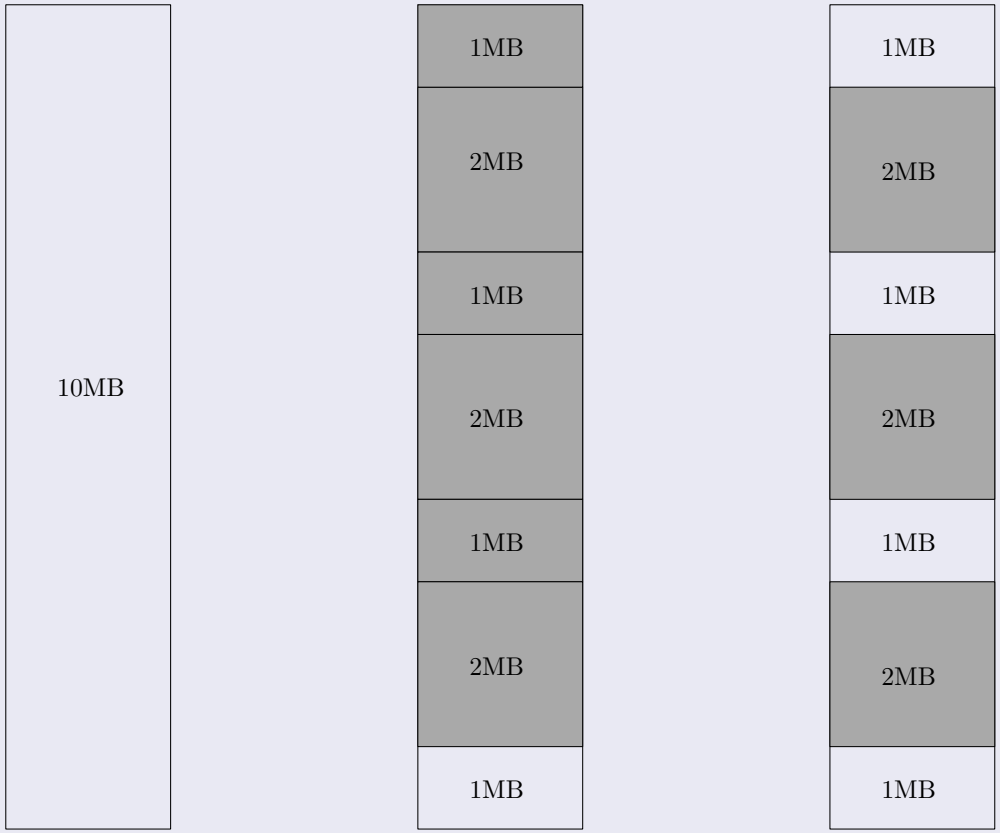
Το αποτέλεσμα αυτής της επιλογής ονομάζεται εσωτερική κατάτμηση.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Παράδειγμα

Έστω ότι η μνήμη που διατίθεται για το σωρό έχει μέγεθος 10MB και ότι η διαχείριση γίνεται με τον αλγόριθμο καλύτερου ταιριάσματος. Αν αρχικά κατασκευαστούν κατά σειρά αντικείμενα μεγέθους 1MB, 2MB, 1MB, 2MB, 1MB και 2MB, και στη συνέχεια καταστραφούν όλα τα αντικείμενα μεγέθους 1MB, συνολικά υπάρχουν ελεύθερα 4MB στα οποία όμως δεν μπορεί να αποθηκευτεί ένα αντικείμενο μεγέθους 2MB.

Διάρκεια Ζωής και Διαχείριση Μνήμης



Διάρκεια Ζωής και Διαχείριση Μνήμης

Εκτός από το πρόβλημα της εξωτερικής κατάτμησης, ένα άλλο πρόβλημα που μειώνει τη διαθέσιμη μνήμη στο σωρό είναι η ύπαρξη σκουπιδιών.

Σκουπίδια ονομάζονται τα αντικείμενα τα οποία είναι αποθηκευμένα στο σωρό, χωρίς ωστόσο να υπάρχει πλέον πρόσβαση προς αυτά από το πρόγραμμα.

Και τα δύο προβλήματα, αν δεν αντιμετωπιστούν από το περιβάλλον εκτέλεσης του προγράμματος, είναι καταστροφικά για την εκτέλεση του προγράμματος, καθώς αν ένα αίτημα για δημιουργία αντικειμένου δεν μπορέσει να ικανοποιηθεί τότε αυτό μπορεί να συνεπάγεται ανώμαλο τερματισμό της λειτουργίας του προγράμματος.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Για την αντιμετώπιση της εξωτερικής κατάτμησης γίνεται ανασυγκρότηση του σωρού (μετακίνηση των ενεργών αντικειμένων σε συνεχόμενες θέσεις μνήμης, έτσι ώστε η διαθέσιμη μνήμη να αποτελείται από συνεχόμενες θέσεις).

Η διαδικασία για την εύρεση των σκουπιδιών, την καταστροφή τους και την επιστροφή της μνήμης που καταλαμβάνουν στο σωρό ονομάζεται συλλογή σκουπιδιών.

Η δημιουργία σκουπιδιών είναι συνειδητή επιλογή σε πολλές γλώσσες, ώστε να αποφεύγονται χειρότερα προβλήματα (όπως οι ξεκρέμαστοι δείκτες).

Διάρκεια Ζωής και Διαχείριση Μνήμης

Στο μοντέλο τιμών για τις μεταβλητές:

- στατικά αντικείμενα είναι τα αντικείμενα που συνδέονται με καθολικές μεταβλητές του προγράμματος ή με στατικές μεταβλητές υποπρογραμμάτων (τοπικές μεταβλητές που διατηρούν τις τιμές τους ανάμεσα σε διαδοχικές κλήσεις ενός υποπρογράμματος). Επίσης στατικά αντικείμενα είναι και διάφορα βοηθητικά αντικείμενα που κατασκευάζονται από το μεταφραστή και αντιστοιχούν σε πληροφορίες που είναι χρήσιμες στο χρόνο εκτέλεσης.

Διάρκεια Ζωής και Διαχείριση Μνήμης

- αντικείμενα στοίβας είναι τα αντικείμενα που συνδέονται με δυναμικές μεταβλητές των υποπρογραμμάτων (τοπικές μεταβλητές που η τιμή τους χάνεται μετά την ολοκλήρωση του υποπρογράμματος)
- αντικείμενα σωρού είναι τα αντικείμενα που συνδέονται με μεταβλητές τύπου δείκτη και σχηματίζονται με εντολές όπως η new της Pascal και η malloc τη C.

Διάρκεια Ζωής και Διαχείριση Μνήμης

Στο μοντέλο αναφορών για τις μεταβλητές:

- όλα τα αντικείμενα αποθηκεύονται στο σωρό.
- στη στοίβα αποθηκεύονται οι δυναμικές μεταβλητές των υποπρογραμμάτων (δηλαδή οι αναφορές των αντικειμένων με τα οποία συνδέονται οι μεταβλητές)
- στατικά αποθηκεύονται οι καθολικές μεταβλητές του προγράμματος και οι στατικές μεταβλητές υποπρογραμμάτων.

Περιεχόμενα

1 Διάρκεια Ζωής και Διαχείριση Μνήμης

2 Εμβέλεια

3 Παραστάσεις

Εμβέλεια

Εμβέλεια μίας μεταβλητής (ή γενικότερα κάποιου ονόματος π.χ σταθεράς, συνάρτησης κ.λ.π.) ονομάζεται το τμήμα του προγράμματος στο οποίο η μεταβλητή είναι ορατή.

Ονομάζουμε περιβάλλον αναφοράς ενός μπλοκ το σύνολο των μεταβλητών που είναι ορατές στο μπλοκ.

Εμβέλεια

Η εμβέλεια μίας μεταβλητής γενικά δεν καλύπτει ολόκληρο το πρόγράμμα. Αυτό είναι χρήσιμο για αρκετούς λόγους:

- Ο προγραμματιστής μπορεί να χρησιμοποιεί ονόματα μεταβλητών σε ένα τμήμα κώδικα που ενδεχομένως χρησιμοποιούνται και σε κάποιο άλλο τμήμα του προγράμματος, χωρίς να δημιουργείται σύγκρουση.
- Μπορεί να τροποποιηθεί ένα τμήμα του κώδικα (π.χ. ένα υποπρόγραμμα) χωρίς να απαιτούνται αλλαγές (δηλώσεις μεταβλητών) έξω από αυτό.
- Το πρόγραμμα είναι πιο αναγνώσιμο αν οι μεταβλητές δηλώνονται κοντά στο σημείο που χρησιμοποιούνται.
- ...

Εμβέλεια

Η εμβέλεια των μεταβλητών σε μία γλώσσα καθορίζεται από τους κανόνες εμβέλειας. Υπάρχουν δύο βασικοί κανόνες εμβέλειας:

- ο στατικός κανόνας εμβέλειας (Pascal, C, Java, Haskell)
- ο δυναμικός κανόνας εμβέλειας (APL, SNOBOL, Lisp, Perl)

Εμβέλεια

Όταν εφαρμόζεται ο στατικός κανόνας, η εμβέλεια μίας μεταβλητής καθορίζεται από το κείμενο του προγράμματος.

Με βάση το στατικό κανόνα, η εμβέλεια μίας μεταβλητής ορίζεται ως το μπλοκ στο οποίο δηλώνεται, καθώς και τα εσωτερικά μπλοκ που περιέχονται σε αυτό, υπό την προϋπόθεση ότι δεν δηλώνεται μέσα σε αυτά μεταβλητή με το ίδιο όνομα.

Εμβέλεια

Παράδειγμα

Έστω το παρακάτω πρόγραμμα Pascal:

Εμβέλεια

```
program p(...);
var s,z:integer; ...

    procedure a(...);
        var x:integer; ...

        procedure c(...);
            var z:char; ...
        ...
    procedure b(...);
        var y:real; ...

        procedure d(...);
            var u:char; ...
        procedure e(...);
            var v:char; ...
        ...
begin (* of program p *) ...
end.
```

Εμβέλεια

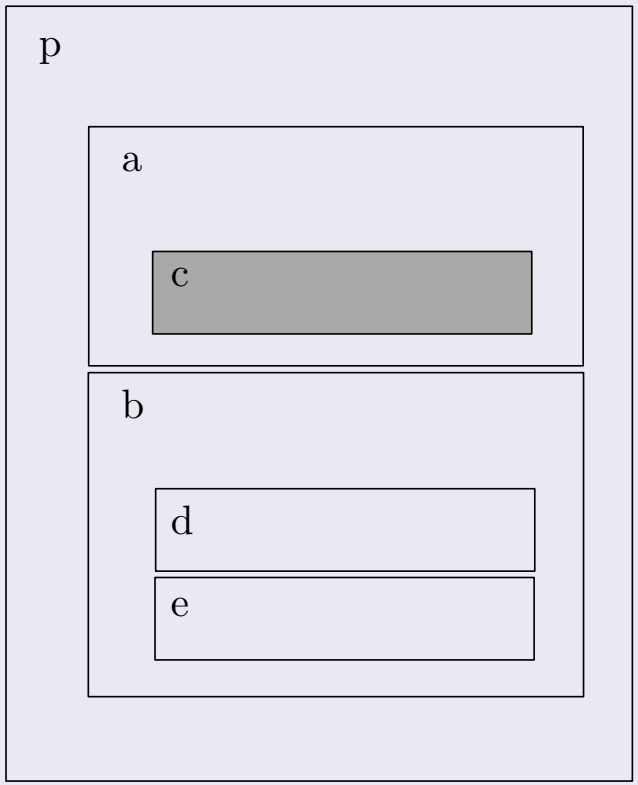
- Η μεταβλητή s που δηλώνεται στο κυρίως πρόγραμμα p είναι ορατή σε ολόκληρο το πρόγραμμα.
- Η μεταβλητή z που δηλώνεται στο κυρίως πρόγραμμα p είναι είναι ορατή σε ολόκληρο το πρόγραμμα εκτός από το υποπρόγραμμα c στο οποίο δηλώνεται άλλη μεταβλητή με το ίδιο όνομα.
- Οι μεταβλητές που ορίζονται στην a (π.χ. η x) είναι ορατές στην a και στη c (αλλά όχι στις b, d, e).
- Οι μεταβλητές που ορίζονται στην b (π.χ. η y) είναι ορατές στις b, d και e.
- Οι μεταβλητές που ορίζονται στην c (π.χ. η z) είναι ορατές μόνο στη c (αντίστοιχα για d, e).

Εμβέλεια

Παρατηρούμε ότι αν σε δύο μπλοκ δηλώνονται μεταβλητές με το ίδιο όνομα και το ένα περιέχεται μέσα στο άλλο (όπως συμβαίνει με το πρόγραμμα p και το υποπρόγραμμα c στο παράδειγμα στα οποία δηλώνονται μεταβλητές με το όνομα z) τότε η εμβέλεια της μεταβλητής που δηλώνεται στο εξετερικό μπλοκ δεν περιλαμβάνει το εσωτερικό μπλοκ. Αυτό το φαινόμενο ονομάζεται τρύπα στην εμβέλεια.

Η εμβέλεια της μεταβλητής z του προγράμματος p του παραδείγματος απεικονίζεται με λευκό χρώμα στο παρακάτω σχήμα:

Εμβέλεια



Εμβέλεια

Όταν εφαρμόζεται ο δυναμικός κανόνας, η εμβέλεια μίας μεταβλητής καθορίζεται από τη σειρά κλήσεων των μπλοκ του προγράμματος.

Με βάση το δυναμικό κανόνα, η εμβέλεια μίας μεταβλητής ορίζεται ως το μπλοκ στο οποίο δηλώνεται, καθώς και όλα τα μπλοκ που ενεργοποιούνται (αρχίζουν να εκτελούνται) μετά από αυτό, μέχρι να ενεργοποιηθεί μπλοκ στο οποίο δηλώνεται μεταβλητή με το ίδιο όνομα.

Εμβέλεια

Παράδειγμα

```
program p(...);
var x:integer;

    procedure a(...);
        var ...
        begin ... end

    procedure b(...);
        var x:real; ...
        begin ... call a ... end;

    procedure c(...);
        var ...
        begin ... call b ... end;


begin (* of program p *) ...
... call c ...
end.
```

Εμβέλεια

Η ακολουθία κλήσεων είναι:
p $\rightarrow$ c $\rightarrow$ b $\rightarrow$ a

τότε η εμβέλεια της μεταβλητής x της που δηλώνεται στο πρόγραμμα p φτάνει μέχρι και την κλήση της c. Η κλήση της a βλέπει τη μεταβλητή x της που δηλώνεται στην b.

Εμβέλεια

Η χρήση των κανόνων εμβέλειας και ειδικότερα του δυναμικού κανόνα, μπορεί να βοηθήσει τον προγραμματιστή να μειώσει τον αριθμό των παραμέτρων ενός υποπρογράμματος.

Παράδειγμα

Παράδειγμα: Εστω το παρακάτω πρόγραμμα Pascal:

Εμβέλεια

Παράδειγμα

```
program CentralShop(output);
var price:real;

    procedure TotalCost(quantity:integer);
        begin writeln(quantity*price:10:2) end;

    procedure DepartmentA(quantity:integer);
        var price:real;
        begin price:=32.00; TotalCost(quantity)
        end;

    procedure DepartmentB(quantity:integer);
        var price:real;
        begin price:=33.33; TotalCost(quantity);
        end;

begin (* of program p *)
  price:=31.20; TotalCost(10); DepartmentA(5); DepartmentB(3)
end.
```

Εμβέλεια

- Αν εφαρμόστει ο στατικός κανόνας εμβέλειας (όπως συμβαίνει στην Pascal) το πρόγραμμα θα τυπώσει τις τιμές 312.00, 156.00, 93.60.
  Αυτό συμβαίνει επειδή η εμβέλεια της μεταβλητής price η οποία δηλώνεται στο πρόγραμμα CentralShop καλύπτει το υποπρόγραμμα TotalCost, ενώ αντίθετα οι μεταβλητές price της DepartmentA και της DepartmentB δεν είναι ορατές μέσα στην TotalCost.
- Αν εφαρμόστει ο δυναμικός κανόνας εμβέλειας το πρόγραμμα θα τυπώσει τις τιμές 312.00, 160.00, 99.99.
  Αυτό συμβαίνει επειδή κάθε φορά το υποπρόγραμμα TotalCost βρίσκεται στην εμβέλεια μίας διαφορετική μεταβλητής price, που εξαρτάται από το μπλοκ από το οποίο κλήθηκε (το CentralShop, το DepartmentA και το DepartmentB αντίστοιχα).

Εμβέλεια

Ονομάζουμε μή τοπική μεταβλητή, μία μεταβλητή που είναι ορατή σε ένα μπλοκ, χωρίς όμως να έχει δηλωθεί σε αυτό.

Οι κανόνες εμβέλειας μας βοηθούν να καθορίσουμε σε ποιά μη τοπική μεταβλητή αναφέρεται ένα όνομα, το οποίο δεν έχει δηλωθεί στο μπλοκ.

Εμβέλεια

Με βάση το στατικό κανόνα εμβέλειας αναζητούμε μία μεταβλητή ξεκινώντας από το τρέχον μπλοκ και προχωρώντας κάθε φορά στο άμεσα εξωτερικότερο μπλοκ, μέχρι να συναντήσουμε δήλωση της μεταβλητής. Αν η μεταβλητή υπάρχει σε ένα μπλοκ για το οποίο υπάρχουν περισότερες από μία ενεργοποιήσεις (λόγω αναδρομής) τότε λαμβάνεται υπόψη η πιο πρόσφατη ενεργοποίηση.

Με βάση το δυναμικό κανόνα εμβέλειας αναζητούμε μία μεταβλητή ξεκινώντας από το τρέχον μπλοκ και ακολουθώντας αντίστροφα την αλυσίδα των κλήσεων, μέχρι να συναντήσουμε μπλοκ που περιέχει δήλωση της μεταβλητής.

Εμβέλεια

Παράδειγμα

```
program p(output);
var x,y,z:integer;
    procedure a;
        var w:integer;
        begin
           x:=x+1; y:=y+1; z:=z+1
        end;
    procedure b(x:integer);
        var z:integer;
        procedure c(z:integer);
           begin
              a; writeln(x,y,z);
           end;
        begin (* of procedure b *)
           z:=50; a; c(60); writeln(x,y,z);
        end;
begin (* of program p *)
  x:=10; y:=20; z:=30; a; b(40); writeln(x,y,z)
end.
```

Εμβέλεια

Επειδή στο πρόγραμμα γίνονται αρκετές αλλαγές στις τιμές των μεταβλητών και υπάρχουν μεταβλητές με το ίδιο όνομα, παρουσιάζουμε την εκτέλεση του προγράμματος σε μορφή πίνακα.

Η πρώτη στήλη του πίνακα αναγράφει την εντολή (ή γενικότερα τη λειτουργία) που εκτελεί το πρόγραμμα, ενώ κάθε μία από τις υπόλοιπες στήλη αντιστοιχεί σε μία μεταβλητή που αποθηκεύεται στη στοίβα κατά την εκτέλεση του προγράμματος.

Για απλούστευσή, θεωρούμε ότι οι στατικές μεταβλητές του προγράμματος αποθηκεύονται στην αρχή της στοίβας.

Εμβέλεια

Η στοίβα μεγαλώνει από τα αριστερά προς τα δεξιά. Η κορυφή της μετατοπίζεται προς τα δεξιά κάθε φορά που καλείται ένα υποπρόγραμμα και προς τα αριστερά κάθε φορά που ολοκληρώνεται ένα υποπρόγραμμα.

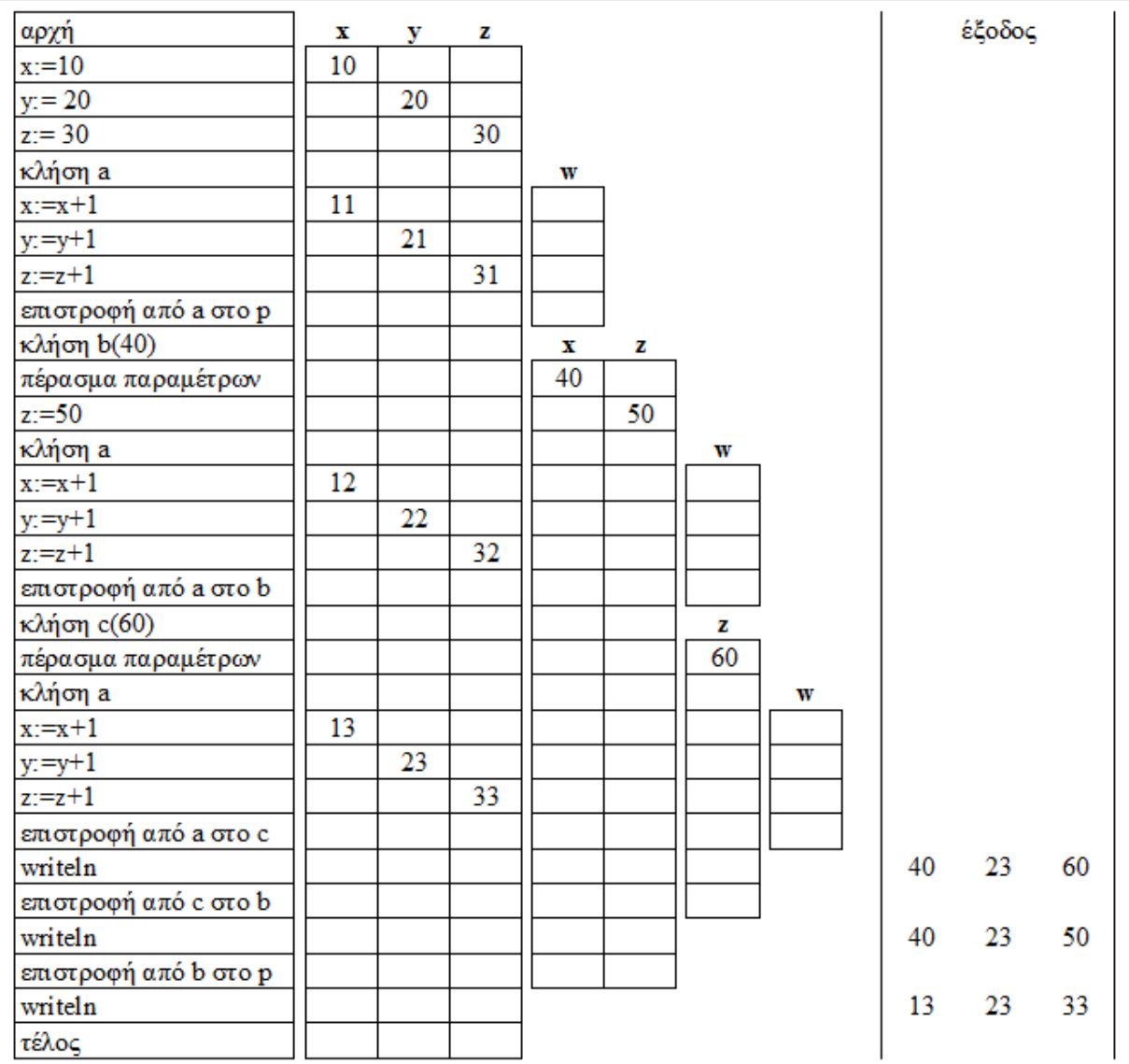
Ο χρόνος τρέχει από πάνω προς τα κάτω.

Στον πίνακα αποτυπώντουμε μόνο τις θέσεις που αντιστοιχούν σε μεταβλητές και όχι ολόκληρο το εγγράφημα δραστηριοποίησης που αποθηκεύται στη στοίβα.

Εμβέλεια

Στον πίνακα που ακολουθεί δίνεται η εκτέλεση του προγράμματος στην περίπτωση που εφαρμόζεται ο στατικός κανόνας εμβέλειας.

Εμβέλεια



αρχή	x	y	z	w	x	z	w	z	w	έξοδος		
x:=10	10											
y:= 20		20										
z:= 30			30									
κλήση a												
x:=x+1	11											
y:=y+1		21										
z:=z+1			31									
επιστροφή από a στο p												
κλήση b(40)					x	z						
πέρασμα παραμέτρων					40							
z:=50						50						
κλήση a												
x:=x+1	12											
y:=y+1		22										
z:=z+1			32									
επιστροφή από a στο b												
κλήση c(60)								z				
πέρασμα παραμέτρων								60				
κλήση a									w			
x:=x+1	13											
y:=y+1		23										
z:=z+1			33									
επιστροφή από a στο c												
writeln											40	23	60
επιστροφή από c στο b												
writeln											40	23	50
επιστροφή από b στο p												
writeln											13	23	33
τέλος												

Εμβέλεια

Παρατηρήσεις:

- Το υποπρόγραμμα a αναφέρεται στις μεταβλητές x,y,z οι οποίες δεν δηλώνονται μέσα σε αυτό. Με βάση το στατικό κανόνα, οι x,y,z αναζητούνται στο αμέσως εξωτερικότερο μπλοκ που είναι το κυρίως πρόγραμμα, όπου πράγματι υπάρχει δήλωση των μεταβλητών αυτών.
  Συνεπώς το υποπρόγραμμα a σε κάθε μία από τις τρεις κλήσεις του αυξάνει κατά ένα τις τιμές των μεταβλητών x,y,z του κυρίως προγράμματος.
- Το υποπρόγραμμα c αναφέρεται στη μεταβλητή x η οποία δεν δηλώνονται μέσα σε αυτό. Με βάση το στατικό κανόνα, η x αναζητείται στο αμέσως εξωτερικότερο μπλοκ που είναι το υποπρόγραμμα b. Στο b πράγματι υπάρχει δήλωση της x.
  Συνεπώς το υποπρόγραμμα c τυπώνει την τιμή της x του υποπρογράμματος b.

Εμβέλεια

- Το υποπρόγραμμα c αναφέρεται στη μεταβλητή y η οποία δεν δηλώνονται μέσα σε αυτό. Με βάση το στατικό κανόνα, η y αναζητείται στο αμέσως εξωτερικότερο μπλοκ που είναι το υποπρόγραμμα b, όπου ωστόσο δεν υπάρχει δήλωση της y. Η αναζήτηση συνεχίζεται στο μπλοκ που περικλείει άμεσα το υποπρόγραμμα b, που είναι το κυρίως πρόγραμμα. Εκεί υπάρχει δήλωση της μεταβλητής y.
  Συνεπώς το υποπρόγραμμα c τυπώνει την τιμή της y του κυρίως προγράμματος.

Εμβέλεια

- Το υποπρόγραμμα c αναφέρεται στη μεταβλητή z για την οποία υπάρχει δήλωση μέσα σε αυτό. Για αυτή τη μεταβλητή δεν χρειάζεται να εφαρμοστεί ο κανόνας εμβέλειας και τυπώνεται η τιμή της τοπικής μεταβλητής z.
- Αντίστοιχα το υποπρόγραμμα b τυπώνει τις τιμές των τοπικών μεταβλητών x,z και την τιμή της μεταβλητής y του κυρίως προγράμματος.

Εμβέλεια

Στον πίνακα που ακολουθεί δίνεται η εκτέλεση του προγράμματος στην περίπτωση που εφαρμόζεται ο δυναμικός κανόνας εμβέλειας.

Εμβέλεια

αρχή	x	y	z	w	x	z	w	w	z	w	έξοδος		
x:=10	10												
y:=20		20											
z:=30			30										
κλήση a													
x:=x+1	11												
y:=y+1		21											
z:=z+1			31										
επιστροφή από a στο p													
κλήση b(40)													
πέρασμα παραμέτρων					40								
z:=50						50							
κλήση a													
x:=x+1					41								
y:=y+1		22											
z:=z+1													
επιστροφή από a στο b						51							
κλήση c(60)													
πέρασμα παραμέτρων							60						
κλήση a													
x:=x+1					42								
y:=y+1		23											
z:=z+1							61						
επιστροφή από a στο c													
writeln											42	23	61
επιστροφή από c στο b													
writeln											42	23	51
επιστροφή από b στο p													
writeln											11	23	31
τέλος													

Εμβέλεια

Παρατηρήσεις:

- Κατά την πρώτη κληση του υποπρογράμματος a οι μεταβλητές x,y,z οι οποίες δεν δηλώνονται μέσα σε αυτό, αναζητούνται στο κυρίως πρόγραμμα, που είναι το μπλοκ από το οποίο έγινε η κλήση του υποπρογράμματος a. Στο κυρίως πρόγραμμα υπάρχει δήλωση των μεταβλητών x,y,z.
  Συνεπώς το υποπρόγραμμα a στην πρώτη κλήση του αυξάνει κατά ένα τις τιμές των μεταβλητών x,y,z του κυρίως προγράμματος.

Εμβέλεια

Παρατηρήσεις:

- Κατά την δεύτερη κληση του υποπρογράμματος a οι μεταβλητές x,y,z, αναζητούνται στο υποπρόγραμμα b από το οποίο έγινε η κλήση του a.
  Στο b υπάρχει δήλωση των x,z, ωστόσο δεν υπάρχει δήλωση της y.
  Η αναζήτηση για την y συνεχίζεται στο μπλοκ που κάλεσε το b, το οποίο είναι το κυρίως πρόγραμμα. Στο κυρίως πρόγραμμα υπάρχει δήλωση της μεταβλητής y.
  Συνεπώς το υποπρόγραμμα a στην δεύτερη κλήση του αυξάνει κατά ένα τις τιμές των μεταβλητών x,z του υποπρογράμματος b καθώς και της μεταβλητής y του κυρίως προγράμματος.

Εμβέλεια

- Κατά την τρίτη κληση του υποπρογράμματος a η αναζήτηση γίνεται με παρόμοιο τρόπο. Ακολουθώντας αντίστροφα την σειρά των κλήσεων, η μεταβλητή z εντοπίζεται στο υποπρόγραμμα c, η x στο υποπρόγραμμα a και η y στο κυρίως πρόγραμμα.
  Συνεπώς το υποπρόγραμμα a στην τρίτη κλήση του αυξάνει κατά ένα τις τιμές της μεταβλητής z του υποπρογράμματος b, της x του υποπρογράμματος b και της y του κυρίως προγράμματος.
- Με παρόμοιο τρόπο γίνεται η εφαρμογή του δυναμικού κανόνα για τις μη τοπικές μεταβλητές των υποπρογραμμάτων b και c.

Εμβέλεια

Παρατηρούμε ότι

- με βάση το στατικό κανόνα όλες οι αναφορές ενός ονόματος μεταβλητής μέσα από ένα μπλοκ αναφέρονται στην ίδια δήλωση.
- με βάση το δυναμικό κανόνα η ίδια αναφορά ενός ονόματος μεταβλητής μέσα από ένα μπλοκ μπορεί να αναφέρεται σε διαφορετικές δηλώσεις, για διαφορετικές ενεργοποιήσεις του μπλοκ.

Εμβέλεια

Μία μεταβλητή ή γενικότερα ένα όνομα μπορεί να δηλώνεται σε διάφορα σημεία ενός μπλόκ. Υπάρχουν διαφορετικοί τρόποι συσχέτισης της εμβέλειας του ονόματος με το μπλοκ που την περιέχει:

- Η εμβέλεια του ονόματος καλύπτει ολόκληρο το μπλοκ, ακόμη και το τμήμα που προηγείται της δήλωσης (π.χ ονόματα συναρτήσεων στη Haskell και μελών μίας κλάσης στη Java και τη C++)
- Η εμβέλεια του ονόματος καλύπτει το τμήμα του μπλοκ που ακολουθεί τη δήλωση. Αν υπάρχει δήλωση του ίδιου ονόματος σε εξωτερικό μπλοκ, τότε η εμβέλεια του ονόματος στο εξωτερικό μπλοκ καλύπτει και το τμήμα του τρέχοντος μπλοκ μέχρι τη δήλωση. (π.χ. ονόματα μεταβλητών στη C, C++, Java)
- Η εμβέλεια του ονόματος καλύπτει ολόκληρο το μπλοκ, ακόμη και το τμήμα που προηγείται της δήλωσης, ωστόσο το όνομα μπορεί να χρησιμοποιηθεί μόνο μετά τη δήλωση (π.χ. ονόματα μεταβλητών στη Pascal, C#)

Εμβέλεια

Πολλές γλώσσες προγραμματισμού περιέχουν ένα σύνολο από προκαθορισμένα ονόματα σταθερών, τύπων, συναρτήσεων κλπ. Αυτές μπορούμε να θεωρήσουμε ότι ανήκουν σε ένα τεχνητό μπλοκ.

Για το στατικό κανόνα εμβέλειας, αυτό το μπλοκ θεωρούμε ότι περιέχει το μπλοκ του κυρίως προγράμματος.

Για το δυναμικό εμβέλειας, θεωρούμε ότι αυτό το μπλοκ βρίσκεται στην αρχή της ακολουθίας των κλήσεων.

Εμβέλεια

Ορισμένες γλώσσες παρέχουν τρόπους ώστε να αποφεύγεται η τρύπα στη εμβέλεια μία μεταβλητής όταν δηλώνεται μεταβλητή με το ίδιο όνομα σε εσωτερικότερο μπλοκ.

Για παράδειγμα στη C++ το ::X αναφέρεται στην καθολική μεταβλητή X.

Επίσης στην Ada μπορούμε να αναφερθούμε ρητά στη μεταβλητή x που δηλώνεται στο υποπρόγραμμα a γράφοντας a.x.

Εμβέλεια

Στις γλώσσες προγραμματισμού στις οποίες δεν υπάρχουν δηλώσεις μεταβλητών (όπως για παράδειγμα η Python), η εισαγωγή μίας τοπικής μεταβλητής γίνεται έμμεσα από τον τρόπο χρήσης της, με βάση κάποιους κανόνες που εξαρτώνται από τη γλώσσα.

Στην Python όταν ένα υποπρόγραμμα αναθέτει τιμή σε μία μεταβλητή, τότε αυτή θεωρείται αυτόματα τοπική μεταβλητή.

Η Python εφαρμόζει το στατικό κανόνα εμβέλειας.

Εμβέλεια

Παράδειγμα

Εστω το παρακάτω πρόγραμμα Python:

```
def f():
        x = y+1
        print(x,y)

x = 1
y = 5
f()
print(x,y)
```

Εμβέλεια

Η εντολή x = y+1 του υποπρογράμματος f δηλώνει έμμεσα μία τοπική μεταβλητή x.

Αντίθετα το όνομα y αναφέρεται στην μεταβλητή y του κυρίως προγράμματος, καθώς δεν γίνεται ανάθεση τιμής στην y εντός του υποπρογράμματος f και συνεπώς η y είναι μη τοπική μεταβλητή.

Αν εκτελεστεί το παραπάνω πρόγραμμα θα τυπωθούν οι τιμές 6 5 από την f και 1 5 από το κυρίως πρόγραμμα.

Εμβέλεια

Παράδειγμα

Αν μέσα σε ένα υποπρόγραμμα της Python θέλουμε να αναθέσουμε τιμή σε μη τοπική μεταβλητή, τότε θα πρέπει να χρησιμοποιήσουμε μία από τις εντολές global ή nonlocal:

```
def f():
        global x
        x = y+1
        print(x,y)

x = 1
y = 5
f()
print(x,y)
```

Εμβέλεια

Η εντολή global x του υποπρογράμματος f δηλώνει ότι το όνομα x αναφέρεται στην μεταβλητή x του κυρίως προγράμματος.

Συνεπώς καμία μεταβλητή στο υποπρογράμματος f δεν είναι τοπική.

Αν εκτελεστεί το παραπάνω πρόγραμμα θα τυπωθούν οι τιμές 6 5 από την f και 6 5 από το κυρίως πρόγραμμα.

Περιεχόμενα

1 Διάρκεια Ζωής και Διαχείριση Μνήμης

2 Εμβέλεια

3 Παραστάσεις

Παραστάσεις

Οι συμβολικές σταθερές είναι ονόματα τα οποία αντιστοιχούν σε τιμές, οι οποίες δεν μπορούν να αλλάξουν κατά την εκτέλεση του προγράμματος.

Η χρήση σταθερών βοηθάει την αναγνωσιμότητα και διευκολύνει τη συντήρηση του προγράμματος.

Παραστάσεις

Υπάρχουν δύο προσεγγίσεις για τις συμβολικές σταθερές:

- Αντικαθίστανται από την τιμή τους σε χρόνο μετάφρασης
- Δημιουργούνται μεταβλητές που η τιμή τους δεν μπορεί να αλλάξει. Αυτό είναι χρήσιμο αν η σταθερά είναι πίνακας ή αν η τιμή της σταθερά καθορίζεται σε χρόνο εκτέλεσης.

Παραστάσεις

Οι γλώσσες συνήθως έχουν κάποιες προκαθορισμένες σταθερές.

Για παράδειγμα η PASCAL έχει προκαθορισμένες τις σταθερές true, false, maxint, nil.

Ορισμένες γλώσσες επιτρέπουν η τιμή της σταθεράς να ορίζεται από μία παράσταση.

Επίσης ορισμένες γλώσσες επιτρέπουν σταθερές σύνθετου τύπου.

Παραστάσεις

Μία παράσταση σχηματίζεται από σταθερές (συμβολικές ή κυριολεκτικές), μεταβλητές (ή στοιχεία πινάκων, πεδία εγγραφών κλπ), συναρτήσεις, τελεστές και παρενθέσεις.

- Οι μεταβλητές και οι σταθερές αποτελούν απλές παραστάσεις.
- Οι τελεστές και οι συναρτήσεις χρησιμοποιούνται ώστε να σχηματιστούν σύνθετες παραστάσεις από πιο απλές (τις οποίες δέχονται ως ορίσματα).
- Οι παρενθέσεις χρησιμοποιούνται για να καθοριστεί η σειρά εφαρμογής των τελεστών. Επίσης συνήθως τα ορίσματα μία συνάρτησης δίνονται μέσα σε παρενθέσεις χωρισμένα μέ κόμμα.

Παραστάσεις

Οι τελεστές και οι συναρτήσεις έχουν παρόμοια χρησιμότητα.

Συνήθως ο όρος τελεστής χρησιμοποιείται για λειτουργίες πάνω σε βασικούς τύπους, οι οποίες συμβολίζονται με χρήση μή αλφαριθμητικών συμβόλων (πχ +, -, *, /, >>, :, ++, ...). Κατα κανόνα οι τελεστές είναι προκαθορισμένοι, αν και ορισμένες γλώσσες δίνουν τη δυνατότητα ορισμού τελεστών από τον προγραμματιστή.

Ο όρος συνάρτηση χρησιμοποιείται συνήθως για υποπρογράμμα τα οποία επιστρέφουν κάποιο αποτέλεσμα και τα οποία έχουν ονόματα που σχηματίζονται με γράμματα του αλφαβήτου και αριθμητικά σύμβολα. Ο ορισμός των συναρτήσεων αποτελεί μέρος ενός προγράμματος, ενώ οι περισσότερες γλώσσες έχουν κάποιες προκαθορισμένες συναρτήσεις.

Παραστάσεις

Οι συναρτήσεις και οι τελεστές δέχονται ένα πλήθος ορισμάτων, από τα οποία εξαρτάται το αποτέλεσμα που επιστρέφει η συνάρτηση ή ο τελεστής.

Μία συνάρτηση (ή ένας τελεστής) λέμε ότι χρησιμοποιεί προθεματικό, ενθεματικό ή επιθεματικό συμβολισμό, αν το όνομα της (ή το σύμβολο, στην περίπτωση του τελεστή) γράφεται πριν, ανάμεσα ή μετά από τα ορίσματα.

Παραστάσεις

Οι περισσότερες γλώσσες χρησιμοποιούν προθεματικό συμβολισμό για τις συναρτήσεις με διάφορες παραλλαγές ως προς τη σύνταξη:

- Pascal, C: f(1,2,3)
- Haskell: f 1 2 3
- Lisp: (f 1 2 3)

Παραστάσεις

Επίσης, οι περισσότερες γλώσσες χρησιμοποιούν ενθεματικό συμβολισμό για τους δυαδικούς τελεστές και προθεματικό συμβολισμό για τους μοναδιαίους τελεστές:

- 1+2, 3*4, 10/4, x && y, x || y
- 1:[2,3] (προσθήκη στοιχείου σε λίστα στη Haskell)
- "Programming" + "Languages" (συνένωση συμβολοσειρών στη Java)
- -q
- !p (λογική άρνηση στη C)
- *p (προσπέλαση μνήμης μέσω δείκτη στη C)

Παραστάσεις

Η Pascal χρησιμοποιεί επιθεματικό τελεστή για την προσπέλαση μνήμης μέσω δείκτη: p^

Η Lisp χρησιμοποιεί προθεματικό συμβολισμό για όλους τους τελεστές, όπως και για τις συναρτήσεις, βάζοντας τον τελεστή μαζί με τα ορίσματα μέσα σε παρένθεση:

(+ 1 2)

Ο συμβολισμός αυτός είναι γνωστός ως Πολωνικός συμβολισμός.

Παραστάσεις

Μια παράσταση μπορεί να εμφανίζεται:

- Στο δεξί μέλος μίας εντολής ανάθεσης.
- Στη συνθήκη ελέγχου μίας εντολής απόφασης ή επανάληψης.
- Ως πραγματική παράμετρος σε μία κλήση υποπρογράμματος.
- Μέσα σε [ ] ώστε να προσδιορίζει το στοιχείο ενός πίνακα.
- ...

Παραστάσεις

Ονομάζουμε αποτίμηση της παράστασης τη διαδικασία υπολογισμού της τιμής της.

Η τιμή μίας παράστασης που περιέχει συναρτήσεις προκύπτει από τις τιμές των απλών παραστάσεων που την αποτελούν και των τιμών που επιστρέφονται από τις συναρτήσεις, με εφαρμογή των τελεστών.

Η σειρά εφαρμογής των τελεστών, επηρεάζει την τιμή της παράστασης.

Παραστάσεις

Για παράδειγμα η αποτίμηση της παράστασης 2 + 3 * 5 δίνει τιμή

- 17, αν εκτελέσουμε πρώτα τον πολλαπλασιασμό
- 25, αν εκτελέσουμε πρώτα την πρόσθεση

Επίσης η αποτίμηση της παράστασης 8 - 3 - 1 δίνει τιμή

- 4, αν εκτελέσουμε πρώτα την αριστερή αφαίρεση
- 6, αν εκτελέσουμε πρώτα τη δεξιά αφαίρεση

Παραστάσεις

Χρησιμοποιούμε παρενθέσεις για να δηλώσουμε την επιθυμητή σειρά εφαρμογής των τελεστών κατά την αποτίμηση.

Μπορούμε να περιορίσουμε τη χρήση παρενθέσεων αν εισάγουμε κανόνες προτεραιότητας και προσεταιρισμού.

Παραστάσεις

Οι κανόνες προτεραιότητας ιεραρχούν τους τελεστές, έτσι ώστε ένας τελεστής με μεγαλύτερη προτεραιότητα να εφαρμόζεται πριν από τελεστές με μικρότερη προτεραιότητα.

Προτεραιότητα τελεστών στη Haskell (μικραίνει όσο κατεβαίνουμε):

not — (μοναδιαίοι)
^ **
* `div` `mod`
+ -
== /= <= >= > <
&&
||

Παραστάσεις

Για παράδειγμα στην στη Haskell η παράσταση

```
x+2*y >= 2*a + 3*b
```

είναι ισοδύναμη με την

```
(x+(2*y)) >= ((2*a) + (3*b))
```

Παραστάσεις

Οι κανόνες προσεταιρισμού καθορίζουν τη σειρά εφαρμογής τελεστών που έχουν την ίδια προτεραιότητα.

Ο συνηθισμένος κανόνας προσεταιρισμού είναι από τα αριστερά προς τα δεξιά.

Στη Haskell:

- οι τελεστές $*$ ‘div‘ ‘mod‘ $+$ $-$ && ||
  προσεταιρίζονται από αριστερά προς τα δεξια
- οι τελεστές ^ $**$
  προσεταιρίζονται από τα δεξιά προς τα αριστερά.

Παραστάσεις

Για παράδειγμα στη Haskell η παράσταση

```
x / y * z
```

είναι ισοδύναμη με την

```
(x / y) * z
```

ενώ η παράσταση

```
x ^ y ^ z
```

είναι ισοδύναμη με την

```
x ^ (y ^ z)
```

Παραστάσεις

Αν επιθυμούμε διαφορετική σειρά εφαρμογής των τελεστών από αυτή που προκύπτει με βάση τους κανόνες προτεραιότητας και προσεταιρισμού, χρησιμοποιούμε παρενθέσεις.

Παραστάσεις

Μπορούμε να κατασκευάσουμε ένα δέντρο που να περιγράφει τη δομή μιας παράστασης.

Για το σκοπό αυτό, πρώτα θα πρέπει να αρθούν όλες οι αμφισημίες, ώστε να είναι μονοσήμαντα ορισμένη η δομή της παράστασης, δηλαδή να είναι ξεκάθαρο ποιες υποπαραστάσεις αποτελούν τα ορίσματα του κάθε τελεστή.

Αυτό γίνεται με βάση του κανόνες προτεραιότητας και προσεταιρισμού: μπορούμε να προσθέσουμε κατάλληλα παρενθέσεις σε μία παράσταση έτσι ώστε τα ορίσματα κάθε τελεστή η συνάρτησης να είναι είτε απλές παραστάσεις (σταθερές ή μεταβλητές) είτε σύνθετες παραστάσεις κλεισμένες σε παρενθέσεις.

Παραστάσεις

Η παραπάνω μετατροπή μας βοηθάει στο να περιγράψουμε την παράσταση με ένα δέντρο:

- Μία απλή παράσταση (συμβολική ή κυριολεκτική σταθερά ή μεταβλητή) παριστάνεται από ένα δέντρο αποτελούμενο από έναν κόμβο με ετικέτα τη σταθερά ή τη μεταβλητή.
- Μία σύνθετη παράσταση παριστάνεται από ένα δέντρο η ρίζα του οποίου έχει ετικέτα το όνομα της συνάρτησης ή του τελεστή που θα δώσει το τελικό αποτέλεσμα, ενώ τα παιδιά του από αριστερά προς τα δεξιά είναι οι ρίζες των δέντρων που αντιστοιχούν στις παραστάσεις που αποτελούν τα ορίσματα της συνάρτησης ή του τελεστή.

Παραστάσεις

Παράδειγμα

Έστω η παράσταση

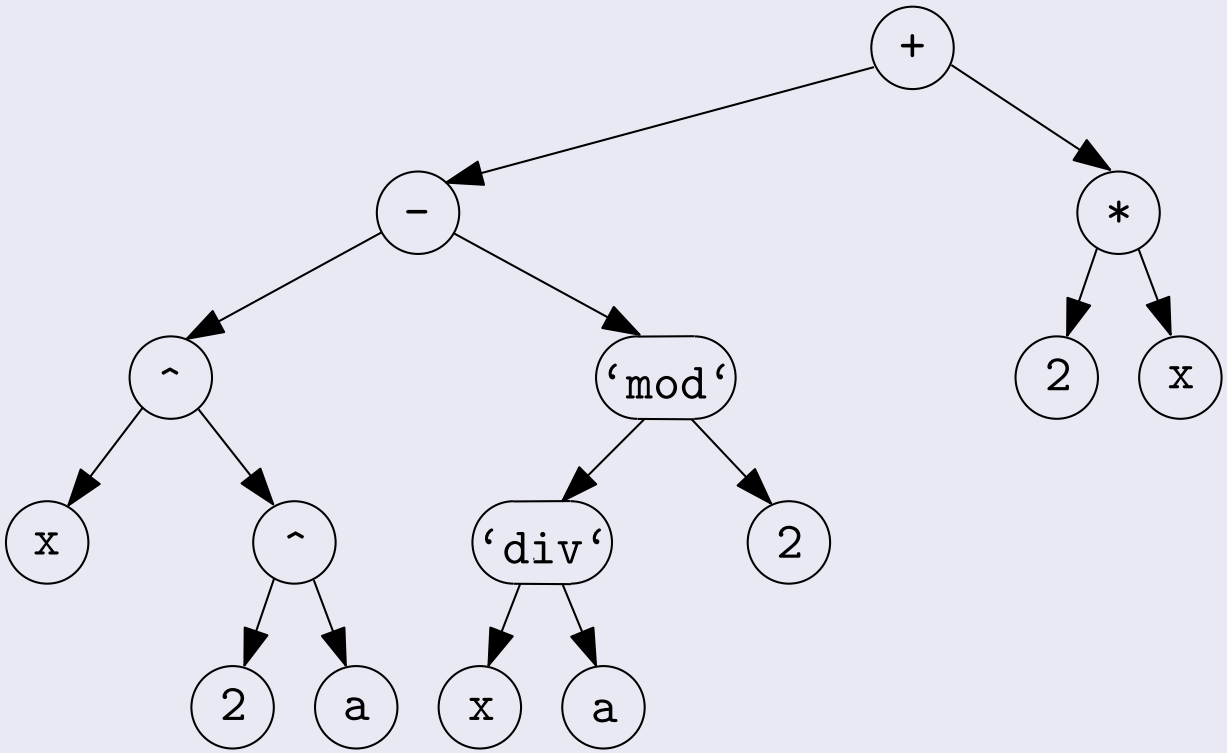
```
x^2^a-x‘div‘a‘mod‘2+2*x
```

Με βάση τους κανόνες προτεραιότητας των τελεστών, η παραπάνω παράσταση γράφεται ισοδύναμα

```
((x^(2^a))-((x‘div‘a)‘mod‘2))+(2*x)
```

Παραστάσεις

Το δέντρο που αντιστοιχεί στην παραπάνω παράσταση είναι:



Παραστάσεις

Παράδειγμα

Έστω η παράσταση

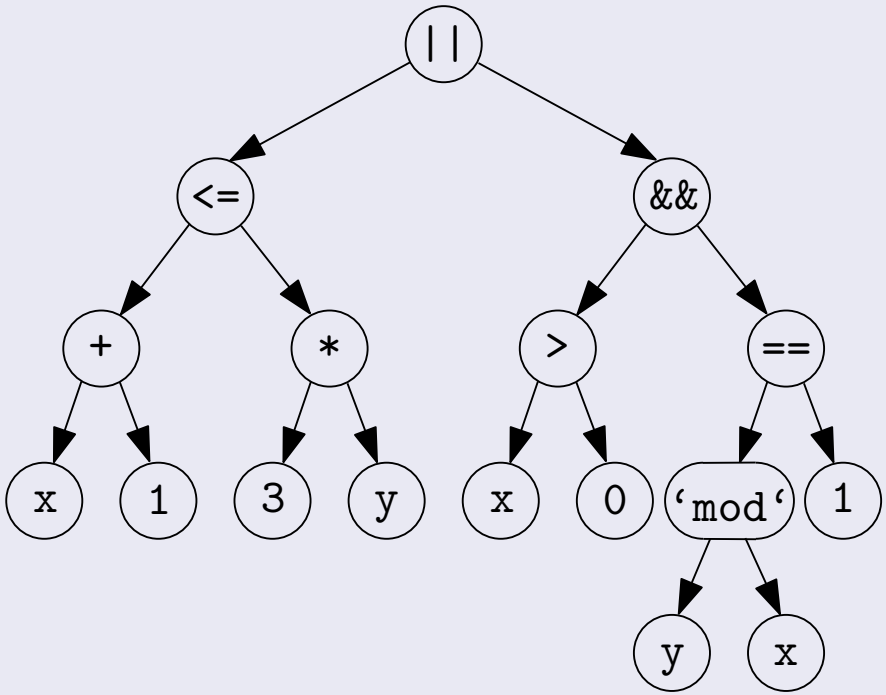
```
x+1<=3*y||x>0&&y`mod`x==1
```

Με βάση τους κανόνες προτεραιότητας των τελεστών, η παραπάνω παράσταση γράφεται ισοδύναμα

```
((x+1)<=(3*y))||((x>0)&&((y`mod`x)==1))
```

Παραστάσεις

Το δέντρο που αντιστοιχεί στην παραπάνω παράσταση είναι:



Παραστάσεις

Στις περισσότερες προστακτικές γλώσσες τα ορίσματα ενός τελεστή ή μίας συνάρτησης αποτιμούνται όλα με κάποια σειρά (συνήθως από τα αριστερά προς τα δεξιά) και στη συνέχεια εφαρμόζεται ο τελεστής ή καλείται η συνάρτηση.

Αυτό αντιστοιχεί με επεξεργασία του δέντρου που αντιστοιχεί στην παράσταση από τα φύλλα προς τη ρίζα.

Παραστάσεις

Σε ορισμένες περιπτώσεις ενδέχεται η αποτίμηση δύο παραστάσεων οι οποίες περιέχουν αριθμητικές πράξεις και είναι ισοδύναμες απο μαθηματική σκοπιά, να δίνει διαφορετικά αποτελέσματα. Αυτό οφείλεται στο ότι οι γλώσσες χρησιμοποιούν περιορισμένο πλήθος από bits για την αποθήκευση των αριθμών και ενδέχεται κάποια πράξη να προκαλέσει υπερχείλιση.

Παραστάσεις

Παράδειγμα

Έστω ο παρακάτω ορισμός σταθεράς στη Haskell:

```
twoBillions :: Int
twoBillions = 2000000000
```

Η αποτίμηση της παράστασης twoBillions `div` 10 + twoBillions `div` 10 δίνει την τιμή 400000000 (που είναι η αναμενόμενη) ενώ η αποτίμηση της παράστασης (twoBillions + twoBillions) `div` 10 (που είναι ισοδύναμη από μαθηματική σκοπιά με την πρώτη, καθώς όλες οι διαιρέσεις είναι ακριβείς) δίνει τιμή -29496730 (που προφανώς είναι λάθος).

Το πρόβλημα δημιουργείται επειδή η αποτίμηση του twoBillions + twoBillions προκαλεί αριθμητική υπερχείλιση.

Παραστάσεις

Σε ορισμένες περιπτώσεις η τιμή μίας παράστασης μπορεί να εξαχθεί πρίν ολοκληρωθεί ο απαιτούμενος υπολογισμός.

Για παράδειγμα αν η μεταβλητή x έχει τιμή 0, τότε η τιμή της παράστασης

(x == 0) or f(y,g(z))

είναι True, και αυτό προκύπτει χωρίς να απαιτείται να υπολογίσουμε την τιμή του f(y,g(z)).

Παραστάσεις

Στη στρατηγική της μερικής αποτίμησης, αποτιμάται πρώτα το ένα από τα δύο ορίσματα ενός τελεστή και στη συνέχεια εξετάζεται το αν μπορεί άμεσα να εξαχθεί το αποτέλεσμα της εφαρμογής του τελεστή, χωρίς να γίνει αποτιμησή του δεύτερου ορίσματος.

Μόνο στην περίπτωση που κάτι τέτοιο δεν είναι δυνατό, αποτιμάται και το δεύτερο όρισμα και στη συνέχεια εφαρμόζεται ο τελεστής.

Παραστάσεις

Πλεονεκτήματα μερικής αποτίμησης:

- Δεν ξοδεύει χρόνο για περιττούς υπολογισμούς.
- Δίνει μεγαλύτερη ευελιξία.

Παραστάσεις

Παράδειγμα

Έστω ότι εκτελείται η παρακάτω εντολή σε C και το n έχει τιμή 0:

```
if (n!=0 && m % n != 0) a = 1; else a = 2;
```

Επειδή η C κάνει μερική αποτίμηση και η συνθήκη n!=0 δεν αληθεύει, δεν θα επιχειρήσει να αποτιμήσει τη συνθήκη m % n != 0 και θα εκτελέσει την εντολή που ακολουθεί το else, δίνοντας στην a την τιμή 2.

Παραστάσεις

Αν τώρα εκετελεστεί η αντίστοιχη εντολή σε Pascal (και το n έχει τιμή 0):

```
if (n<>0) and (m mod n <> 0) then a := 1 else a := 2
```

τότε, επειδή η Pascal κάνει ολική αποτίμηση, θα επιχειρήσει να αποτιμήσει τη συνθήκη m mod n <> 0, η οποία λόγω της διαίρεσης με το 0 θα προκαλέσει σφάλμα χρόνου εκτέλεσης και διακοπή της λειτουργίας του προγράμματος.

Παραστάσεις

Για να δουλεύει σωστά το πρόγραμμα ακόμη και όταν το n έχει τιμή 0 θα πρέπει να μετασχηματίσουμε την εντολή χρησιμοποιώντας φωλιασμένα if:

```
if (n<>0) then
      if (m mod n <> 0) then a := 1
      else a := 2
else a := 2
```

Στην περίπτωση που το n έχει τιμή 0 η Pascal θα εκτελέσει το εξωτερικό else, αποφεύγοντας την αποτίμηση της συνθήκης m mod n <> 0, και θα δώσει στην a την τιμή 2.

Παραστάσεις

Παράδειγμα

Η χρησιμότητα της μερικής αποτίμησης είναι μεγαλύτερη όταν η συνθήκη χρησιμοποιείται σε έναν βρόχο while.

Το παρακάτω τμήμα προγράμματος C αναζητεί μία τιμή n σε μία συνδεδεμένη λίστα s:

```
p = s;
while (p != NULL && p->data != n) p = p->next;
```

Επειδή η C κάνει μερική αποτίμηση, το p→data θα αποτιμηθεί μόνο σε περίπτωση που ο δείκτης p δεν είναι κενός, δηλαδή μόνο όταν δείχνει σε κάποιο στοιχείο της λίστας. Αν η τιμή n δεν υπάρχει στη λίστα, τότε η εκτέλεση του βρόχου θα ολοκληρωθεί κανονικά, με το p να έχει τιμή NULL.

Παραστάσεις

Το αντίστοιχο τμήμα προγράμματος σε Pascal είναι το παρακάτω:

```
p := s;
while (p <> nil) and (p^.data <> n) do p := p^.next
```

Επειδή η Pascal κάνει ολική αποτίμηση, το p^.data θα αποτιμηθεί ακόμη και στην περίπτωση που ο δείκτης p είναι κενός, προκαλώντας σφάλμα χρόνου εκτέλεσης. Συνεπώς αν η τιμή n δεν υπάρχει στη λίστα, τότε η εκτέλεση του προγράμματος θα διακοπεί άμεσα.

Παραστάσεις

Για να μήν προκαλείται σφάλμα όταν το n δεν εμφανίζεται στη λίστα θα πρέπει να μετασχηματίσουμε την εντολή while' χρησιμοποιώντας μία βοηθητική λογική μεταβλητή:

```
p := s;
continue := true;
while continue do
    if p = nil then continue := false
    else if p^.data = n then continue := false
    else p := p^.next
```

Στην περίπτωση που ο δείκτης είναι κενός, η λογική μεταβλητή continue παίρνει τιμή false προκαλώντας τερματισμό του βρόχου, χωρίς να αποτιμηθεί το p^.data.

Παραστάσεις

Η μερική αποτίμιση εφαρμόζεται από πολλές γλώσσες (π.χ. C, Java, ...) για τους δυαδικούς λογικούς τελεστές (διάζευξη και σύζευξη).

Η Haskell χρησιμοποιεί την οκνηρή αποτίμηση, η οποία υπολογίζει μόνο τα ορίσματα που είναι απαραίτητα για να εξαχθεί η τιμή μίας παράστασης όχι μόνο για τους λογικούς τελεστές, αλλά και για όλες τις συναρτήσεις.

Στον αντίποδα, η Pascal χρησιμοποιεί ολική αποτίμηση για όλους τους τελεστές.

Παραστάσεις

Η αποτίμηση μίας παράστασης δεν πρέπει να δημιουργεί παρενέργειες όπως:

- Αλλαγή τιμών σε μεταβλητές.
- Είσοδο ή έξοδο δεδομένων.
- Δημιουργία ή καταστροφή αντικειμένων (δέσμευση ή αποδέσμευση μνήμης).

Παραστάσεις

Μία παράσταση είναι δυνατόν να προκαλεί παρενέργειες αν

- Κάποιος τελεστής προκαλεί παρενέργειες. Κατά κανόνα αυτό δε συμβαίνει υπάρχουν όμως ορισμένες εξαιρεσεις όπως οι τελεστές $++$, $--$, $+=$ κ.λ.π που υπάρχουν στη C και τη Java.
- Αν περιέχει κλήση μία συνάρτησης που προκαλεί παρενέργειες.

Αν η μία παράσταση περιέχει τελεστές ή συναρτήσεις που προκαλούν παρενέργειες, τότε το αποτέλεσμα της αποτίμησης μπορεί να εξαρτάται από τη σειρά με την οποία καλούνται οι συναρτήσεις και αποτιμούνται οι μεταβλητές.

Παραστάσεις

Παράδειγμα

```
program p(output);
var n:integer;

    function f(k:integer):integer;
    var x:integer;
        begin
           n:=n+1;
           f:=k+2
        end;

begin
n:=5;
n:=f(0)+n;
writeln(n);
end.
```

Παραστάσεις

Στο παραπάνω πρόγραμμα η κλήση της f αλλάζει την τιμή του n.

- Αν το n αποτιμηθεί πριν από την κλήση της f(n) τότε το πρόγραμμα τυπώνει τον αριθμό 7
- Αν το n αποτιμηθεί μετά από την κλήση της f(n) τότε το πρόγραμμα τυπώνει τον αριθμό 8

Παραστάσεις

Συμπεραίνουμε ότι οι γλώσσες που επιτρέπουν οι κλήσεις συναρτήσεων να προκαλούν παρενέργειες, για να έχουν σαφώς ορισμένη σημασιολογία, θα πρέπει να καθορίζεται η σειρά με την οποία αποτιμώνται τα ορίσματα των συναρτήσεων και των τελεστών. Η Java και η C# καθορίζουν ότι η αποτίμηση των ορισμάτων γίνεται από αριστερά προς τα δεξιά.

Ωστόσο, πολλές γλώσσες επιλέγουν να μην ορίσουν τη σειρά αποτίμησης των ορισμάτων, ώστε να είναι πιο ευέλικτος ο μεταφραστής στη φάση βελτιστοποίησης του κώδικα.