HY255 Εργαστήριο Λογισμικού – L03

Pointers, Strings, Arrays

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Τι είναι ο τύπος “pointer σε …”

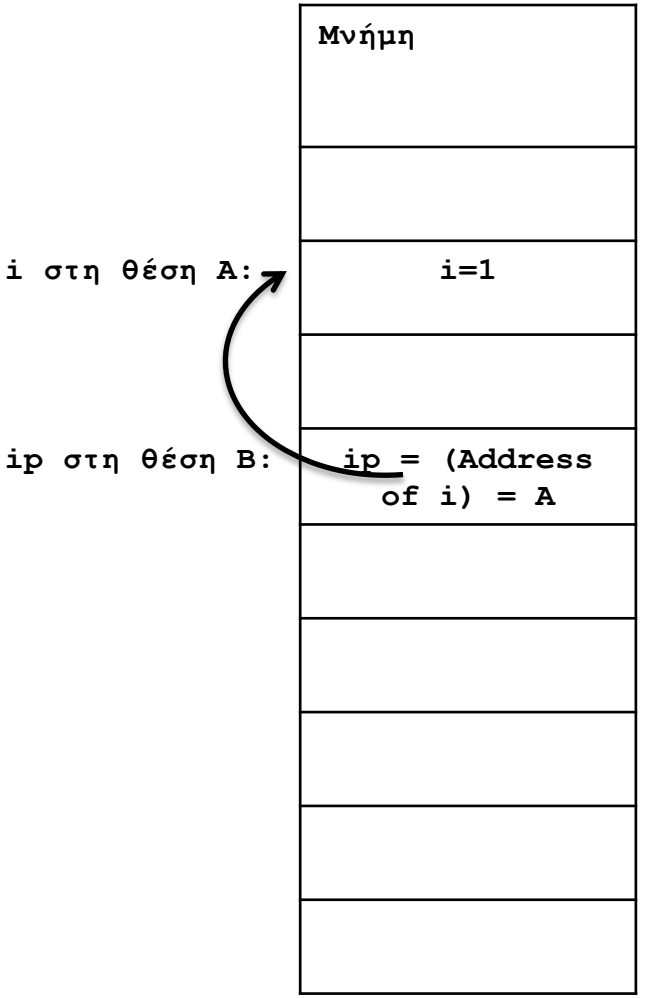
- Ένας τύπος σαν όλους τους άλλους…
- Από την πλευρά της γλώσσας:
  - Είναι ένας τύπος που μας επιτρέπει να δείχνουμε σε άλλα ”αντικείμενα” (π.χ. μεταβλητές, τιμές).
- Από την πλευρά του συστήματος:
  - Είναι ένας τύπος που μας επιτρέπει να δείχνουμε σε θέσεις μνήμης (αφού οι μεταβλητές είναι ονόματα για θέσεις μνήμης).
- Για να δηλώσουμε μια μεταβλητή τύπου pointer (όπως έχετε δει) χρειαζόμαστε τα περιβόητα ”*”.
  - Π.χ. char *cp;
  - Το cp είναι μια μεταβλητή τύπου pointer σε άλλες μεταβλητές/τιμές τύπου char.
- Τα περιεχόμενα μιας μεταβλητής τύπου pointer τα ονομάζουμε ”διεύθυνση/address”.

Τι ορίζει ο τύπος pointer

- (1) Το μέγεθος των αντίστοιχων μεταβλητών (όπως όλοι οι τύποι): Το μέγεθος μια μεταβλητής τύπου pointer είναι (συνήθως):
  - 32 bits (4 bytes) σε ένα σύστημα που υποστηρίζει 32-bit address spaces (4GB)
  - 64 bits (8 bytes) σε ένα σύστημα που υποστηρίζει 64-bit address spaces (16HexaB)
  - (Εμείς στις ασκήσεις μας δουλεύουμε με 32-bit address spaces, αν και σε πολλές από αυτές δεν παίζει ρόλο, επειδή η γλώσσα μας παρουσιάζει μια αφαίρεση του συστήματος).
- (2) Τις πράξεις που μπορούμε να κάνουμε σε κάθε μεταβλητή του τύπου “pointer σε…”. Ο τύπος “pointer σε…” υποστηρίζει 2 νέες πράξεις:
  - (1) & → Ο τελεστής αυτός δημιουργεί μια τιμή τύπου “pointer σε…“
  - (2) * → Ο τελεστής αυτός επιστρέφει τα περιεχόμενα της διεύθυνσης που περιέχονται σε μια μεταβλητή τύπου ”pointer σε...” (dereferencing)
  - Αν δεν είχαμε αυτούς τους τελεστές, δεν θα μπορούσαμε ούτε να δημιουργήσουμε αλλά ούτε και να χειριστούμε τιμές του τύπου ”pointer σε” (addresses)
  - (3) Assignment, π.χ. ip = &i;
  - (4) αριθμητική, π.χ. *(ip + 1) = 3;
  - (5) σύγκριση, π.χ. if (ip == &i) ...
- Μπορούμε να έχουμε μεταβλητές τύπου “pointer σε…” σε οποιοδήποτε τύπο (βασικό ή δομημένο και επίσης σε άλλο τύπο pointer)
- Ο τύπος μιας μεταβλητής pointer περιέχει και τον τύπο του αντικειμένου που δείχνει, πχ “pointer σε char” και όχι “pointer”. Οπότε μια μεταβλητή τύπου pointer σε int έχει διαφορετικό τύπο από μια μεταβλητή pointer σε float.

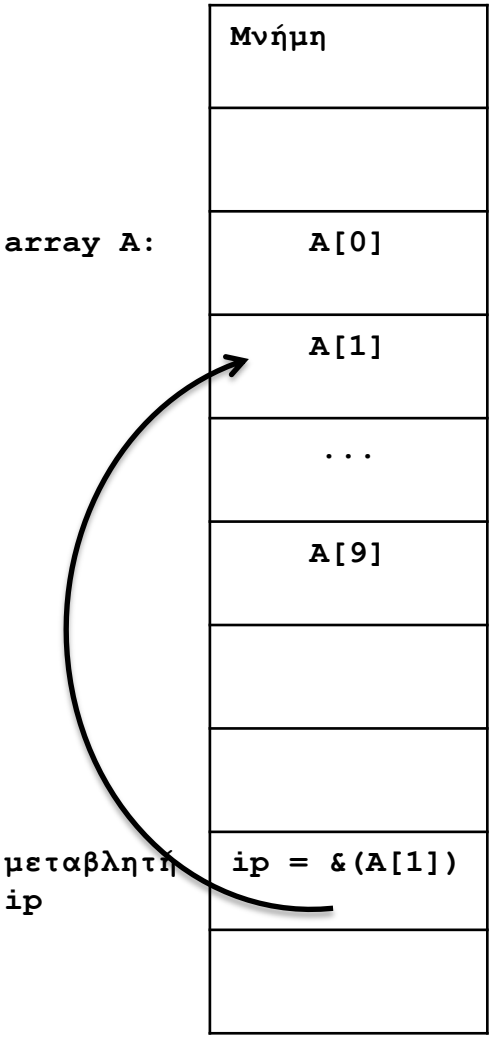
Παράδειγμα

- Δηλώσεις μεταβλητών
- Μια μεταβλητή τύπου int
  - int i=1;
- Μια μεταβλητή τύπου pointer σε int
  - int *ip;
- Δημιουργούμε μια τιμή τύπου address (&i) και την αναθέτουμε στην μεταβλητή ip
  - ip = &i;
- Αναθέτουμε τον int 5 στο αντικείμενο στο οποίο δείχνει η μεταβλητή ip, και το οποίο αντικείμενο είναι τύπου int (αφού το ip είναι τύπου pointer σε int)
  - *ip = 5;
- Αναθέτουμε στο i την τιμή (int) στην οποία δείχνει η μεταβλητή ip "pointer σε int"
  - i = *ip;
- Όταν γράφουμε το ip αλλάζουμε την θέση B (ip)
- Όταν γράφουμε το *ip αλλάζουμε την θέση A (i)
- sizeof(i) = sizeof (*ip) = sizeof (int) → συνήθως 4
- sizeof (ip) = sizeof(&i) = sizeof (int *) → συνήθως 4 (για μας)



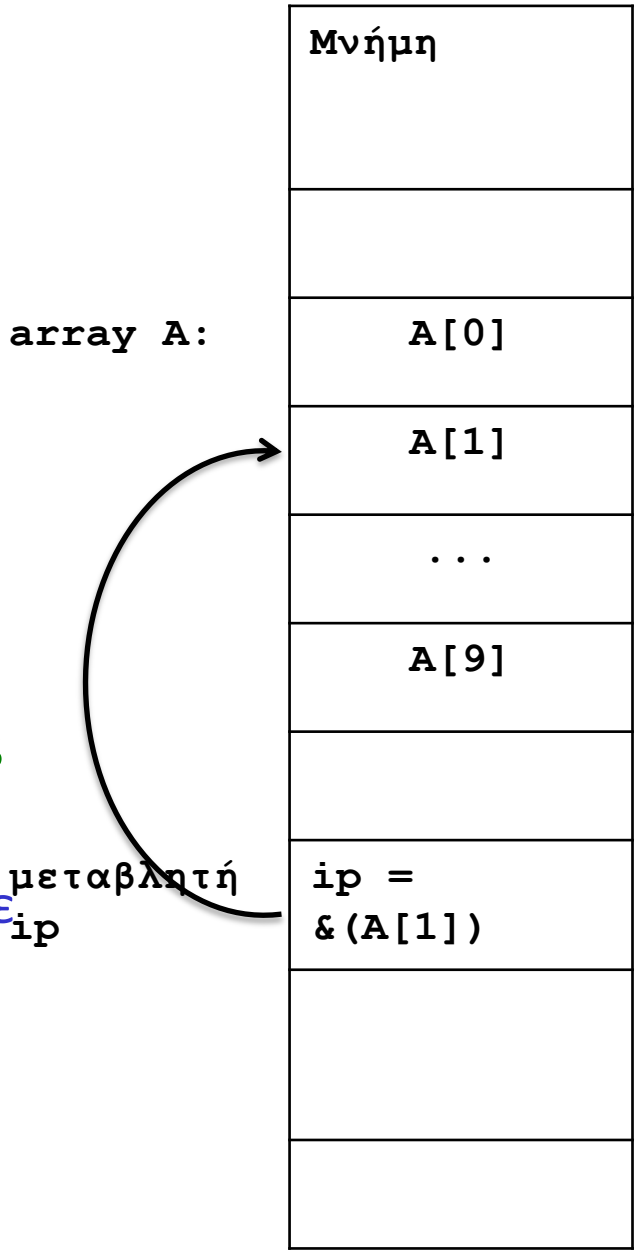
Ποια η σχέση των arrays με pointers?

- Έχουμε ακούσει/δει ότι τα arrays έχουν στη C κάποια σχέση με pointers, π.χ.:
  - int A[10];
  - int *ip;
  - ip = &(A[0]);
  - Τέτοιες σχέσεις προκύπτουν από τον ορισμό των τύπων
- Τι και γιατί επιτρέπεται στην γλώσσα C?
- **Ορισμός:** Το όνομα ενός array στη C είναι μια σταθερή τιμή τύπου pointer (σε τύπο ίδιο με τον τύπο των στοιχείων του array)
  - Αφού ισχύει αυτό επιτρέπεται πλέον να πούμε
  - ip = A; # το A είναι τιμή τύπου pointer σε int
  - ip = A+1 # αφού οι pointers επιτρέπουν arithmetic ops
  - if (A < ip) # αφού οι pointers επιτρέπουν συγκρίσεις
  - Αλλά και
  - *A = 0; *(A+2) = 0; x = (A+2)[5], κ.ο.κ.
- Επιπλέον, εφόσον το όνομα ενός array είναι ισοδύναμο με μια τιμή τύπου "pointer σε", τότε μια οποιαδήποτε μεταβλητή τύπου "pointer σε" μπορεί να χρησιμοποιηθεί σαν όνομα array, άρα:
  - ip[0] = 0;
  - (ip + 1)[0] = 0;



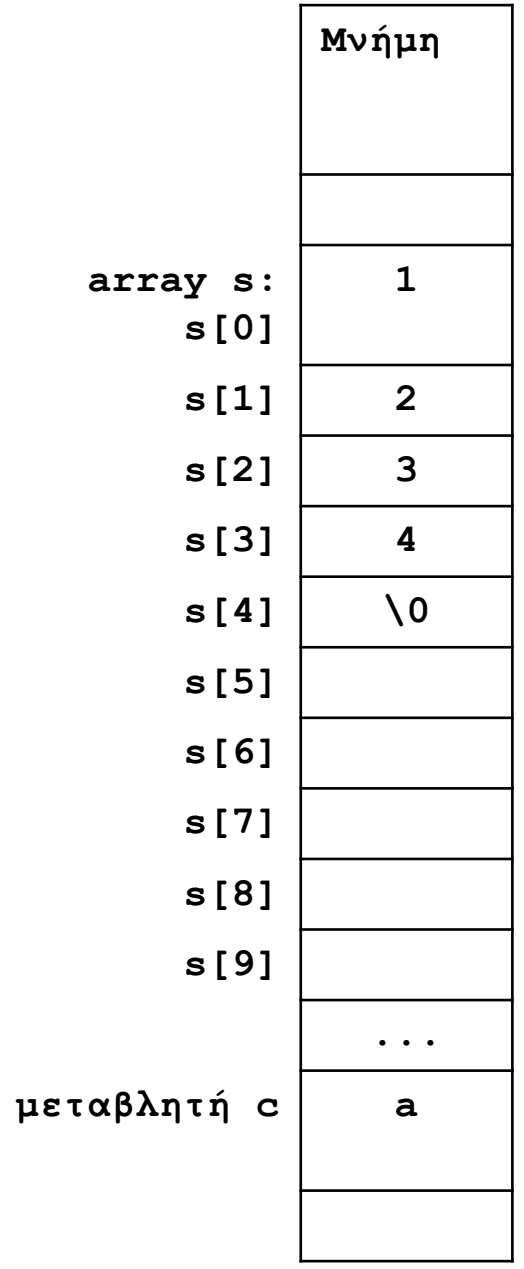
Ωστόσο:

- Ένα array είναι διαφορετικός τύπος από τον τύπο pointer σε
- Π.χ. Το array είναι μια μεταβλητή που έχει τόσο χώρο στη μνήμη όσο όλα τα στοιχεία του και επιτρέπει συγκεκριμένες πράξεις
- Ένας pointer είναι π.χ. 4 bytes και επιτρέπει διαφορετικές πράξεις
- Επίσης ο υπολογισμός των εκφράσεων κατά την εκτέλεση του προγράμματος γίνεται διαφορετικά όταν περιέχουν ονόματα arrays ή pointers, π.χ.
  - A[i]: πρόσθεσε το i στο A (σταθερή τιμή) και φέρε τα περιεχόμενα της θέσης που προκύπτει
  - ip[i]: φέρε τα περιεχόμενα της μεταβλητής ip, πρόσθεσε το i, φέρε τα περιεχόμενα της θέσης που προκύπτει



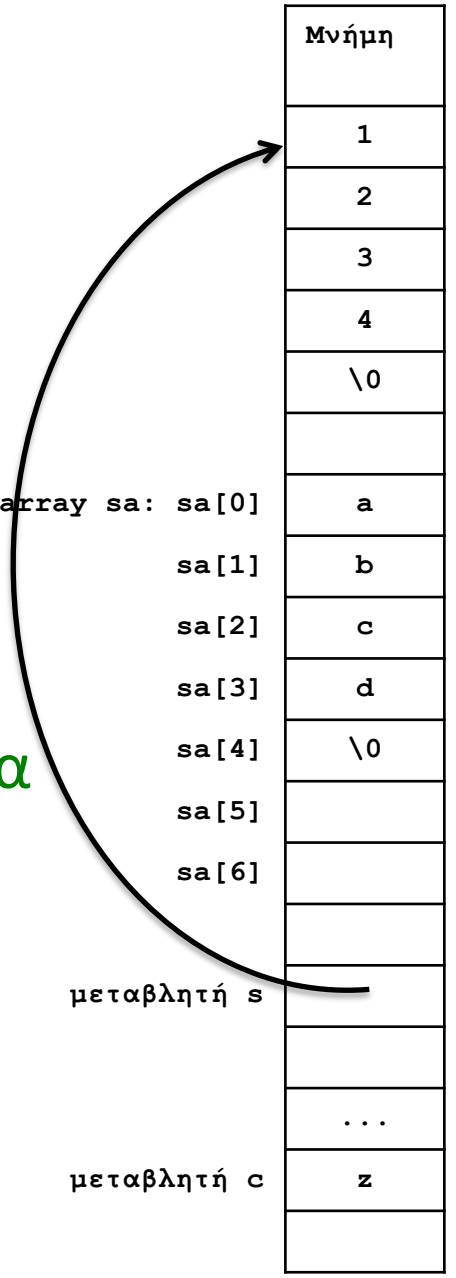
Ποια η σχέση των strings με arrays

- Στη C δεν υπάρχει χωριστός τύπος string
- Αντί αυτού χρησιμοποιείται η σύμβαση/ορισμός:
  - (1) String είναι ένα array από στοιχεία char που το τερματίζεται με '\0', π.χ.
    - char s[10] = "1234";
    - char c = 'a';
  - (2) Ωστόσο, η γλώσσα ορίζει ότι σταθερές τύπου string περιγράφονται μέσα σε "..." και αυτόματα τερματίζονται με \0.
- Το string s έχει χώρο για 10 chars
- Κατά την δήλωση αρχικοποιείται σε ένα string "1234" που αυτόματα τερματίζεται σε \0 (άρα χρησιμοποιεί τις 5 από τις 10 θέσεις του s)
- Αφού, εξ' ορισμού, τα strings είναι arrays από chars που τερματίζονται σε '\0', μπορούμε να χρησιμοποιήσουμε για strings ότι εκφράσεις επιτρέπουν τα arrays, π.χ. c=s[1]; *(s+2) = 'a';



Ποια η σχέση των strings με pointers?

- Ότι υπαγορεύει η γλώσσα με τους μέχρι τώρα ορισμούς arrays και pointers …
  - char *s = “12345”;
  - char sa[6] = ”abcd”;
  - char c = ‘z’;
  - Το s είναι μια μεταβλητή τύπου pointer που δείχνει σε μια σταθερή τιμή string (array από chars που τερματίζεται με ‘\0’).
    - Για το s ισχύει ότι ισχύει για pointers
  - Το sa είναι ένα array από chars που έχει αρχικοποιηθεί σε μια σταθερή τιμή string (chars με \0 στο τέλος)
- Τα s, sa, c είναι (εντελώς) διαφορετικά αντικείμενα στη μνήμη
- Η γλώσσα μας επιτρέπει να γράψουμε όλες τις εκφράσεις που επιτρέπονται για arrays, pointers
  - *s = ‘a’;
  - sa[1] = ‘a’
  - s = sa;



Ακριβέστερα: Πότε ένα array είναι το ίδιο με ένα pointer

- Ένα array name σε μια έκφραση (σε αντίθεση με μια δήλωση) είναι ισοδύναμο με έναν pointer
  - Αλλά: στις δηλώσεις μεταβητών το όνομα ενός array είναι διαφορετικό αντικείμενο από το όνομα ενός pointers
    - Π.χ. οι δηλώσεις int A[10] και int *A δηλώνουν διαφορετικά αντικείμενα.
  - Εκτός από την περίπτωση: Ένα array name είναι το ίδιο με έναν pointer στο πρώτο element του array στην δήλωση παραμέτρων συναρτήσεων
    - Π.χ. Οι δηλώσεις f(int A[10]) και f(int *A) είναι ακριβώς ίδιες
    - Αυτή η εξαίρεση έγινε στη C για λόγους απόδοσης όταν οι παράμετροι των συναρτήσεων είναι μεγάλα arrays
- Ένα index σε έναν πίνακα είναι πάντα ισοδύναμο με το offset από έναν pointer
- Η σχέση των arrays με pointers και η χρήση της αριθμητικής των pointers έχουν ”συνέπειες” στη C:
  - Οι θέσεις ενός array πρέπει να είναι σε συνεχόμενες θέσεις μνήμης
  - Οι θέσεις ενός array “πηγαίνουν” από μικρότερες προς μεγαλύτερες διευθύνσεις στη μνήμη

Προσοχή με τους pointers

- Όταν χρησιμοποιείται pointers, arrays, strings δηλώστε προσεκτικά την κάθε μεταβλητή ανάλογα με το τι θέλετε να εκφράσετε, π.χ.
  - Θέλετε να κρατήσετε χώρο στη μνήμη; Πόσο;
  - Χρειάζεστε μια βοηθητική μεταβλητή που θα δείχνει/διατρέχει άλλες μεταβλητές;
- Γιατί χρειαζόμαστε pointers?
  - Απόδοση, π.χ. Οι pointers μας επιτρέπουν να συνθέτουμε αντικείμενα από υπάρχουσες τιμές, χωρίς να χρειάζεται να δημιουργούμε νέα αντίγραφα.
  - Ευκολία στην έκφραση συγκεκριμένων προγραμμάτων (many would argue against)
- Παρατηρήστε ότι μέχρι τώρα δεν έχουμε πει τίποτε για δυναμική μνήμη
  - Οι pointers δεν ταυτίζονται με την δυναμική μνήμη
  - Μέχρι τώρα τους ορίσαμε και τους χρησιμοποιούμε χωρίς δυναμική μνήμη
  - Όλη η μνήμη που χρησιμοποιούμε δεσμεύεται μέσω δηλώσεων μεταβλητών
- Πολλές φορές οι pointers είναι η πηγή πολλών-πολλών bugs και προβλημάτων, για αυτό και υπάρχουν γλώσσες που απαγορεύουν τους pointers, π.χ.
  - `int *f(void){int i=2; return &i;}`
  - `int main(void){int ip; ip=f(); *ip=...}`
  - Αυτό είναι ένα απόλυτα valid και παντελώς λάθος πρόγραμμα που δεν έχει κανένα νόημα…

Reading

- King Ch. 11, 12, 13
- Chapter 5 of
  (online) The C Book, Mike Banahan, Declan Brady and Mark Doran, Second Edition, 2003. [html] [pdf (updated 2020)]