ΗΥ340 : ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

```
VAR i:Integer;
    FUNCTION(Symbol) replicate
        x = (function(x,y){return x+y;});
          class DelFunctor: public std::unary_function<
```

ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

ΗΥ340 : ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ

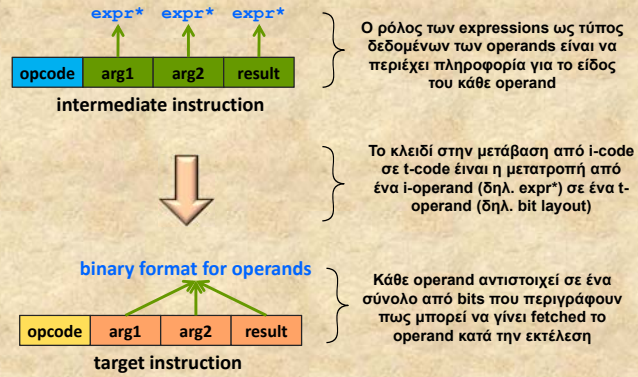
Διάλεξη 12η
ΠΕΡΙΒΑΛΛΟΝ ΕΚΤΕΛΕΣΗΣ - Ι

Περιεχόμενα

Αντιστοίχιση μεταβλητών σε θέση μνήμης (1/7)

- Στην παραγωγή ενδιάμεσου κώδικα, κάθε όρισμα τύπου έκφρασης *expr**, εκτός από τις σταθερές και συναρτήσεις, αντιστοιχεί πάντα σε κάποιο σύμβολο του πίνακα συμβόλων (πεδίο *sym* του *expr*)
  - Αυτό μπορεί να είναι είτε κρυφή η φανερή μεταβλητή
- *Ο τρόπος αντιστοίχησης ενός τέτοιου αναγνωριστικού ονόματος σε πραγματική θέση μνήμης ορίζεται από το περιβάλλον εκτέλεσης*
  - Το περιβάλλον εκτέλεσης είναι τα χαρακτηριστικά και οι συνθήκες του χώρου εκτέλεσης ενός προγράμματος τελικού κώδικα στη μηχανή στόχου
  - Κατά την παραγωγή τελικού κώδικα σε εντολές της μηχανής στόχου, στη θέση των συμβολικών ονομάτων που υπάρχουν στον ενδιάμεσο κώδικα θα πρέπει να υπάρχουν ορίσματα που προδιαγράφουν διευθύνσεις μνήμης

Ένθετο



Αντιστοίχιση μεταβλητών σε θέση μνήμης (2/7)

- Π.χ. έστω μία εντολή εκχώρησης *a=x+y*; η οποία σε ενδιάμεσο κώδικα παράγει τις δύο εντολές ως εξής:

```
ADD       x   y _t1
ASSIGN    _t1 a
```

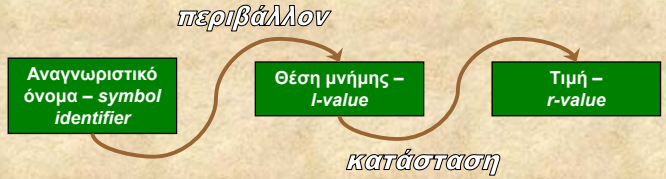
- Καθώς η μηχανή στόχου έχει πραγματική μνήμη και συγκεκριμένους κανόνες αναφοράς και σύνθεσης διευθύνσεων (*addressing*), πρέπει στην αντιστοίχιση των δύο αυτών εντολών σε εντολές μηχανής, αντί των ***a***, ***x***, ***y*** και ***_t1*** να υπάρχουν θέσεις μνήμης (απόλυτες ή σχετικές)
- Η πληροφορία που κρατείται για μεταβλητές στον πίνακα συμβόλων πρέπει να είναι επαρκής για την αντιστοίχηση τους σε θέσεις μνήμης
  - ***επομένως οι προδιαγραφές του περιβάλλοντος εκτέλεσης επηρεάζουν τις δομές του πίνακα συμβόλων***

Αντιστοίχιση μεταβλητών σε θέση μνήμης (3/7)

- Στις γλώσσες προγραμματισμού ξεχωρίζουμε μεταξύ των παρακάτω δύο εννοιών
  - *Περιβάλλον – environment*, που είναι συνάρτηση η οποία αντιστοιχεί αναγνωριστικά ονόματα σε θέσεις μνήμης
  - *Κατάσταση – state*, που είναι συνάρτηση η οποία αντιστοιχεί μία θέση μνήμης στην τιμή που αποθηκεύεται σε αυτή

Αντιστοίχιση μεταβλητών σε θέση μνήμης (4/7)

- Ένα περιβάλλον αντιστοιχεί έναν αναγνωριστικό όνομα σε ένα *l-value*
  - *Θέση αποθήκευσης – storage location*
- Μία κατάσταση αντιστοιχεί ένα l-value σε ένα *r-value*
  - *Τιμή - value*



Αντιστοίχιση μεταβλητών σε θέση μνήμης (5/7)

- Η κατάσταση μεταβάλλεται μόνο μέσω εντολών εκχώρησης (όλων των ειδών)
- Όταν το περιβάλλον αντιστοιχεί την θέση μνήμης ***m*** στο αναγνωριστικό όνομα ***a*** λέμε ότι
  - το ***a*** δεσμεύεται στο ***m*** – ***a*** *bound to* ***m***, ενώ η σχέση αυτή ονομάζεται αντιστοίχιση του ***a*** – *binding of* ***a***
- Η αντιστοίχηση των ονομάτων είναι η δυναμική πλευρά των δηλώσεων
  - Κάθε δηλωμένο όνομα θα πρέπει να προκαλέσει μία αντιστοίχιση, η οποία υφίσταται μόνο στο χώρο εμβέλειας της δήλωσης (λέγεται και χρόνος ζωής μίας αντιστοίχισης – *binding lifetime*)

Αντιστοίχιση μεταβλητών σε θέση μνήμης (6/7)

- Ο τρόπος που οργανώνεται η μνήμη και εφαρμόζεται η αντιστοίχηση εξαρτάται από τις παρακάτω ερωτήσεις
  - Υποστηρίζονται αναδρομικές κλήσεις;
  - Ποιος είναι ο χρόνος ζωής των τοπικών μεταβλητών μίας συνάρτησης;
  - Επιτρέπεται σε συναρτήσεις η χρήση μη τοπικών μεταβλητών;
  - Επιτρέπονται συναρτήσεις μέσα σε συναρτήσεις;
  - Επιτρέπεται πρόσβαση σε τοπικές μεταβλητές εξώτερων συναρτήσεων;
  - Πως γίνεται η τροφοδότηση πραγματικών παραμέτρων σε συναρτήσεις;
  - Οι συναρτήσεις είναι και τιμές;
  - Η διαχείριση μνήμης είναι αυτόματη η χειροκίνητη;
  - Οι μεταβλητές έχουν στατικούς τύπους ή δυναμικούς;

Αντιστοίχιση μεταβλητών σε θέση μνήμης (7/7)

```
1:    x=10;
2:    {     x=20;
3:          local x = 30;
4:          {     x = y = z = 40;
5:                local x = 50;
6:          }
7:    }
8:    x=60;
```

πληροφορία στον πίνακα συμβόλων

Όνομα	Γραμμή	Εμβέλεια	Offset
x	1	0	0
x	3	1	1
x	5	2	4
y	4	2	2
z	4	2	3
...	...	...	...

- Κάθε διαφορετική μεταβλητή του πίνακα συμβόλων πρέπει να αντιστοιχηθεί από το περιβάλλον σε διαφορετική θέση μνήμης
- Όλες οι παραπάνω μεταβλητές έχουν οριστεί σε καθολικό περιβάλλον εμβέλειας – global scope space και έχουν συνεχόμενα offsets
- Στην alpha θεωρούμε ότι όλες οι μεταβλητές, ανεξαρτήτως τύπου περιεχόμενης τιμής, έχουν τις ίδιες απαιτήσεις αποθήκευσης.
- Ο χώρος που χρειάζονται είναι σταθερός και μπορεί να περιέχει τιμή κάθε τύπου

Περιεχόμενα

Οργάνωση μνήμης προγράμματος (1/14)

- Θα προσπαθήσουμε να προσδιορίσουμε εξ αρχής έναν κατάλληλο τρόπο οργάνωσης της μνήμης
  - Θεωρώντας απλώς ότι το περιβάλλον εκτέλεσης μπορεί να προσφέρει στο πρόγραμμα μας ένα συνεχόμενο τμήμα μνήμης
  - Καθώς και κάποιους καταχωρητές που μπορούν να χρησιμοποιηθούν από το πρόγραμμά μας ελεύθερα (εμείς θα τους χρησιμοποιήσουμε για τη διαχείριση μνήμης)
- Μία τελευταία παραδοχή είναι ότι η μηχανή στόχου μπορεί να μας προσφέρει πρόσβαση στη μνήμη τόσο με απόλυτες όσο και με σχετικές διευθύνσεις, π.χ.
  - mem[*N*], *N σταθερά*
  - mem[**x**+*N*], **x** καταχωρητής, *N σταθερά*
  - mem[**x**+**y**], **x** καταχωρητής, **y** καταχωρητής

Οργάνωση μνήμης προγράμματος (2/14)

- *Υλοποίηση της environment συνάρτησης για μεταβλητές καθολικής εμβέλειας (1/2)*
  - Καθώς το πλήθος τους είναι ήδη γνωστό από την διαδικασία μεταγλώττισης και παραμένει αμετάβλητο
  - Καθώς κάθε μεταβλητή, ανεξαρτήτως τύπου δεδομένων κατά την εκτέλεση, καταλαμβάνει *μία θέση μνήμης* της εικονικής μηχανής alpha
  - Μπορούμε να χρησιμοποιήσουμε ένα συνεχόμενο κομμάτι μνήμης για τις μεταβλητές καθολικής εμβέλειας μεγέθους ίσου με το πλήθος των μεταβλητών αυτών

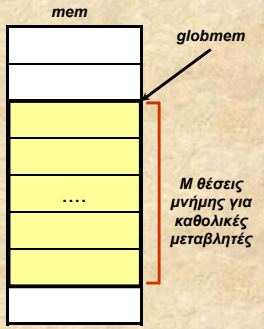
Οργάνωση μνήμης προγράμματος (3/14)

- *Υλοποίηση της environment συνάρτησης για μεταβλητές καθολικής εμβέλειας (2/2)*

Λαμβάνουμε ένα κομμάτι μνήμης μεγέθους ***M***, με ***M*** τον αριθμό των μεταβλητών καθολικής εμβέλειας. Χρησιμοποιούμε έναν καταχωρητή ***R*** της μηχανής τον οποίο θα τον αποκαλούμε συμβολικά ***globmem*** ο οποίος περιέχει πάντα την αρχική διεύθυνση ***A*** αυτού του τμήματος. Εάν η διεύθυνση ***A*** είναι 0, δεν χρησιμοποιούμε τον ***R***.

∀ μεταβλητή καθολικής εμβέλειας ***v***, με ***v*** σύμβολο από τον πίνακα συμβόλων:

environemt(*v*) : *mem*[***globmem*** + *v->offset*]



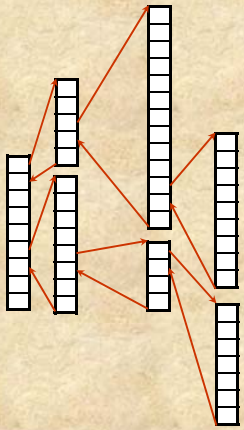
Οργάνωση μνήμης προγράμματος (4/14)

- *Τυπικά ορίσματα και τοπικές μεταβλητές (1/8)*
  - Στη γλώσσα *alpha*, η αντιστοίχιση μνήμης για τέτοιου είδους μεταβλητές έχει νόημα μόνο κατά την κλήση μίας συνάρτησης
    - δεν επιτρέπεται πρόσβαση στις τοπικές μεταβλητές και τα τυπικά ορίσματα μίας συνάρτησης παρά μόνο μέσα από εντολές της ίδιας της συνάρτησης
  - Τη στιγμή που μία συνάρτηση ***f*** καλείται, υπάρχει ενδεχόμενο πριν επιστρέψει να προκαλέσει άμεσα ή έμμεσα και άλλη κλήση της ***f*** ενώ η παρούσα κλήση είναι ήδη ενεργή
    - Αυτό θα φανεί καλύτερα αφού αμέσως ορίσουμε το *δέντρο ενεργοποίησης ή κλήσης συναρτήσεων*

Οργάνωση μνήμης προγράμματος (5/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (2/8)

- Η ροή ελέγχου είναι ακολουθιακή, δηλ. η εκτέλεση αποτελείται από ακολουθία βημάτων, με τον έλεγχο να βρίσκεται σε κάποιο συγκεκριμένο σημείο του προγράμματος σε κάθε βήμα.
  - Μπορούμε να θεωρήσουμε ως τέτοιου είδους βήματα τις εκτελέσιμες εντολές του ενδιάμεσου κώδικα.
- Η εκτέλεση μίας συνάρτησης αρχίζει από την πρώτη εντολή της και τελικά επιστρέφει ακριβώς στην εντολή που έπεται της εντολής κλήσεως.
- Κατά την κλήση μίας συνάρτησης, καθώς εκτελούνται οι εντολές της, άλλες συναρτήσεις μπορούν να κληθούν με τον ίδιο τρόπο.
- Έτσι, η ροή ελέγχου μεταξύ συναρτήσεων μπορεί να αναπαρασταθεί μέσω δέντρων, όπως θα δούμε σύντομα.



Οργάνωση μνήμης προγράμματος (6/14)

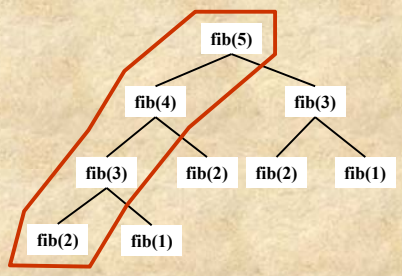
Τυπικά ορίσματα και τοπικές μεταβλητές (3/8)

- Κάθε εκτέλεση των εντολών μίας συνάρτησης ονομάζεται ενεργοποίηση μίας συνάρτησης - *activation*. Ο χρόνος ζωής μίας κλήσης - *call lifetime*, είναι ο χρόνος από τη στιγμή της κλήσης, έως και τη στιγμή που ο έλεγχος επιστρέφει στην εντολή αμέσως μετά τη κλήση.
- Για κάθε ενεργοποίηση μίας συνάρτησης και για τη διάρκεια ζωής της ενεργοποίησης αυτής, θα πρέπει να παρέχεται «*περιβάλλον*» για κάθε τυπικό όρισμα και τοπική μεταβλητή της συνάρτησης.
  - Είναι προφανές ότι εάν έχω την ίδια χρονική στιγμή πέραν της μίας ενεργοποιήσεις της ίδιας συνάρτησης, τότε κάθε μία θα πρέπει να έχει το δικό της ξεχωριστό «*περιβάλλον*»,
  - καθώς κάθε μία μπορεί να έχει τη δική της «*κατάσταση*», δηλ. διαφορετική αντιστοίχιση τιμών σε τυπικά ορίσματα ή τοπικές μεταβλητές.

Οργάνωση μνήμης προγράμματος (7/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (4/8)

```
function fib(n) {
  if (n == 1 || n == 2)
        return 1;
  else
        return fib(n-1) + fib(n-2);
}
```



•Ένα οποιοδήποτε μονοπάτι στο δέντρο ενεργοποίησης, ξεκινώντας από τη ρίζα, καταδεικνύει μία σειρά από ταυτόχρονα υπαρκτές ενεργοποιήσεις, με αυτή που βρίσκεται πιο κοντά στη ρίζα να έχει και τη μεγαλύτερη διάρκεια ζωής.

•Κάθε ενεργοποίηση απαιτεί ξεχωριστό περιβάλλον για την αποθήκευση του τυπικού ορίσματος *n*, καθώς όπως φαίνεται αυτό διατηρεί διαφορετικές τιμές (καταστάσεις) στην κάθε ενεργοποίηση κατά μήκος ενός τέτοιου μονοπατιού.

Οργάνωση μνήμης προγράμματος (8/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (5/8)

Το απαιτούμενο μέγεθος του «περιβάλλοντος», δηλ. ο συνολικός συνεχόμενος χώρος μνήμης, που απαιτείται για κάθε ενεργοποίηση με πραγματικά ορίσματα $x_1,...,x_n$ μίας συνάρτησης ***f*** είναι πάντα:

$$n + \text{αριθμός τοπικών μεταβλητών}(\boldsymbol{f})$$

Είναι κάτι τέτοιο προβλέψιμο κατά την διάρκεια μεταγλώττισης; Η απάντηση είναι «*εξαρτάται από τη σημασιολογία της γλώσσας*». Ενώ ο αριθμός των πραγματικών ορισμάτων είναι προβλέψιμος στη γλώσσα *alpha*, η συνάρτηση που πραγματικά καλείται ενδέχεται να προσδιορίζεται μόνο κατά την εκτέλεση. Το ίδιο ισχύει και στη C, C++ και Java.

Οργάνωση μνήμης προγράμματος (9/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (6/8)

Η λύση. Μόλις προσδιοριστεί η καλούμενη συνάρτηση, της δίνεται τόσος χώρος μνήμης ***M*** όσο *τουλάχιστον* το απαιτούμενο περιβάλλον της συνάρτησης. Όταν τελειώνει μία ενεργοποίηση (επιστρέφει η κλήση), τότε απελευθερώνεται αυτή η μνήμη.

- Η μνήμη αυτή χωρίζεται σε δυο τμήματα. Τα πραγματικά ορίσματα (τμήμα ***N***) και τις τοπικές μεταβλητές (τμήμα ***K***).
- Ταυτόχρονα, χρησιμοποιούμε δύο καταχωρητές, R1 και R2, τους οποίους τους δίνουμε τα συμβολικά ονόματα ***callmem*** και ***datamem*** όπως στο σχήμα.

Οργάνωση μνήμης προγράμματος (10/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (7/8)

∀ τυπική μεταβλητή ***x*** και τοπική μεταβλητή ***y*** της συνάρτησης, με ***x,y*** σύμβολα από τον πίνακα συμβόλων:

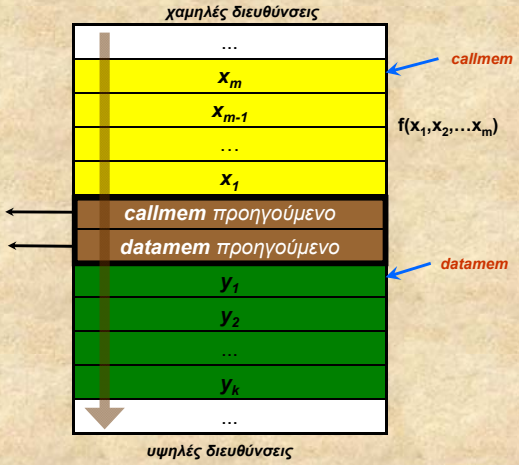
environemt(*x*) : *mem*[***datamem*** - *x->offset-1*]
environemt(*y*) : *mem*[***datamem*** + *y->offset*]

Ερώτημα: μετά το πέρας μίας ενεργοποίησης, επιστρέφουμε εν γένει στην προηγούμενη. Αυτό σημαίνει ότι οι τιμές των ***callmem*** και ***datamem*** καταχωρητών πρέπει να έχουν αναπροσαρμοστεί στην προηγούμενη ενεργοποίηση. Πως γίνεται κάτι τέτοιο?
Απάντηση: «σώζουμε» τις προηγούμενες τιμές των καταχωρητών στη μνήμη της νέας κλήσης, και τις επαναφέρουμε όταν η κλήση πρόκειται να επιστρέψει.

Βασικά δεν χρησιμοποιούμε τον *callmem* καθόλου για υπολογισμό διευθύνσεων, αλλά μόνο για διαχείριση μνήμης, κυρίως την προετοιμασία του χώρου μνήμης του περιβάλλοντος κλήσης της συνάρτησης.

Οργάνωση μνήμης προγράμματος (11/14)

Τυπικά ορίσματα και τοπικές μεταβλητές (8/8)



•Η μετατροπή αυτή αλλάζει ελαφρώς τον τρόπο υπολογισμού του περιβάλλοντος για τα τυπικά ορίσματα, καθώς πρέπει να υπολογίζεται και ο χώρος για τις τιμές των δύο σωμένων καταχωρητών.

•Αυτός ό ειδικός χώρος λέγεται, όπως αναμένεται, «σωμένο περιβάλλον προηγούμενης κλήσης» - *saved environment*

•Θα δούμε σε λίγο ότι εκτός από αυτούς τους καταχωρητές, θα πρέπει να σωθεί κάπου και ο αριθμός της εντολής του προγράμματος αμέσως μετά την εντολή κλήσης, ώστε να συνεχίσει η εκτέλεση από εκεί όταν περατωθεί η παρούσα κλήση.

Οργάνωση μνήμης προγράμματος (12/14)

- Ανακεφαλαιώνοντας τα προηγούμενα έχουμε τις εξής ανάγκες:
  - Μνήμης «εφ άπαξ» του περιβάλλοντος για τις μεταβλητές καθολικής εμβέλειας το οποίο παραμένει αμετάβλητου μεγέθους κατά τη διάρκεια της εκτέλεσης
  - Μνήμης δυναμικά για το περιβάλλον κάθε συνάρτησης και για κάθε ξεχωριστή ενεργοποίηση της
  - Θεωρούμε ότι τον κώδικα τον αποθηκεύουμε σε κάποιο χώρο μνήμης στον οποίο δεν επιτρέπεται πρόσβαση άλλη εκτός από ανάγνωση – *write protected*
- Ακολουθεί ο πιο συνηθισμένος τρόπος οργάνωσης της μνήμης, ώστε να καλύπτονται όλες αυτές οι ανάγκες

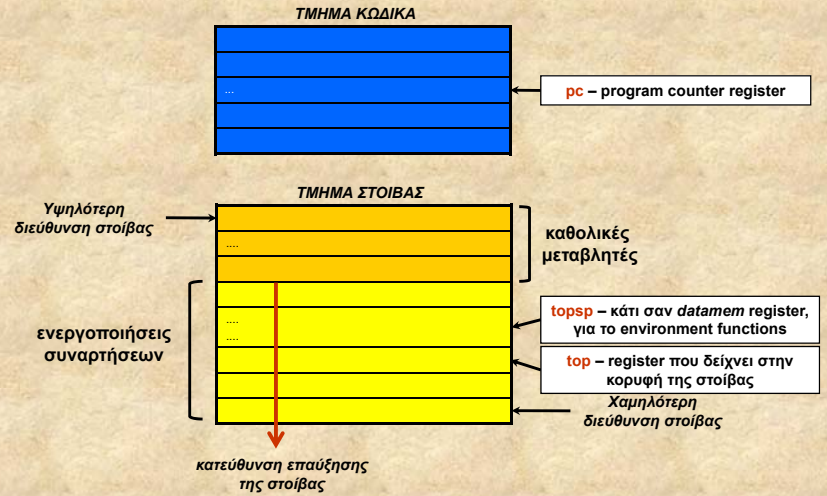
Οργάνωση μνήμης προγράμματος (13/14)

Τμήμα κώδικα προγράμματος (read only) – *code segment*
Τμήμα στατικών δεδομένων - *static data segment*
Στοίβα για περιβάλλοντα ενεργοποίησης συναρτήσεων – *stack segment*
↓ *Η στοίβα και ο σωρός «ανταγωνίζονται» για την ίδια μνήμη* ↑
Σωρός για δυναμική παραχώρηση μνήμης προγράμματος - *heap segment*

•Στη γλώσσα alpha, έχουμε ξεχωριστό heap segment το οποίο είναι «κρυφό» και δεν ανταγωνίζεται με τη στοίβα, ενώ δεν έχουμε ξεχωριστό static data segment, καθώς τα στατικά δεδομένα λαμβάνουν σταθερό αρχικό χώρο από τη στοίβα.

•Επίσης, το τμήμα κώδικα βρίσκεται πάντα σε ένα ξεχωριστό προστατευμένο τμήμα μνήμης.

Οργάνωση μνήμης προγράμματος (14/14)



Περιεχόμενα

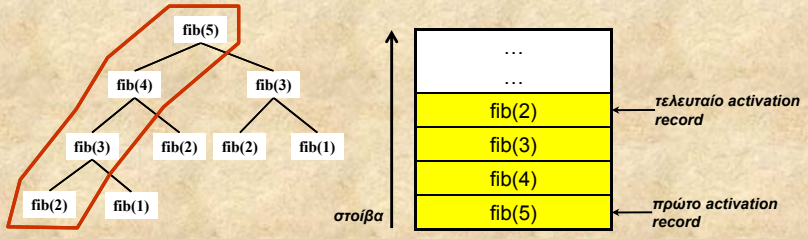
- Αντιστοίχιση μεταβλητών σε θέσεις μνήμης
- Οργάνωση μνήμης προγράμματος
- *Καταχωρήσεις ενεργών κλήσεων*
- Περιπτώσεις ληξιπρόθεσμων αναφορών μνήμης
- Από σύμβολα σε θέσεις μνήμης στην *alpha*

Καταχωρήσεις ενεργών κλήσεων (1/10)

- Το περιβάλλον που απαιτείται για την υποστήριξη της ενεργοποίησης μίας συνάρτησης περιέχει ουσιαστικά την κατάσταση της κλήσης αυτής
  - Ονομάζεται καταχώρηση ενεργής κλήσης – *activation record*, και υφίσταται καθόσον η κλήση είναι ενεργή
- Έχουμε πει ότι είναι σημαντικό να δρομολογείται η παραχώρηση μνήμης για κάθε τέτοια ενεργοποίηση δυναμικά, ενώ θα πρέπει να απελευθερώνεται η μνήμη ακριβώς με το πέρας κάθε τέτοιας κλήσης
- Ο συνήθης διαχειριστικός τρόπος είναι μέσω στοίβας, οπότε με κάθε νέα κλήση γίνεται *push* ένα νέο activation record, ενώ αυτό γίνεται *popped* ακριβώς όταν η κλήση επιστρέφει

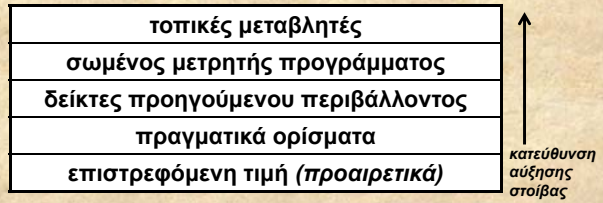
Καταχωρήσεις ενεργών κλήσεων (2/10)

- Αυτό σημαίνει ότι μεταγενέστερες κλήσεις θα είναι πάντα υψηλότερα στην στοίβα από ότι προγενέστερες,
- ενώ στην κορυφή θα βρίσκεται πάντα το activation record της συνάρτησης εκείνης στον κώδικα της οποίας βρίσκεται ο μετρητής προγράμματος – *program counter*
- Για κάθε δύο διαδοχικά activation records ***f***:r_k, ***g***:r_{k+1} της στοίβας, με ***k+1*** υψηλότερα του ***k***, η κλήση της ***g***:r_{k+1} έχει γίνει από την κλήση ***f***:r_k από εντολή κλήσης που περιέχεται στο σώμα της ***f***.



Καταχωρήσεις ενεργών κλήσεων (3/10)

- *Κάθε activation record συνήθως περιέχει τα εξής*:
  - Το περιβάλλον της κλήσης (πραγματικά ορίσματα και τοπικές μεταβλητές)
  - Τον μετρητή του προγράμματος τη στιγμή της κλήσης (δηλ. το ακριβές σημείο στο οποίο έγινε η κλήση)
  - Το σωμένο περιβάλλον της προηγούμενης κλήσης. Εδώ θα δούμε μία μικρή παραλλαγή της τεχνικής μας.
  - Την επιστρεφόμενη τιμή από την κλήση της συνάρτησης (εάν επιστρέφεται κάτι) – *υπάρχει και άλλη τεχνική*



Καταχωρήσεις ενεργών κλήσεων (4/10)

- Ο χώρος που απαιτείται στη στοίβα για κάθε activation record περιέχει ένα τμήμα που είναι προβλέψιμο κατά τη μεταγλώττιση καθώς και ένα άλλο το οποίο προσδιορίζεται μόνο κατά την κλήση (τα πραγματικά ορίσματα).
- Με κάθε νέα κλήση οι τοπικές μεταβλητές και τα ορίσματα αντιστοιχούν σε νέες περιοχές μνήμης, καθώς κάθε νέα καταχώρηση γίνεται *pushed* στην κορυφή της στοίβας,
- ενώ με το πέρας της κλήσης οι αντίστοιχες τιμές «σβήνονται» καθώς η μνήμη απελευθερώνεται
- Κατά την κλήση των συναρτήσεων ξεχωρίζουμε δύο πράγματα
  - την ευθύνη του κλητευτή (caller) και του καλούμενου (called)
  - τις ενέργειες για την κλήση και για την επιστροφή από κλήση

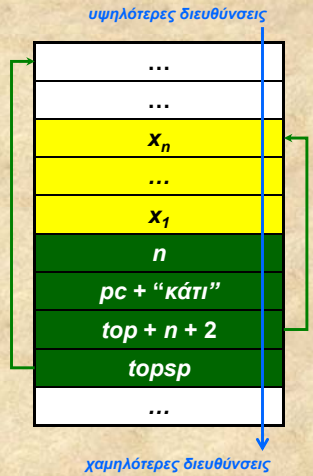
Καταχωρήσεις ενεργών κλήσεων (5/10)

Καταχωρήσεις ενεργών κλήσεων (6/10)

Ακολουθία κλήσης. Χρησιμοποιείται ο δείκτης *top*, ο οποίος σε κάθε push μειώνεται κατά 1 (θεωρούμε τη στοίβα να μεγαλώνει από υψηλές προς χαμηλές διευθύνσεις).

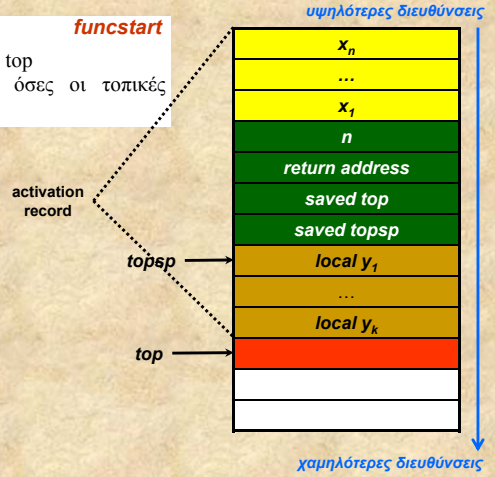
- Ενέργειες του κλητευτή
  - Αποτίμησε τα *n* πραγματικά ορίσματα ($x_1,...,x_n$)
  - Τοποθέτησε τα ορίσματα στη στοίβα με τη σειρά $x_n...x_1$
  - Τοποθέτησε στη στοίβα τον αριθμό των ορισμάτων *n*
  - Τοποθέτησε στη στοίβα τη διεύθυνση επιστροφής από κλήση
  - Τοποθέτησε στη στοίβα την τιμή του top πριν αρχίσει η ακολουθία κλήσης, δηλ. *top*+*n*+2 — **call**
  - Τοποθέτησε στη στοίβα την τιμή του *topsp*
  - Θέσε τον μετρητή προγράμματος στη διεύθυνση της συνάρτησης

Σημειώνουμε ότι κάθε αναφορά σε ενέργεια «τοποθέτησε στη στοίβα…» σημαίνει λειτουργία *push* με χρήση του δείκτη *top*. Π.χ., στο διπλανό σχήμα, *push* ένα actual argument *x* σημαίνει:
stack[top--] = x;



Καταχωρήσεις ενεργών κλήσεων (7/10)

- Ενέργειες του καλούμενου — **funcstart**
  - Θέσε το topsp ίσο με την τιμή του top
  - Ανέβασε το top τόσες θέσεις όσες οι τοπικές μεταβλητές



Καταχωρήσεις ενεργών κλήσεων (8/10)

Ακολουθία επιστροφής από κλήση. — **funcend**

- Αποθήκευσε την τιμή του *top* σε έναν καταχωρητή *r1*
- Επανέφερε το *top* στην προηγούμενη σωμένη τιμή
- Αποθήκευσε την διεύθυνση επιστροφής σε έναν καταχωρητή *r2*
- Επανέφερε την προηγούμενη διεύθυνση του *topsp*
- Καθάρισε την στοίβα από το *r1+1* έως και το *top*
- Θέσε *pc = r2*

Η ακολουθία εντολών επιστροφής από κλήση σχεδιάζεται έτσι ώστε να επιστρέψουμε στην εντολή ακριβώς μετά την κλήση, με *top* και *topsp* να δείχνουν στο activation record του κλητευτή.

Εναπόθεση επιστρεφόμενου αποτελέσματος.

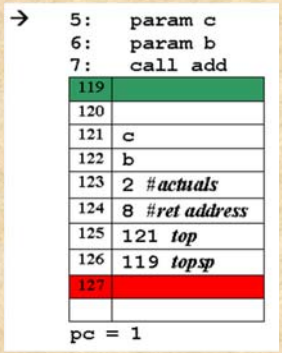
- Αποθήκευσε το αποτέλεσμα σε έναν ειδικό καταχωρητή *retval* η σε μία θέση στην στοίβα που κρατείται για το σκοπό αυτό ακριβώς πριν το πρώτο πραγματικό όρισμα.
- *Jump* στο τέλος της συνάρτησης — **return**

Καταχωρήσεις ενεργών κλήσεων (9/10)

```
function add(x,y) { return x+y; }
a = add(b, c);

1:   funcstart add
2:   add x y _t1
3:   return _t1
4:   funcend add
5:   param c
6:   param b
7:   call add
8:   getretval _t1
9:   assign _t1 a
```

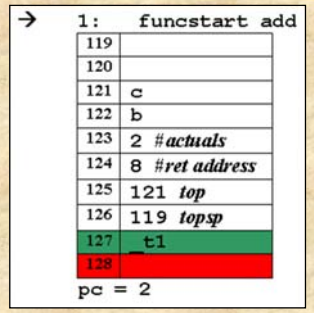
Για λόγους απλότητας στο παράδειγμα χρησιμοποιούμε στοίβα που αυξάνεται από χαμηλότερες προς υψηλότερες διευθύνσεις



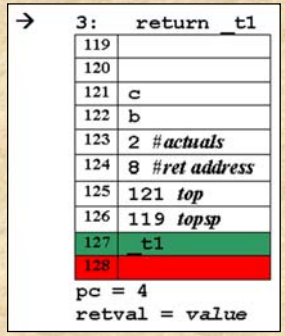
top
topsp

Υπολογισμός πραγματικών ορισμάτων, εναπόθεση των τιμών τους στη στοίβα και κλήση της συνάρτησης – όλα όσα έχουν συμβεί πριν «αναλάβει» η συνάρτηση.

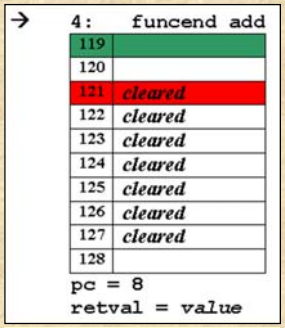
Καταχωρήσεις ενεργών κλήσεων (10/10)



Έναρξη κλήσης. Αναδιάταξη των *top* και *topsp* λαμβάνοντας υπόψη τις τοπικές μεταβλητές της συνάρτησης



Επιστροφή αποτελέσματος. Αποθήκευση τιμής στον *retval* register και jump στο τέλος της συνάρτησης



Επαναφορά των προηγούμενων *top* και *topsp*, καθάρισμα του activation record, επιστροφή στην σωμένη διεύθυνση.

Περιεχόμενα

- Αντιστοίχιση μεταβλητών σε θέσεις μνήμης
- Οργάνωση μνήμης προγράμματος
- Καταχωρήσεις ενεργών κλήσεων
- *Περιπτώσεις ληξιπρόθεσμων αναφορών μνήμης*
- Από σύμβολα σε θέσεις μνήμης στην *alpha*

Ληξιπρόθεσμες αναφορές μνήμης (1/3)

- Μία αναφορά σε θέση μνήμης ορίζεται ως ληξιπρόθεσμη – *dangling reference*
  - *εάν χρησιμοποιείται ως κάποια μεταβλητή **x** στη λογική του προγράμματος όταν ο χρόνος ζωής της μεταβλητής **x** έχει ήδη παρέλθει.*
- Οι ληξιπρόθεσμες αναφορές είναι πολύ επικίνδυνες καθώς δεν εντοπίζονται αυτόματα από όλες τις γλώσσες
- Θα δούμε αντιπροσωπευτικά παραδείγματα ληξιπρόθεσμων αναφορών και τα προβλήματα που προκαλούν, καθώς και το γιατί δεν έχουμε τέτοια φαινόμενα στην *alpha*
  - Είναι πολύ σημαντικό κατά τη σχεδίαση της γλώσσας να γνωρίζετε τι είδους ληξιπρόθεσμες αναφορές μπορεί να επιτρέπουν οι επιλογές σας

Ληξιπρόθεσμες αναφορές μνήμης (2/3)

Σε δυναμική παραχώρηση
- Χρήση μετά την απελευθέρωση μνήμης – *post-disposal overwrite*.
- Χρήση μετά την απελευθέρωση και παραχώρηση μνήμη – *recycled allocation overwrite*.

Σε τοπικές μεταβλητές
- Χρήση διεύθυνσης τοπικής μεταβλητής λήξαντος activation record – *popped local access*
- Χρήση μη-τοπικής μεταβλητής μη-ενεργούς εξώτερης συνάρτησης – *invalid non-local access*

```
int* x;
x  = (int*) malloc(4);
…
free(x);
char* s = (char*) malloc(4);
strcpy(s, "int");
…
*x = 10;
```

πιθανό recycled allocation overwrite

```
int* dangling (void) {
        int x;
        return &x;
}
```

popped local access

```
function f(x) {
        return (function(){ return x; });
}
```

invalid non-local access **– θα ίσχυε στην *alpha* εάν επιτρέπαμε πρόσβαση στις τοπικές μεταβλητές συναρτήσεων από άλλες συναρτήσεις που περιέχονται. Στο συγκεκριμένο παράδειγμα τι συμβαίνει στη γλώσσα *alpha*;**

Ληξιπρόθεσμες αναφορές μνήμης (3/3)

- Στη γλώσσα *alpha* ο τρόπος τροφοδότησης παραμέτρων σε κλήση γίνεται πάντα βάσει της τιμής και όχι της διεύθυνσης
  - Αυτή η τακτική είναι γνωστή ως *call-by-value* σε αντίθεση με το *call-by-reference*
  - Στην Pascal υποστηρίζονται και τα δύο.
  - Στη C υποστηρίζεται μόνο call-by-value, αλλά επειδή υπάρχουν δείκτες ο προγραμματιστής μπορεί να εξομοιώσει το call-by-reference
  - Στην *alpha* υφίσταται μόνο call-by-value. Δεν πρέπει να μας μπερδεύουν οι πίνακες των οποίων οι τιμές είναι εσωτερικές αναφορές σε δυναμικά δημιουργούμενους πίνακες.

Περιεχόμενα

- Αντιστοίχιση μεταβλητών σε θέσεις μνήμης
- Οργάνωση μνήμης προγράμματος
- Καταχωρήσεις ενεργών κλήσεων
- Περιπτώσεις ληξιπρόθεσμων αναφορών μνήμης
- *Από σύμβολα σε θέσεις μνήμης στην alpha*

Από σύμβολα σε θέσεις μνήμης στην *alpha* (1/2)

- Θα θεωρήσουμε στο περιβάλλον εκτέλεσης μία στοίβα μεγέθους ***N+1*** η οποία επεκτείνεται από υψηλότερες προς χαμηλότερες διευθύνσεις.
  - Έστω ότι η υψηλότερη διεύθυνση είναι η ***stack[N]*** και η χαμηλότερη ***stack[0]***
- Θα θεωρήσουμε την δομή για activation records που έχουμε παρουσιάσει πρωτύτερα.
- Θα θεωρήσουμε αποθήκευση των μεταβλητών καθολικής εμβέλειας στις πρώτες ***M*** θέσεις της στοίβας.
  - Προφανώς από τη θέση ***stack[N-M]*** και «κάτω» μπορούμε να έχουμε activation records.
- Έστω ***K*** το μέγεθος του τμήματος στη στοίβα ακριβώς μετά τα πραγματικά ορίσματα και πριν τις τοπικές μεταβλητές για την αποθήκευση περιβάλλοντος

Από σύμβολα σε θέσεις μνήμης στην *alpha* (2/2)

- Για σύμβολο *sym* που είναι τοπική μεταβλητή σε συνάρτηση, η αντιστοίχηση σε διεύθυνση μνήμης είναι:
  - `stack[topsp - sym->offset]`
- Για σύμβολο *sym* που είναι τυπικό όρισμα σε συνάρτηση, η αντιστοίχηση σε διεύθυνση μνήμης είναι:
  - `stack[topsp + K + sym->offset + 1]`
- Για σύμβολο *sym* που είναι μεταβλητή καθολική εμβέλειας, η αντιστοίχηση σε διεύθυνση μνήμης είναι:
  - `stack[N - sym->offset]`