

Κεφάλαιο 5: Τύποι Δεδομένων

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας, Π. Χατζηδούκας

Γενικά (1)

Όλοι μας έχουμε αναπτύξει μια διαισθητική άποψη για το τι είναι ένας Τύπος Δεδομένων. Δηλαδή:

- Συλλογή τιμών από ένα πεδίο («δηλωτική» προσέγγιση - denotational)
- Εσωτερική δομή συνόλου δεδομένων, που περιγράφεται μέχρι το επίπεδο ενός μικρού συνόλου βασικών τύπων («κατασκευαστική» προσέγγιση - constructive)
- Συλλογή καλά ορισμένων λειτουργιών που μπορούν να εφαρμοστούν στα στοιχεία ενός Τύπου Δεδομένων («αφαιρετική» προσέγγιση – abstraction based)

Γενικά (2)

Τύπος Δεδομένων (ΤΔ):

Ένα σύνολο *στοιχείων* και ένα σύνολο *ενεργειών* πάνω στα στοιχεία αυτά, οι οποίες *ενέργειες δημιουργούν, υποστηρίζουν, καταστρέφουν, τροποποιούν και συλλέγουν* εμφανίσεις των στοιχείων.

- Κάθε ΓΠ παρέχει ένα καταρχήν σύνολο ΤΔ (**βασικοί ΤΔ** – primitive). Π.χ.
 - *Επιτακτικές Γλώσσες*: Integer, Real, Character, Boolean,...
- Ο προγραμματιστής συνήθως έχει τη δυνατότητα ορισμού νέων ΤΔ.

Γενικά (3)

Οι ΤΔ εξυπηρετούν 2 βασικούς **στόχους**:

1. Δημιουργούν ένα συγκεκριμένο *περιβάλλον* (context) στο οποίο εκτελούνται διάφορες λειτουργίες, απαλλάσσοντας τον προγραμματιστή από τη σχετική δουλειά. Π.χ.
Η έκφραση $A + B$ θα επιβάλλει ακέραιη πρόσθεση αν τα A , B είναι τύπου **integer**, ενώ αν τα A , B είναι τύπου **float**, θα επιβάλλει πρόσθεση αριθμητικής floating-point.
2. Αποτρέπουν τους προγραμματιστές από *λάθος λειτουργίες* που ίσως προσπαθήσουν να κάνουν. Π.χ. να προσθέσουν ένα χαρακτήρα με ένα record (*struct*).

Γενικά (4)

■ Σύστημα Τύπων (Type System):

Η δυνατότητα ορισμού νέων ΤΔ και δήλωσης μεταβλητών, οι τιμές των οποίων περιορίζονται στα στοιχεία ενός ΤΔ, με πραγματοποίηση *ελέγχου τύπου*. Ένα Type System αποτελείται από:

1. Ένα μηχανισμό **ορισμού** ΤΔ και συσχέτισής τους με συγκεκριμένες κατασκευές της γλώσσας, δηλαδή με αυτές που έχουν τιμή, όπως μεταβλητές, παράμετροι, εκφράσεις, ...
2. Ένα σύνολο **κανόνων** για *Ισοδυναμία ΤΔ* (type equivalence), *Συμβατότητα ΤΔ* (type compatibility) και *Εξαγωγή ΤΔ* (type inference).

Γενικά (5)

■ Έλεγχος Τύπου (Type Checking):

Διαδικασία επιβεβαίωσης ότι το πρόγραμμα υπακούει στους κανόνες Συμβατότητας ΤΔ της ΓΠ. Δύο έννοιες σχετίζονται με τον Έλεγχο Τύπου:

□ Γλώσσες Ισχυρών Τύπων (Strongly Typed)

Απαγορεύουν την εφαρμογή ενεργειών σε στοιχεία που δεν πρέπει να υποστηρίζουν τις ενέργειες αυτές.

□ Γλώσσες Στατικών Τύπων (Statically Typed)

Strongly Typed γλώσσες στις οποίες ο Έλεγχος Τύπου γίνεται κατά τη μετάφραση (συνήθως, μόνο το μεγαλύτερο μέρος του ελέγχου)

Γενικά (6)

Παραδείγματα:

- Η **Common Lisp** είναι strongly typed, αλλά όχι statically typed.
- Οι **C**, **Ada** είναι statically typed.
- Η **Pascal** είναι σχεδόν statically typed.
- Η **Java** είναι strongly typed, με περίεργο μίγμα πραγμάτων που άλλα ελέγχονται στατικά, και άλλα δυναμικά.

Γενικά (7)

■ Πολυμορφισμός (polymorphism):

Επιτρέπει σε ένα τμήμα κώδικα να δουλεύει με αντικείμενα πολλαπλών τύπων.

- Αναγκαίος ο έλεγχος ΤΔ στο χρόνο εκτέλεσης (κόστος).
- **Lisp, Smalltalk**
- Σε αρκετές object-oriented ΓΠ (**C++**, **Java**, ...) υπάρχει ο **Πολυμορφισμός Υποτύπων** (subtype polymorphism):
Επιτρέπει σε μια μεταβλητή X τύπου T να συσχετίζεται με ένα αντικείμενο οποιουδήποτε υπο-τύπου του T .
- Καθώς οι υπο-τύποι υποστηρίζουν όλες τις λειτουργίες του T , ο μεταφραστής είναι σίγουρος ότι κάθε πράξη αποδεκτή για αντικείμενο τύπου T , θα είναι αποδεκτή για κάθε αντικείμενο που συσχετίζεται με τη X .
- Μπορεί να υλοποιηθεί στο χρόνο μετάφρασης.

Κατηγορίες ΤΔ

■ Βαθμωτοί (scalar) ή Απλοί ΤΔ

Το Πεδίο Τιμών (domain) τους αποτελείται από σταθερές τιμές που έχουν μόνο ένα χαρακτηριστικό

- *Βασικοί* ΤΔ (primitive types): `int`, `real/float`, `bool`, `char`
- Τύποι *Απαρίθμησης* (enumeration types)
- Τύποι *Υποπεριοχής* (subrange types)

■ Διακριτοί ή Τακτικοί (ordinal) ΤΔ

Βαθμωτοί ΤΔ των οποίων τα μέλη του Πεδίου Τιμών τους αντιστοιχίζονται με ακέραιους (από τους παραπάνω βαθμωτούς, όχι οι `real/float`).

■ Σύνθετοι ή Δομημένοι (structured) ΤΔ

Το domain αποτελείται από μέλη που έχουν συντεθεί από ένα σύνολο άλλων ΤΔ (`arrays`, `records`, `pointers`, ...)

Βασικοί ΤΔ (Primitive)

- Δεν ορίζονται με βάση άλλους ΤΔ
- Ορισμένοι βασικοί ΤΔ απλώς αντανακλούν το H/W (π.χ. integer)
- Άλλοι βασικοί ΤΔ θέλουν μόνο λίγο επιπλέον S/W υποστήριξη
- **C**, **C++** : **int**, **float**, **double**, **char**
- **C++**, **Java** : Οι παραπάνω, και επιπλέον **bool** (-ean)

Βασικοί ΤΔ (2)

1. Ακέραιοι Αριθμοί (Integer)

- Το υποσύνολο των ακέραιων από MININT έως MAXINT
- Πολλοί Η/Υ και Γλώσσες υποστηρίζουν διάφορα μεγέθη. Π.χ.
 - `int` (4 bytes)
 - `short int` (2 bytes)
 - `long int` (8 bytes)
 - `unsigned int` (2 bytes)
- **Java** : `byte` (1 byte), `short` (2 bytes), `int` (4 bytes), `long` (8 bytes)

Βασικοί ΤΔ (3)

2. Αριθμοί Κινητής Υποδιαστολής (Floating – Point)

- Είναι προσεγγιστικές αναπαραστάσεις πραγματικών αριθμών.
- IEEE floating – point standard 754:

- Single precision (π.χ. **float**):



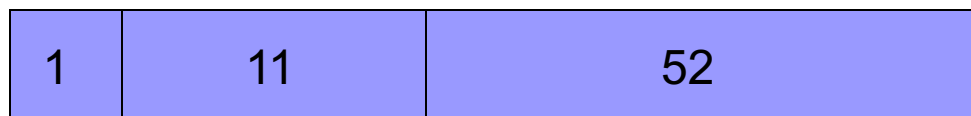
4 bytes (32 bits)

Πρόσημο

Εκθέτης

Πολλαπλασιαστής

- Double precision (π.χ. **double**):



8 bytes (64 bits)

Πρόσημο

Εκθέτης

Πολλαπλασιαστής

Βασικοί ΤΔ (4)

3. Boolean

- Δύο Τιμές: FALSE, TRUE (1 byte)
- Όλες οι γλώσσες έχουν, εκτός της C.

4. Χαρακτήρες

- `char` στη C.
- Κωδικοποίηση ASCII: 1 byte
- Κωδικοποίηση UNICODE: τουλάχιστον 2 bytes

Ακολουθίες Χαρακτήρων (1)

(Character Strings)

Σχεδιαστικά ερωτήματα:

- Βασικός ΤΔ ή ειδικός τύπος char array;
- Στατικό ή Δυναμικό μήκος;
- **Pascal, Ada, C, C++** : Όχι βασικός ΤΔ. Char Array.
Π.χ. `char line[100];`
- **C** : String operations από Standard Library `string.h` :
 - `strcpy` (μετακίνηση/αντιγραφή string)
 - `strcat` (πρόσθεση strings)
 - `strcmp` (σύγκριση strings)
 - `strlen` (αριθμός χαρακτήρων)

Ακολουθίες Χαρακτήρων (2)

- **C** : Char pointers (δείχνουν σε char array):

```
char *str = "mary";
```

Τα strings τελειώνουν με null, οπότε δεν χρειάζεται η γνώση του τρέχοντος μήκους του string.

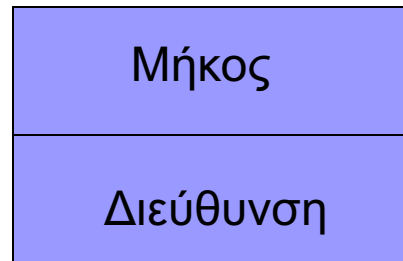
- **Java, C++** :

String class για string objects

- **SNOBOL, Perl** : Πλήρως δυναμικά strings

Ακολουθίες Χαρακτήρων (3)

- **FORTRAN, COBOL, Pascal** : Static Strings



- **C, C++** : Limited Dynamic Strings



Τύποι Απαρίθμησης (enumeration types)

- Στις περισσότερες (και παλιότερες) γλώσσες, το domain των τύπων απαρίθμησης περιγράφεται από μια *διατεταγμένη* λίστα τιμών (σταθερών).
- Στις γλώσσες αυτές, οι μόνες πράξεις που επιτρέπονται, είναι ο έλεγχος *ισότητας* και *διάταξης*, και η *ανάθεση*.
- Π.χ. στην **Pascal**:

```
type days = (mon, tue, wed, thu, fri, sat, sun);
```

```
var x: days;
```

- Υπάρχουν οι predefined functions: ORD, PRED, SUCC
- Απαγορεύεται η χρήση ίδιας σταθεράς σε άλλο ορισμό

Τύποι Απαρίθμησης (2)

- Στις **C**, **C++**, οι τιμές είναι ουσιαστικά ακέραιες σταθερές, οπότε μπορούν να χρησιμοποιηθούν όπως αυτές (ορθογωνιότητα).

- Παράδειγμα:

```
enum boolean {NO, YES, FALSE = 0, TRUE}  
boolean A;  
A = YES - 1;
```

- Επίσης:

```
enum days {mon=1, tue=2, wed=3, ...}  
ή enum days {mon=1, tue, wed, ...}  
ή enum days {mon=1, tue, wed, ... , MON=1, TUE, WED, ...}  
days i = MON;
```

Αν δεν αντιστοιχίσουμε την 1^η τιμή, το default είναι 0. 18

Τύποι Υποπεριοχής (subrange types)

- Είναι ΤΔ οι τιμές του οποίου είναι ένα *συνεχές υποσύνολο* των τιμών κάποιου διακριτού (ordinal) ΤΔ (ακέραιοι, χαρακτήρες, απαριθμήσεις, άλλες υποπεριοχές).
- Εμφανίστηκαν για πρώτη φορά στην **Pascal**, και στη συνέχεια σε μεταγενέστερες ΓΠ της οικογένειας **Algol**.
- **Pascal**:

```
type test_score = 0 .. 100;  
    workday = mon .. fri;
```

ΤΔ Array (1)

- Ομάδα από ομογενή στοιχεία δεδομένων που προσδιορίζονται από τη θέση τους στην ομάδα, σε σχέση με το πρώτο. Ορισμός:
C: `int A[20]` **Pascal:** `A: ARRAY [0..19] of INTEGER`
- **Σχεδιαστικά θέματα**
 1. Default κατώτερη τιμή του δείκτη:
 - **C, C++** : 0
 - **FORTRAN** : 1
 - **Pascal:** Ορίζεται από το χρήστη

ΤΔ Array (2)

2. Διαστάσεις δεικτών:

- **C, C++** : 1 διάσταση, αλλά κάθε element του array μπορεί να είναι array (multidimensional): `int B[5][4]`
- **FORTRAN** : 3 διαστάσεις στην **I**, 7 στη **FORTRAN IV**

3. Αρχικοποίηση τιμών array κατά τη δήλωση:

- **FORTRAN 77** : `INTEGER L(3)`
`DATA L /0, 5, 9/`
- **C, C++** : `int L [] = {4, 7, 8, 53}` Ορίζεται και το μήκος array (4)
`char name [] = "freddie"` Array μήκους 8 (null στο τέλος)
`char *names [] = {"Bob", "Jake", "Mary"}`
Array of pointers σε χαρακτήρες: Το `names[0]` είναι pointer στο γράμμα 'B' στο char array "Bob/null"
- **Pascal** : Όχι

ΤΔ Array (3)

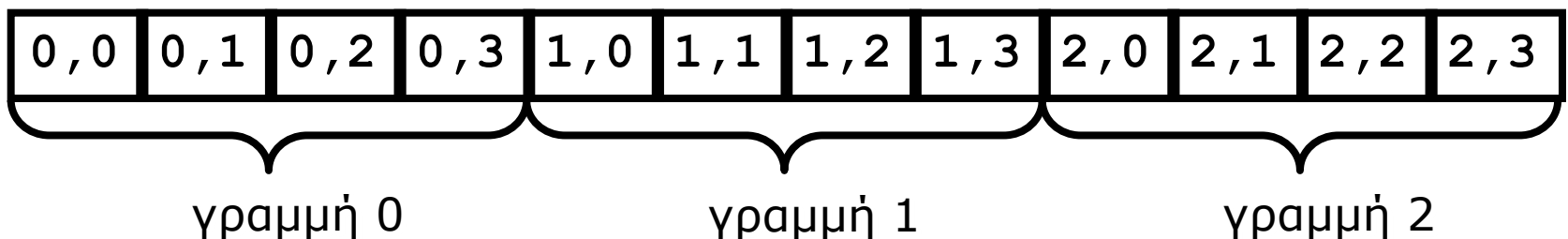
- Στη C μπορούμε να δηλώσουμε και πίνακες πολλών διαστάσεων
- Παράδειγμα δήλωσης πίνακα 2 διαστάσεων:

```
int a[3][4];
```

Γραμμές 0..2

Στήλες 0..3

- Παράδειγμα προσπέλασης στοιχείου πίνακα 2 διαστάσεων:
`a[1][0] = 5; /* στη γραμμή 1 και στήλη 0 αποθήκευσε το 5 */`
- Στη C η αποθήκευση των στοιχείων πολυδιάστατου πίνακα γίνεται ανά γραμμή, π.χ.



ΤΔ Record (1)

- Ομάδες στοιχείων που αποτελούν συνθέσεις από συγκεκριμένο αριθμό (πιθανόν) ανομοιογενών στοιχείων δεδομένων, που αναγνωρίζονται από το όνομά τους.
- Διαφορές από arrays:
 - Τα συστατικά των records μπορεί να είναι ετερογενή.
 - Τα στοιχεία των records έχουν συμβολικά ονόματα (id), ενώ των arrays καθορίζονται από το δείκτη.

ΤΔ Record (2)

■ C, C++ :

```
struct EmployeeType
{
    int ID;
    int Age;
    float Salary;
    char Dept;
} Employee[500] ;
```

Ορίζεται record ΤΔ με όνομα `EmployeeType`, και παράλληλα δηλώνεται ένα array 500 θέσεων με όνομα `Employee`, με στοιχεία τύπου `EmployeeType`.

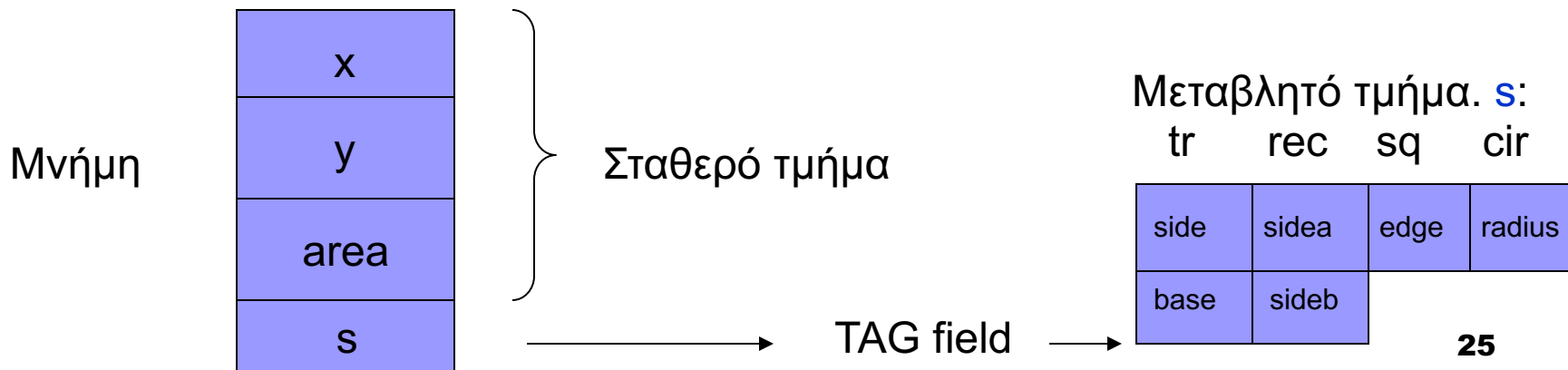
Πρόσβαση στα στοιχεία του: `Employee[3].Salary`

Άλλος ορισμός (απλής μεταβλητής): `struct EmployeeType A;`

TΔ Record (3)

■ Pascal :

```
type shape = (triangle, rectangle, square, circle);
coordinates = RECORD
  x, y: real;
  area: real;
  case s: shape of
    triangle: (side: real; base: real);
    rectangle: (sidea, sideb: real);
    square: (edge: real);
    circle: (radius: real)
end;
```



ΤΔ nested structs (4)

Εμφωλευμένες Δομές

```
struct point { /* δομή για ένα σημείο */
    double x, y;
};

struct rect { /* δομή για ένα παραλληλόγραμμο
              που ορίζεται από 2 αντιδιαμετρικές
              κορυφές του */
    struct point pt1;
    struct point pt2;
};

struct rect screen;
```

Το `screen.pt1.x` αναφέρεται στη x συντεταγμένη του σημείου `pt1` της δομής `screen`

ΤΔ nested structs and functions (4)

Δομές και Συναρτήσεις

```
/* συνάρτηση που επιστρέφει δομή point */  
struct point makepoint(double x, double y)  
{  
    struct point temp;  
    temp.x = x;  
    temp.y = y;  
    return temp;  
}
```

```
/* ορισμός παραλληλογράμμου με κορυφές (0,0) και (10,10)*/  
screen.pt1 = makepoint(0.0, 0.0);  
screen.pt2 = makepoint(10.0, 10.0);
```

```
if (screen.pt1 != screen.pt2) ...
```

Προσοχή: οι δομές δε μπορούν να συγκριθούν (μόνο τα επιμέρους πεδία τους)

ΤΔ δείκτη (pointer)

- **Δείκτης:** Αναφορά σε θέση μνήμης
- **Μεταβλητή ΤΔ δείκτη:** Όνομα (id) με τιμή που είναι αναφορά σε διεύθυνση μνήμης. Δηλαδή, οι τιμές της περιέχονται στο σύνολο:
[διευθύνσεις μνήμης, nil (τίποτα)].
- Δεν είναι Δομημένος ΤΔ

Θέματα Σχεδιασμού

- Εμβέλεια και διάρκεια ζωής μιας μεταβλητής ΤΔ δείκτη
- Διάρκεια ζωής Heap – Dynamic μεταβλητής
- Τυχόντες περιορισμοί στον ΤΔ του αντικειμένου που δείχνουν

ΤΔ δείκτη (2)

■ Λόγοι ύπαρξης ΤΔ δείκτη:

- Πολλά στοιχεία μπορούν να συνδέονται μεταξύ τους χωρίς να παρέχονται συγκεκριμένα ονόματα για όλα (στις *Explicit Heap – Dynamic Variables*)
- Βάση για έμμεση διευθυνσιοδότηση.
- Επιτρέπουν την ταυτόχρονη τοποθέτηση των στοιχείων σε πολλές δομές (π.χ. λίστες, arrays).
- Με λίγες δηλώσεις, μπορούμε να έχουμε μεγάλη ποικιλία στοιχείων που συνδέονται με πολλούς τρόπους και προσπελούνται με ομοιόμορφο τρόπο.

ΤΔ δείκτη (3)

- Συνέπειες (και πιθανά προβλήματα) ύπαρξης δεικτών:
 - Μπορεί αρκετές Μεταβλητές Δείκτη (ΜΔ) να αναφέρονται στο ίδιο αντικείμενο, στο ίδιο σημείο, ή σε διαφορετικά σημεία του ίδιου αντικειμένου.
 - Πρέπει η ΓΠ να παρέχει σημειογραφία για διάκριση **θέσης**, **τιμής-*r*** και **τιμής-*l***.
 - Μια μεταβλητή ΤΔ δείκτη, μπορεί να μη δείχνει πουθενά.

Π.χ.:

ΤΔ δείκτη (4)

Στη **C++** μπορούμε να έχουμε:

```
int *arrayPtr1;  
int *arrayPtr2 = new int[100];  
arrayPtr1 = arrayPtr2;  
delete [ ] arrayPtr2;
```

Πρόβλημα **αιωρούμενης αναφοράς...**

ΤΔ δείκτη (5)

- Κάποια αντικείμενα μπορεί να μη δεικτοδοτούνται πλέον από καμία ΜΔ. Π.χ.:

```
int *P1 = new int;  
int *P2 = new int;  
.....  
P2 = P1;
```

Τώρα πλέον, η heap μνήμη στην οποία έδειχνε αρχικά το **P2**, δεν μπορεί να προσπελασθεί (χαμένη μεταβλητή ή σκουπίδι...)

ΤΔ δείκτη (6)

- **C, C++** : Οι δείκτες και τα arrays είναι στενά συνδεδεμένες έννοιες:

```
int n;
```

```
int *a;    (δείκτης σε integer)
```

```
int b[10]; (array 10 integers)
```

Τα παρακάτω είναι νόμιμα:

```
a = b;
```

(το a θα δείχνει στο b[0])

```
n = a[3];
```

(το n παίρνει την τιμή του b[3])

```
n = *(a+3)
```

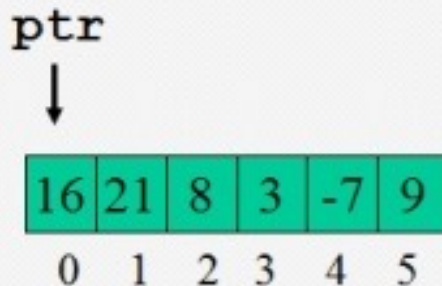
(ίδιο με το προηγούμενο)

ΤΔ δείκτη (6.1)

Δείκτες και Πίνακες

Μια μεταβλητή πίνακα είναι ένας **σταθερός** δείκτης προς το πρώτο στοιχείο του πίνακα

```
int table[6] = {16,21,8,3,-7,9};  
int *ptr;  
ptr = &table[0];
```

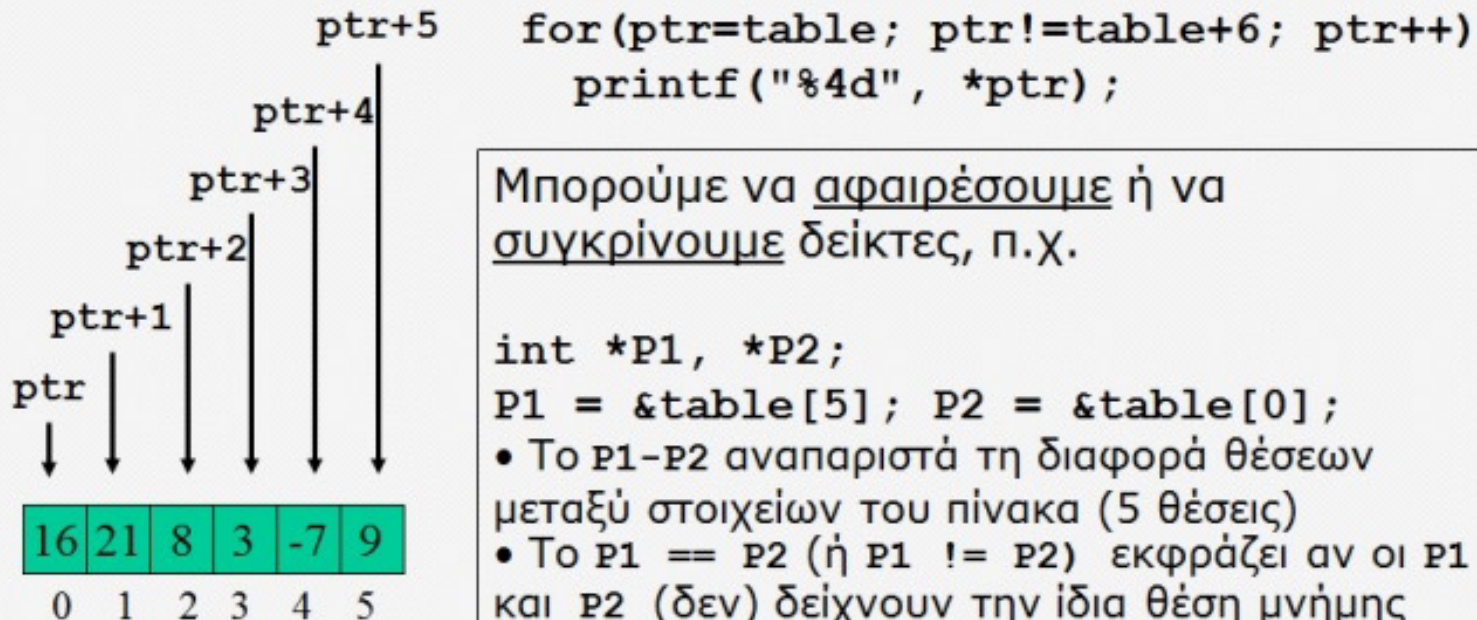


Το να γράψουμε `table` είναι το ίδιο με το να γράψουμε `&table[0]` ή `ptr`.

ΤΔ δείκτη (6.2)

Αριθμητική Δεικτών

Μπορούμε να προσθέσουμε μια ακέραια τιμή σε ένα δείκτη, π.χ. το `ptr=ptr+1` ή `ptr++` κάνει το δείκτη να δείξει το επόμενο στοιχείο του πίνακα `table` (δηλ. στον επόμενο `int`)



Προσοχή: η αριθμητική δεικτών δεν εφαρμόζεται σε μεταβλητές πίνακα

ΤΔ δείκτη (6.3)

Συμβολοσειρές και Δείκτες



magic

```
#include <stdio.h>
```

```
void main(void)
```

```
{ const char nul = 0;
```

```
  char *magic = "abracadabra";
```

```
  char *ptr;
```

```
  for (ptr = magic; *ptr != nul; ptr++)
```

```
    printf("%c", *ptr);
```

```
}
```

ΤΔ δείκτη (7)

- * για αποαναφοροποίηση και ορισμό μεταβλητών δείκτη
- & διεύθυνση μνήμης

```
int *ptr;
```

```
int count, init, index;
```

```
...
```

```
ptr = &init;
```

```
count = *ptr;
```

```
ptr = ptr + index;
```

(το `index` κλιμακώνεται ανάλογα με τα κελιά μνήμης στα οποία δείχνει το `ptr`)

```
ptr = 0
```

(δηλαδή `ptr` γίνεται ίσο με `nil`)

ΤΔ δείκτη (8)

Γενικά οι δείκτες δείχνουν σε αντικείμενα της heap memory. Στη **C** μπορεί να δείχνει ο δείκτης και σε αντικείμενα της stack μνήμης. Στην **Pascal** όχι.

ΠΙΝΑΚΕΣ ΜΕ ΔΕΙΚΤΕΣ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ???

ΠΙΝΑΚΕΣ ΜΕ ΔΕΙΚΤΕΣ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ (1)

```
#include <stdio.h>
```

```
void add(int a, int b)
```

```
{
```

```
    printf("Addition is %d\n", a+b);
```

```
}
```

```
void subtract(int a, int b)
```

```
{
```

```
    printf("Subtraction is %d\n", a-b);
```

```
}
```

```
void multiply(int a, int b)
```

```
{
```

```
    printf("Multiplication is %d\n", a*b);
```

```
}
```

ΠΙΝΑΚΕΣ ΜΕ ΔΕΙΚΤΕΣ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ (2)

```
int main()
{
    // fun_ptr_arr is an array of function pointers
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;

    printf("Enter Choice: 0 for add, 1 for subtract and 2 for multiply\n");
    scanf("%d", &ch);
    if (ch > 2) return 0;
    (*fun_ptr_arr[ch])(a, b);
    return 0;
}
```

Enter Choice: 0 for add, 1 for subtract and 2 for multiply

2

Multiplication is 150

ΤΔ δείκτη (9)

- Δημιουργία αντικειμένου από τη heap memory:
 - **C** : `int *x;`
`x = malloc(sizeof(int));`
 - **C++** : `int *x;`
`x = new int;`
 - **Pascal** : `var ^x: integer;`
`new (x)`
- Καταστροφή αντικειμένου από τη heap memory:
 - **C** : `free(x);`
 - **C++** : `delete x;`
 - **Pascal** : `dispose(x);`

ΤΔ δείκτη (9.1)

Ας συζητήσουμε τι συμβαίνει στον κώδικα που ακολουθεί:

Δήλωση δείκτη P χωρίς Αρχικοποίηση

```
int *P;
```

Ο P «δείχνει» σε περιοχή μνήμης μεγέθους 1 ακεραίου που παραχωρήθηκε δυναμικά



```
P = (int *) malloc (sizeof (int));
```



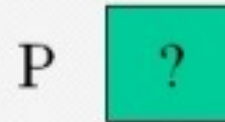
```
*P = 10;
```

Στην περιοχή μνήμης που «δείχνει» ο P αποθηκεύεται η τιμή 10



```
free (P);
```

Απελευθερώνεται η περιοχή μνήμης όπου «δείχνει» ο P



ΤΔ δείκτη (9.2)

Δυναμική δημιουργία πίνακα μέσω δείκτη

```
int *p;
p = (int *) malloc(100 * sizeof(int));
if (p == NULL)
    ...
for (i = 0; i < 100; i++)
    p[i] = ...
free(p);
```

Ορίζουμε δείκτη προς τον επιθυμητό τύπο στοιχείων

Παραχωρούμε στο δείκτη την απαιτούμενη ποσότητα μνήμης (ανάλογα με τον αριθμό των στοιχείων που επιθυμούμε)

Ελέγχουμε αν δεν ήταν δυνατή η παραχώρηση της μνήμης

Προσπελάζουμε τα στοιχεία του πίνακα

Απελευθερώνουμε την παραχωρηθείσα μνήμη, όταν δε χρειαζόμαστε πλέον τον πίνακα

ΤΔ δείκτη (10)

ΤΔ Αναφοράς (Reference Type)

- Είναι ουσιαστικά ένα ψευδώνυμο μιας υπάρχουσας μεταβλητής. Χρήση κυρίως ως παράμετροι συναρτήσεων.
- Διαφορές με pointers:
 - Οι αναφορές δεν έχουν ως τιμή διεύθυνση/εις μνήμης.
 - Οι αναφορές δεν μπορούν να δείχνουν σε null.
 - Μια αναφορά πρέπει να αρχικοποιηθεί με την δήλωσή της, και δεν μπορεί να αλλάξει η αναφορά σε άλλο αντικείμενο.

Π.χ. στη C++ :

```
int result = 0;  
int &ref_result = result;  
...  
ref_result = 100;
```

Τα `result` και `ref_result` είναι ψευδώνυμα...

ΤΔ δείκτη σε structs (11)

Αρχικοποίηση και Προσπέλαση Δομής

```
struct point {  
    double x, y;  
};
```

```
struct point p1 = {0.5, 0.4}, p2, *ppoint;
```

```
printf("%f, %f\n", p1.x, p1.y);
```

```
ppoint = &p1;
```

```
printf("%f, %f\n", ppoint->x, (*ppoint).y);
```

```
ppoint = &p2;
```

```
ppoint->x = 7.0;
```

```
(*ppoint).y = 6.0;
```

Αρχικοποίηση δομής

Μη αρχικοποιημένη δομή

Δομή.Πεδίο

Δείκτης σε δομή

Ο ppoint δείχνει στην p1

Δείκτης->Πεδίο

ή (*Δείκτης).Πεδίο

ΤΔ δείκτη σε string πεδίο δομής(12)

Πίνακες Δομών

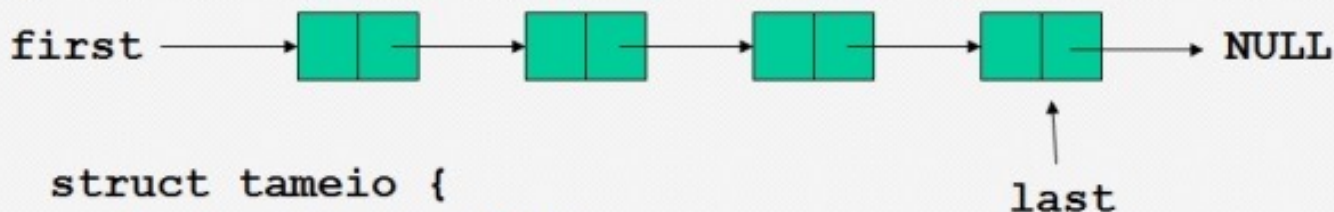
```
struct key {  
    char *word;  
    int count;  
} keytab[] = {  
    {"break", 0},  
    {"case", 0},  
    {"char", 0},  
    {"const", 0},  
    {"continue", 0},  
    {"default", 0},  
    {"unsigned", 0},  
    {"void", 0},  
    {"while", 0}  
};
```

← Πίνακας δομών με δεσμευμένες λέξεις της C και το πλήθος εμφάνισής τους. Το μέγεθος του πίνακα keytab προκύπτει από το πλήθος των στοιχείων αρχικοποίησης (keytab[9])

ΤΔ δείκτη σε πεδίο αυτοαναφορικής δομής (13)

Δυναμικές (συνδεδεμένες) Δομές

- Πρόκειται για δομές που περιέχουν δείκτες προς άλλες δομές
- Μας επιτρέπουν να χειριστούμε δεδομένα όταν μας είναι άγνωστο το πλήθος τους (με δυναμική παραχώρηση μνήμης)
- Π.χ. μπορούμε να αναπαραστήσουμε τους πελάτες σε ένα ταμείο με μια συνδεδεμένη λίστα, στην οποία να προσθέτουμε (αφαιρούμε) κόμβους, καθώς προστίθενται (αποχωρούν) πελάτες

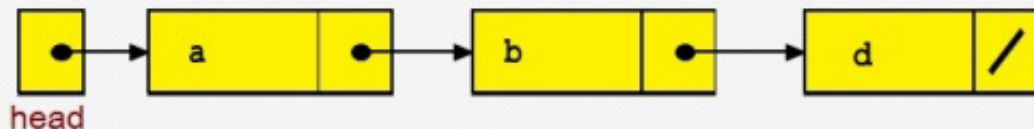


```
struct tameio {  
    char pelatis[20];  
    struct tameio *next;  
};  
  
struct tameio *first, *last;
```

ΤΔ δείκτη σε πεδίο αυτοαναφορικής δομής (13.1)

Συνδεδεμένες Λίστες

Ακολουθίες κόμβων όπου κάθε κόμβος έχει δείκτη προς τον επόμενο



```
typedef struct node *NODEPOINTER;
```

```
typedef struct node {  
    char data;  
    NODEPOINTER next;  
} NODE;
```

```
typedef struct node {  
    char data;  
    struct node *next;  
} NODE;
```

ή

```
typedef NODE *NODEPOINTER;
```

Δείκτης σε κόμβο:

```
NODEPOINTER head;
```

Προσπέλαση στοιχείου κόμβου:

```
head -> data ή (*head).data
```


ΤΔ δείκτη σε πεδίο αυτοαναφορικής δομής (13.2)

Συνδεδεμένες Λίστες (συνέχεια)

Αρχικοποίηση λίστας:

```
head = NULL;
```

Εισαγωγή στην αρχή:

```
NODEPOINTER insert(NODEPOINTER h, char l) {
    NODEPOINTER temp;

    temp = (NODEPOINTER) malloc(sizeof(NODE));
    if (temp == NULL) {
        fprintf(stderr, "ERROR OF MEMORY ALLOCATION\n");
        exit(1);
    }
    temp ->data = l;
    temp ->next = h;
    return(temp);
}
```

Κλήση insert:

```
head = insert(head, 'a');
```

```
STACK example
/*Linked-List Implementation*/
head = NULL;
head = insert(head, 'd');
head = insert(head, 'b');
head = insert(head, 'a');
```

/*Το head είναι ο δείκτης top που δείχνει το κορυφαίο στοιχείο που θα γίνει pop. Εδώ το insert είναι το push */

ΤΔ δείκτη σε πεδίο αυτοαναφορικής δομής (13.3)

Συνδεδεμένες Λίστες (συνέχεια)

Διάσχιση και εμφάνιση:

```
void display(NODEPOINTER h)
{
    while (h != NULL) {
        printf("LETTER: %c\n", h->data);
        h = h->next;
    }
}
```

Εφαρμογές ΤΔ δείκτη σε πεδία αυτοαναφορικής δομής (13.4)

Μερικές άλλες Δυναμικές Δομές

- Στοιίβες: LIFO (last-in-first-out)
- Ουρές: FIFO (first-in-first-out)
- Διπλά συνδεδεμένες λίστες
- Δυαδικά Δέντρα
- Γράφοι

Ισοδυναμία ΤΔ (1)

- Ο Έλεγχος Τύπου (στατικός ή δυναμικός) εμπεριέχει σύγκριση μεταξύ του ΤΔ του πραγματικού ορίσματος που δίνεται σε μια operation, με τον ΤΔ που αναμένεται στην operation.
- Αν είναι ισοδύναμοι, γίνεται αποδεκτό το όρισμα και προχωράει η operation.
- Αν δεν είναι, τότε είτε **error**, ή γίνεται μετατροπή του ΤΔ του πραγματικού ορίσματος, ώστε να είναι συμβατό με αυτό που αναμένεται (*Συμβατότητα*)
- **Ισοδύναμοι ΤΔ:** *Αν ένας τελεστής του ενός ΤΔ μπορεί να υποκατασταθεί από τελεστή του άλλου ΤΔ, χωρίς Μετατροπή ΤΔ.*

Ισοδυναμία ΤΔ (2)

- Το πρόβλημα της αναγνώρισης της ισοδυναμίας ΤΔ, οφείλεται στη δυνατότητα να ορίζει νέους (σύνθετους) ΤΔ ο χρήστης. Στους απλούς ΤΔ συνήθως δεν υπάρχει πρόβλημα.
- Παράδειγμα (γλώσσα τύπου Pascal):

```
type V1: array[1..10] of real;  
      V2: array[1..10] of real;  
var X, Z: V1;  
     Y: V2;  
procedure Sub(A: V1);  
end;  
BEGIN  
    X:= Y;  
    Sub(Y);  
END.
```

Ερωτήματα:

- X, Y, Z, A έχουν ίδιο ΤΔ;
- Επιτρέπεται το X:= Y;
- Επιτρέπεται το Sub(Y);

Ισοδυναμία ΤΔ (3)

Δύο λύσεις στο πρόβλημα:

■ **A. Ισοδυναμία Ονομάτων** (name equivalence)

Δύο ΤΔ είναι ισοδύναμοι μόνο αν έχουν το ίδιο όνομα. Δηλαδή στο παράδειγμα:

$V1 \neq V2$ και $X := Z$ σωστό, $X := Y, \text{Sub}(Y)$ λάθος

Δημοφιλής λύση: **Java**, **C** και **C++** για struct, enum, union, **Ada**

■ **B. Ισοδυναμία δομών** (structural equivalence)

Δύο ΤΔ είναι ισοδύναμοι αν ορίζουν μεταβλητές με ίδια συστατικά και δομή. Δηλαδή στο παράδειγμα:

$V1 = V2$ και $X := Z, X := Y, \text{Sub}(Y)$ σωστά

C, **C++** για υπόλοιπα, **Algol-68**, **FORTRAN**, **COBOL**

Pascal : Συνδυασμός των 2 (Ισοδυναμία Δήλωσης) 54

Ισοδυναμία ΤΔ (4)

Η **Ada** έχει επίσης δύο ειδικές μορφές ΤΔ:

- **Παράγωγος (Derived) ΤΔ:**

```
type Celsius is new Float;
```

```
type Fahrenheit is new Float;
```

Celsius και **Fahrenheit** δεν είναι ισοδύναμοι ΤΔ. Ουσιαστικά εφαρμόζεται *Ισοδυναμία Ονομάτων*.

- **Υποτύπος (Subtype) ΤΔ:**

```
subtype Small_Int is Integer range 0..99;
```

Ο ΤΔ **Small_Int** είναι ισοδύναμος με τον **Integer**. Ουσιαστικά εφαρμόζεται *Ισοδυναμία Δομών*.

- ```
type D is new Integer range 1..100;
```

- ```
subtype S is Integer range 1..100;
```

Τα **D** και **S** δεν είναι ισοδύναμοι ΤΔ αν και ίδιας δομής. Το **S** είναι ισοδύναμος ΤΔ με το **Integer** και κάθε υποτύπο του. Το **D** όχι.

Συμβατότητα ΤΔ

■ Μετατροπή ΤΔ

Διαδικασία στην οποία μια τιμή ενός ΤΔ, μετατρέπεται σε τιμή άλλου ΤΔ.

Κατηγορίες Μετατροπών:

- Διεύρυνση (widening). Π.χ. από `int` σε `float`
- Περιορισμός (narrowing). Π.χ. από `float` σε `int` (truncation)

Δύο Τρόποι:

- *Implicit* (στη `C` όλα επιτρέπτά, στην `Pascal` μόνο διεύρυνση `INT` σε `REAL`)
- *Explicit* (στην `Pascal` : `i:= trunc(r)`, `i:= round(r)`)

Εξαγωγή ΤΔ (type inference)

```
type Atype = 0..20;  
      Btype = 10..20;  
var  a : Atype;  
     b : Btype;
```

Ποιος είναι ο ΤΔ του $a + b$;

Συνήθης απάντηση: Ο αρχικός βασικός ΤΔ του
subrange ΤΔ, δηλαδή *integer*