HY255 Εργαστήριο Λογισμικού - L04

Αποκωδικοποίηση Δηλώσεων στη C

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Δηλώσεις μεταβλητών

- Η δήλωση είναι ένα όνομα για μια μεταβλητή και ένας τύπος
- Είναι απαραίτητες;
  - Στη C ναι, αλλά σε άλλες γλώσσες όχι (python, scripting languages)
- Γιατί είναι χρήσιμες (και γιατί υπάρχουν);
  - 1. Μας επιτρέπουν να εκφράσουμε το πρόγραμμά μας με ακρίβεια (κατανόηση του προγράμματος από μας)
  - 2. Καθορίζουν την συμπεριφορά των μεταβλητών κατά την εκτέλεση του προγράμματος (ορθότητα που ελέγχεται από τον compiler)
  - 3. Καθορίζουν την μορφή (layout) των δεδομένων μας στη μνήμη (απόδοση κατά την εκτέλεση)
- Και τα (3) αυτά θέματα είναι εξαιρετικά σημαντικά για τον σχεδιασμό και την υλοποίηση συστημάτων και εφαρμογών που να δουλεύουν σωστά υπό όλες τις συνθήκες
  - Οπότε θα παρατηρήσετε ότι οι γλώσσες που χρησιμοποιούνται για την υλοποίηση λογισμικού συστημάτων και εφαρμογών χρησιμοποιούν δηλώσεις μεταβλητών
  - Γλώσσες που είναι πιο κοντά σε scripting και επεξεργασία δεδομένων/μοντέλων σε συγκεκριμένες περιοχές
  - Και τα δύο αυτά έχουν νόημα, για διαφορετικούς σκοπούς

“Πολύπλοκες” δηλώσεις μεταβλητών

- Γιατί να υπάρχουν δηλώσεις όπως
  - enum {...} (*actions[N])(char c);
- Πρώτα από όλα, δεν χρειάζεται όλες ή πολλές δηλώσεις να έχουν πολύπλοκη δομή - simple is good.
- Οι δηλώσεις αποτελούν σημαντικό μέρος της περιγραφής του πρόγραμματός μας
- Η εκφραστική τους δύναμη δίνει σε εμάς την δυνατότητα να εκφράσουμε με αποδοτικό τρόπο και με ακρίβεια αυτό που θέλουμε να κάνει το πρόγραμμά μας
  - Π.χ. πως περιγράφουμε ένα σύνολο από συναρτήσεις που μπορούμε να επιλέξουμε ποια συνάρτηση θα εκτελέσουμε με βάση μια άλλη μεταβλητή/τιμή;
  - Π.χ. πως περιγράφουμε ένα πακέτο δικτύου που θα περιέχει strings μεταβλητού και άγνωστου μεγέθους τα οποία μπορούν να γίνουν update και να βρίσκονται σε μη συνεχόμενες θέσεις μνήμης του προγράμματος;
- Όταν χρειαστεί να περιγράψουμε ή να κατανοήσουμε τέτοιες περιγραφές, ενδέχεται να χρειαστούμε τύπους πέρα από τους βασικούς.

Αποκωδικοποίηση δηλώσεων στη C
(Παλίνδρομο από την υλοποίηση του compiler για τις δηλώσεις)

1. Πήγαινε στο “leftmost” όνομα (identifier)	→ πες: ο ”identifier” είναι ...
2. Κοίταξε το σύμβολο στα δεξιά (2a) αν είναι [	→ για κάθε ζεύγος πες: array (n στοιχείων) of … • Το n είναι η τιμή μέσα στο αντίστοιχο ζεύγος []
(2b) ή αν είναι (	→ διάβασε μέχρι την αντίστοιχη) και πες: function returning
3. Αν το σύμβολο στα αριστερά είναι (	→ διάβασε μέχρι την αντίστοιχη) **και πήγαινε στο βήμα 2**
4. Αν το σύμβολο στα αριστερά είναι: * const volatile (⌘)	→ συνέχισε να διαβάζεις μέχρι να μην είναι κάποιο από αυτά τα τρία και • για *, πες: pointer to ... • για const, πες: read-only ... • για volatile, πες: volatile ... (⌘) **και πήγαινε στο βήμα 3**
5. Τα υπόλοιπα σύμβολα (tokens) είναι ο βασικός τύπος της δήλωσης	→ διάβασε τα υπόλοιπα σύμβολα, π.χ. unsigned int
(⌘) Το volatile θα απασχολήσει μόνο όσους αργότερα ενδιαφερθούν για παραλληλισμό	

Παραδείγματα

- int i;
  - 1: Η μεταβλητή i είναι
  - 2: -
  - 3: -
  - 4: -
  - 5: int
- int const i;
  - 1: Η μεταβλητή i είναι
  - 2: -
  - 3: -
  - 4: read-only
  - 3: -
  - 4: -
  - 5: int
- int const *pi;
  - 1: Η μεταβλητή pi είναι
  - 2: -
  - 3: -
  - 4: pointer σε read-only
  - 3: -
  - 4: -
  - 5: int
- int *const pi;
  - 1: Η μεταβλητή pi είναι
  - 2: -
  - 3: -
  - 4: read-only pointer σε
  - 3: -
  - 4: -
  - 5: int

Παραδείγματα - δοκιμάστε

- int A[10];
- int *(A[10]);
- int (*A)[10];
- int (B[10])[20];
- unsigned int C[10][20][30];
- char const * (*pf(int))[10];
- char *const *(*h)(int, char *);
- struct {int x; int y;} *s[10];
- struct {int x; int y;} (*s)[10];

Στη C...

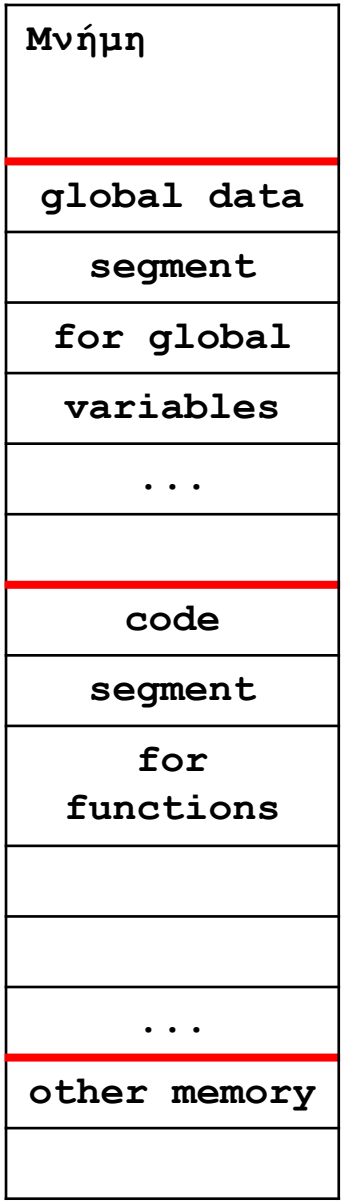
- Μπορούμε να δηλώσουμε μεταβλητές οποιουδήποτε τύπου εκτός από:
- Συναρτήσεις που επιστρέφουν μια άλλη συνάρτηση (π.χ. κάτι σαν f()())
  - ΑΛΛΑ: μια συνάρτηση μπορεί να επιστρέψει pointer σε συνάρτηση (π.χ. int (*f())();)
- Συναρτήσεις που επιστρέφουν ένα array (π.χ. f()[])
  - ΑΛΛΑ: μια συνάρτηση μπορεί να επιστρέψει pointer σε array (π.χ. int (*f())[];)
- Arrays που περιέχουν συναρτήσεις (π.χ. f[]())
  - ΑΛΛΑ: ένα array μπορεί να περιέχει pointers σε συναρτήσεις (π.χ. int (*f[])();)
- Σημ1: Ένα array μπορεί να περιέχει arrays (π.χ. int A[][])
- Σημ2: Δεν έχουμε αναφερθεί ακόμη καθόλου σε δυναμική μνήμη και μπορούμε ήδη να γράφουμε “σημαντικά” προγράμματα

Layout μεταβλητών και δηλώσεων στη μνήμη

- Η δήλωση κάθε μεταβλητής (αντικειμένου) καθορίζει και το layout που έχει στη μνήμη η μεταβλητή
- Ποια είναι η εικόνα που έχουν στη μνήμη μεταβλητές (αντικείμενα) που έχουν δηλωθεί ως, π.χ.:
  - int A[10][2];
  - int *(B[10]);
  - int **C;
  - struct {int x; int y;} *(D[10]);
  - struct {int x; int y;} (*E)[10];
  - struct {int size; int A[100]) s1;
  - struct {int size; int *A;} s2;

Δομή ενός Προγράμματος

- Στη C όλα τα προγράμματα ακολουθούν την εξής δομή, από τρία μέρη:
  - 1. <C pre-processor directives>
  - …
  - 2. <Δηλώσεις μεταβλητών, τύπων etc.>
  - …
  - 3. <Συναρτήσεις με κώδικα> -- Κάθε συνάρτηση αποτελείται από
  - <Τοπικές Δηλώσεις>
  - <Κώδικα της συνάρτησης>
  - …
- Ένα πρόγραμμα, όταν αρχίζει να εκτελείται έχει στην μνήμη δομή που αντιστοιχεί σε αυτή τη δομή του κώδικα
  - Global data segment, όπου τοποθετούνται όλες οι global μεταβλητές
    - Κάθε δήλωση global μεταβλητής αντιστοιχεί σε κάποια/κάποιες θέσεις σε αυτό το μέρος της μνήμης
  - Code segment όπου τοποθετείται ο εκτελέσιμος κώδικας όλων των συναρτήσεων
    - Κάθε συνάρτηση αφού μεταφραστεί τοποθετείται σε αυτό το μέρος της μνήμης
  - Η μνήμη του προγράμματος περιέχει και άλλα segments/πληροφορίες, π.χ. για τις τοπικές μεταβλητές στη στοίβα, που θα τα δούμε αργότερα
    - Μπορούμε να έχουμε προγράμματα χωρίς τοπικές μεταβλητές; Βεβαίως – αλλά ο στόχος είναι να χρησιμοποιούμε τοπικές μεταβλητές όσο μπορούμε γιατί οδηγούν σε καλύτερα σχεδιασμένα προγράμματα. Μάλιστα, η πρόκληση είναι συνήθως το αντίθετο, δηλαδή να σχεδιάζουμε προγράμματα με όσο το δυνατόν λιγότερες ή και καθόλου global μεταβλητές. Περισσότερα για αυτά αργότερα.
- Τα περιεχόμενα του global και του code segment δημιουργούνται όταν το πρόγραμμα μεταφράζεται
- Επίσης, το μέγεθός τους είναι (global data, code) είναι προκαθορισμένο και γνωστό κατά την μετάφραση του προγράμματος, πριν την εκτέλεση



Reading

- King 18.{1,3,4,5}
- If interested, check Chapter 3 (Unscrambling Declarations in C) of Expert C Programming: Deep C Secrets. Peter van der Linden. 1st Edition, Prentice Hall, 1994.