Σύνθετοι Τύποι Δεδομένων

Σύνθετοι τύποι δεδομένων

- Συχνά τα δεδομένα ενός προγράμματος δεν είναι απλά κάποιοι μεμονωμένοι αριθμοί ή χαρακτήρες.
- Χρειάζεται να **ομαδοποιήσουμε** περισσότερα δεδομένα διαφορετικού τύπου σε μια **οντότητα** για την πιο εύκολη και αποδοτική διαχείριση τους.
- Π.χ. 3 ακέραιες μεταβλητές που χρησιμοποιούμε για να καταγράψουμε τη μέρα, το μήνα και το έτος μιας ημερομηνίας.
- Με την βοήθεια των **σύνθετων** τύπων δεδομένων, ο προγραμματιστής μπορεί να ορίσει **δικούς του** τύπους με βάση τους υπάρχοντες τύπους δεδομένων.

Δομή - `struct`

- Η δομή (`struct`) ομαδοποιεί δεδομένα διαφορετικών τύπων σε μια μεγαλύτερη και εννιαία οντότητα.
- Τα επιμέρους δεδομένα της δομής ονομάζονται πεδία.
- Το μέγεθος της δομής καθορίζεται (αυτόματα) από τον μεταφραστή, έτσι ώστε να υπάρχει χώρος για την αποθήκευση τιμών των πεδίων.
- Η μνήμη μιας δομής δεσμεύεται **συνολικά**, ως ένα κομμάτι, όμως τα πεδία **δεν** καταλαμβάνουν πάντα συνεχόμενες θέσεις μέσα σε αυτό το κομμάτι.
- **Προσοχή:** η αποθήκευση των δεδομένων που αντιστοιχούν στα διάφορα πεδία στη μνήμη της δομής γίνεται από τον μεταφραστή – **χωρίς** εμείς να γνωρίζουμε τις λεπτομέρειες.

```
struct xxx {
  /* δήλωση πεδίων */
};
…
struct xxx var1,var2;
```

όνομα τύπου δομής

ονόματα μεταβλητών

```
struct xxx {
  /* δήλωση πεδίων */
} var1;
…
struct xxx var2;
```

όνομα τύπου δομής

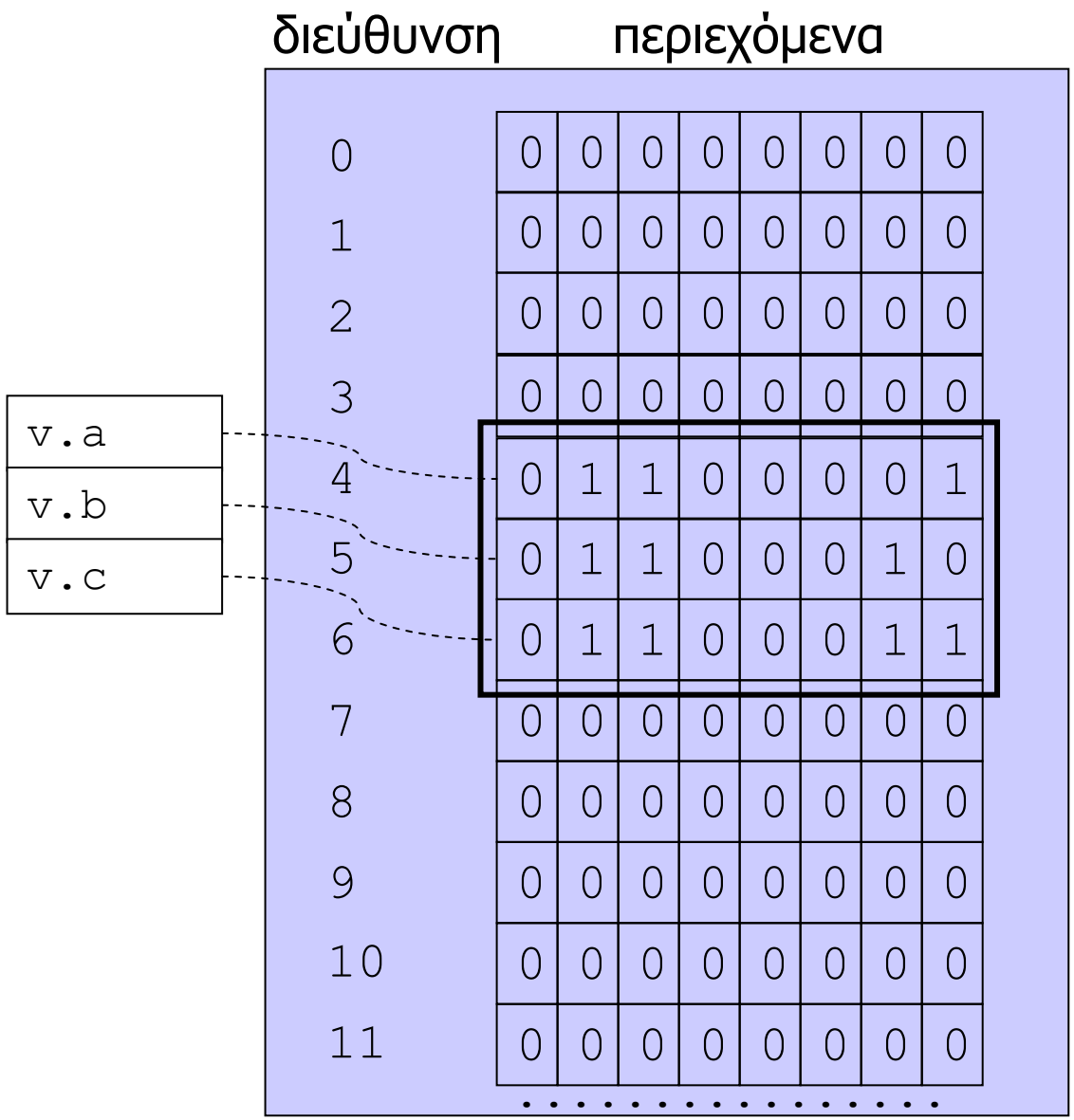
ονόματα μεταβλητών

```
struct {
  /* δήλωση μεταβλητών */
} var1,var2;
```

ανώνυμος τύπος δομής

ονόματα μεταβλητών

```
struct abc {
  char a,b,c;
};
…
struct abc v;

v.a='a';
v.b='b';
v.c='c';
```



Προγραμματισμός με `struct`

- Αναφορά στα πεδία μιας μεταβλητής `struct` γίνεται μέσω του τελεστή `.` που τοποθετείται **ανάμεσα** στο όνομα της μεταβλητής και το όνομα του πεδίου.
- Η ανάθεση τιμών σε μεταβλητές `struct` γίνεται είτε σε συνολικό επίπεδο δομής, όπως για κανονικές μεταβλητές, είτε σε επίπεδο επιμέρους πεδίων.
- Αν δύο μεταβλητές έχουν τον ίδιο τύπο `struct T` τότε μπορεί να γίνει **απ' ευθείας** ανάθεση της τιμής της μιας μεταβλητής στην άλλη.
- Γίνεται αντιγραφή **ολόκληρης** της περιοχής της μνήμης (δεν γίνεται αντιγραφή πεδίο προς πεδίο).

```
struct date {
  int day    /* αριθμός ημέρας: 1..31 */
  int month; /* αριθμός μήνα: 1..12 */
  int year;  /* αριθμός έτους */
};

struct date d1,d2,d3; /* μεταβλητές struct date */

int main(int argc, char *argv[]) {
  d1.day = 25;
  d1.month = 12;
  d1.year = 2000+5;

  d2.day = d1.day;
  d2.month = d1.month;
  d2.year = d1.year;

  d3 = d1;
  d3.year++;
}
```

Σύγκριση `struct`

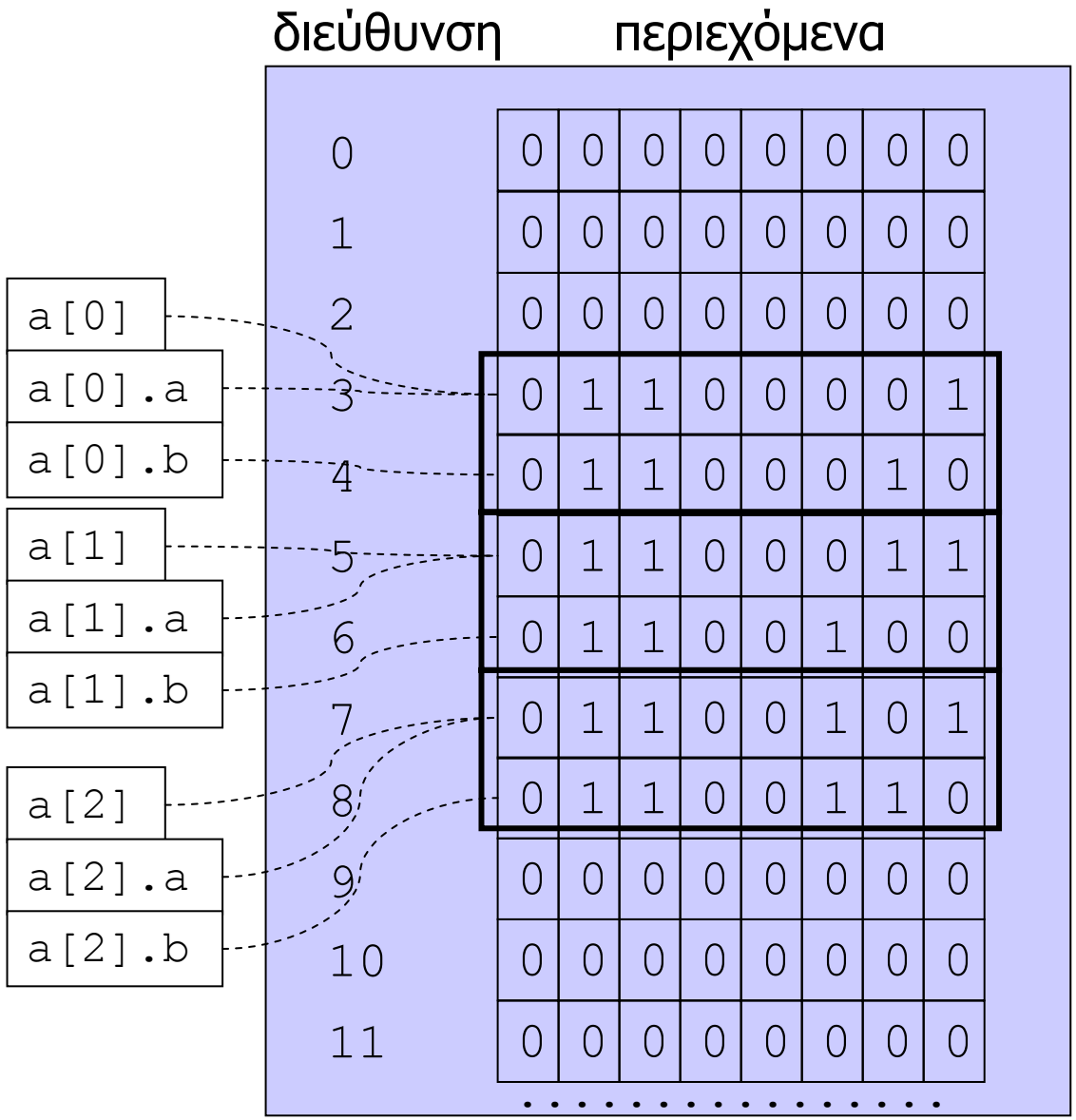
- **Δεν** υποστηρίζεται η σύγκριση των περιεχομένων (συμβατών) `struct` σε **συνολικό** επίπεδο δομής.
- Ο λόγος είναι ότι ο μεταφραστής δεν μπορεί να γνωρίζει την **σημασία** που δίνει ο προγραμματιστής στα επιμέρους πεδία της δομής.
- Η «προφανής» σύγκριση byte προς byte μπορεί να δώσει μη επιθυμητά αποτελέσματα.
- Ο προγραμματιστής πρέπει να υλοποιήσει **δικές του** μεθόδους σύγκρισης με βάση την «σημασία» των επιμέρους πεδίων της δομής.
- **Αντιστοιχία:** σύγκριση πινάκων που περιέχουν αλφαριθμητικά (π.χ. μέσω της συνάρτησης `strcmp`).

```
struct date {
  int day;
  int month;
  int year;
};

…

int datecmp(struct date d1, struct date d2) {
  if (d1.year<d2.year) { return(-1); }
  else if (d1.year>d2.year) { return(1); }
  else if (d1.month<d2.month) { return(-1); }
  else if (d1.month>d2.month) { return(1); }
  else if (d1.day<d2.day) { return(-1); }
  else if (d1.day>d2.day) { return(1); }
  else return(0);
}
```

Πίνακες από `struct`

- Όπως και για τους βασικούς τύπους δεδομένων, μπορεί να οριστούν πίνακες από `struct`.
- Δεσμεύεται ένας **συνεχόμενος** χώρος στην μνήμη για την αποθήκευση όλων των στοιχείων του πίνακα.
- Η πρόσβαση στα στοιχεία του πίνακα γίνεται ακριβώς με τον ίδιο τρόπο, δηλαδή προσδιορισμός θέσης του στοιχείου στον πίνακα –μέσα στα επιτρεπτά όρια.
- Από τη στιγμή προσδιοριστεί ένα συγκεκριμένο στοιχείο του πίνακα, πρόσβαση στα επιμέρους πεδία του `struct` γίνεται κανονικά μέσω του τελεστή .

```
struct s {
  char a,b;
};
…
struct s a[3];

a[0].a='a';
a[0].b='b';

a[1].a='c';
a[1].b='d';

a[2].a='e';
a[2].b='f';
```



Πεδία bits

- Στα πεδία μιας δομής που ορίζονται ως `int` μπορεί προαιρετικά να οριστούν τα bits που χρειάζονται για την αποθήκευση των τιμών – αν φυσικά μπορεί να περιοριστεί το πεδίο τιμών τους εκ των προτέρων.
- Ο μεταφραστής μπορεί να εκμεταλλευτεί αυτή την πληροφορία για να συμπιέσει τα δεδομένα της δομής.
- Αυτό δεν είναι απαραίτητο να γίνει, και σε κάθε περίπτωση η μέθοδος συμπίεσης εξαρτάται από τον μεταφραστή – πιθανό πρόβλημα ασυμβατότητας.
- Η κωδικοποίηση και ερμηνεία των πεδίων bits είναι αποκλειστική υπόθεση του προγραμματιστή.

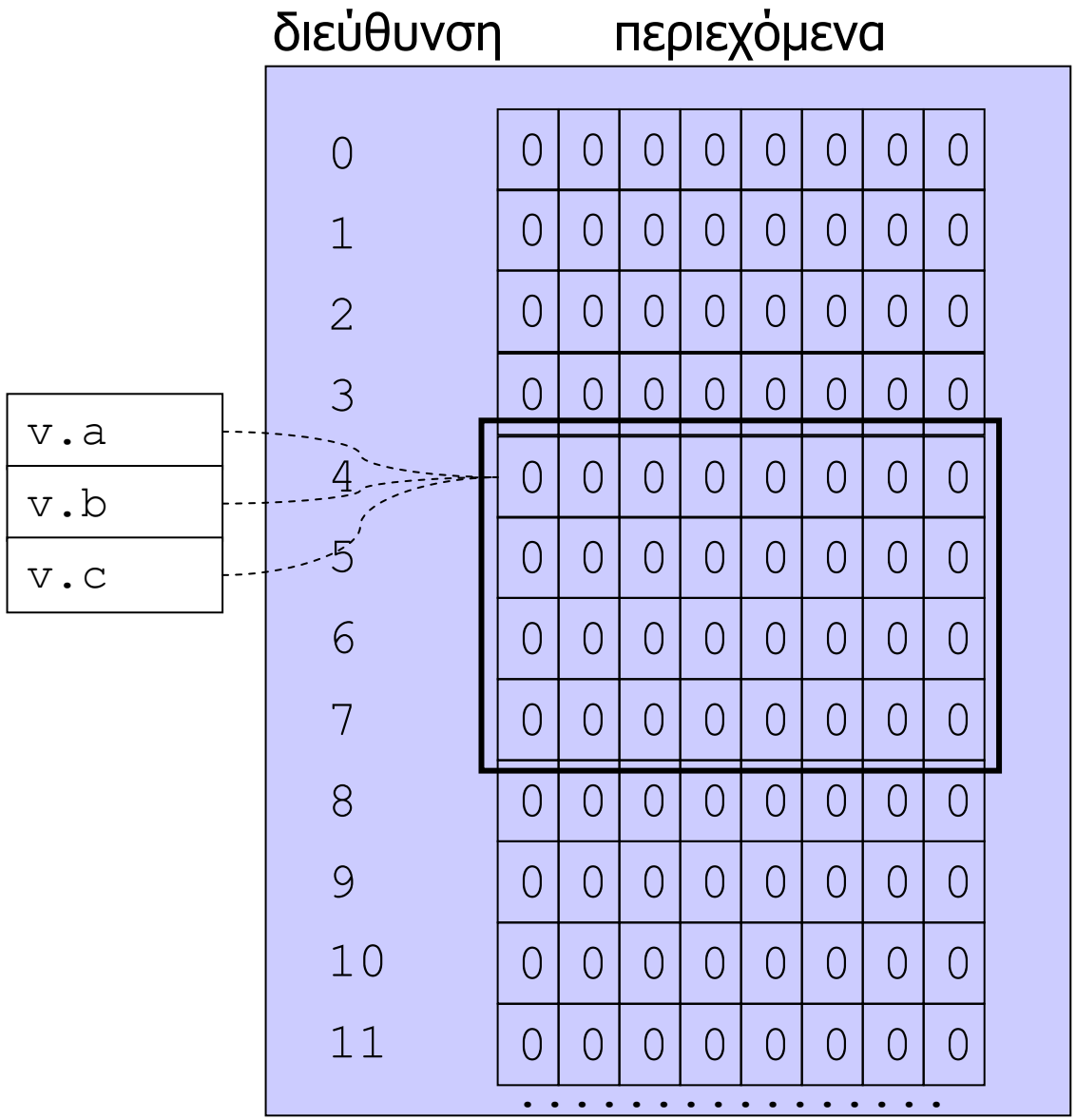
```
struct date {
  int day;
  int month;
  int year;
};
```

```
struct compact_date {
  int day:5;          /* 31 διαφορετικές τιμές */
  int month:4;        /* 12 διαφορετικές τιμές */
  int year;           /* δεν υπάρχει περιορισμός */
};
```

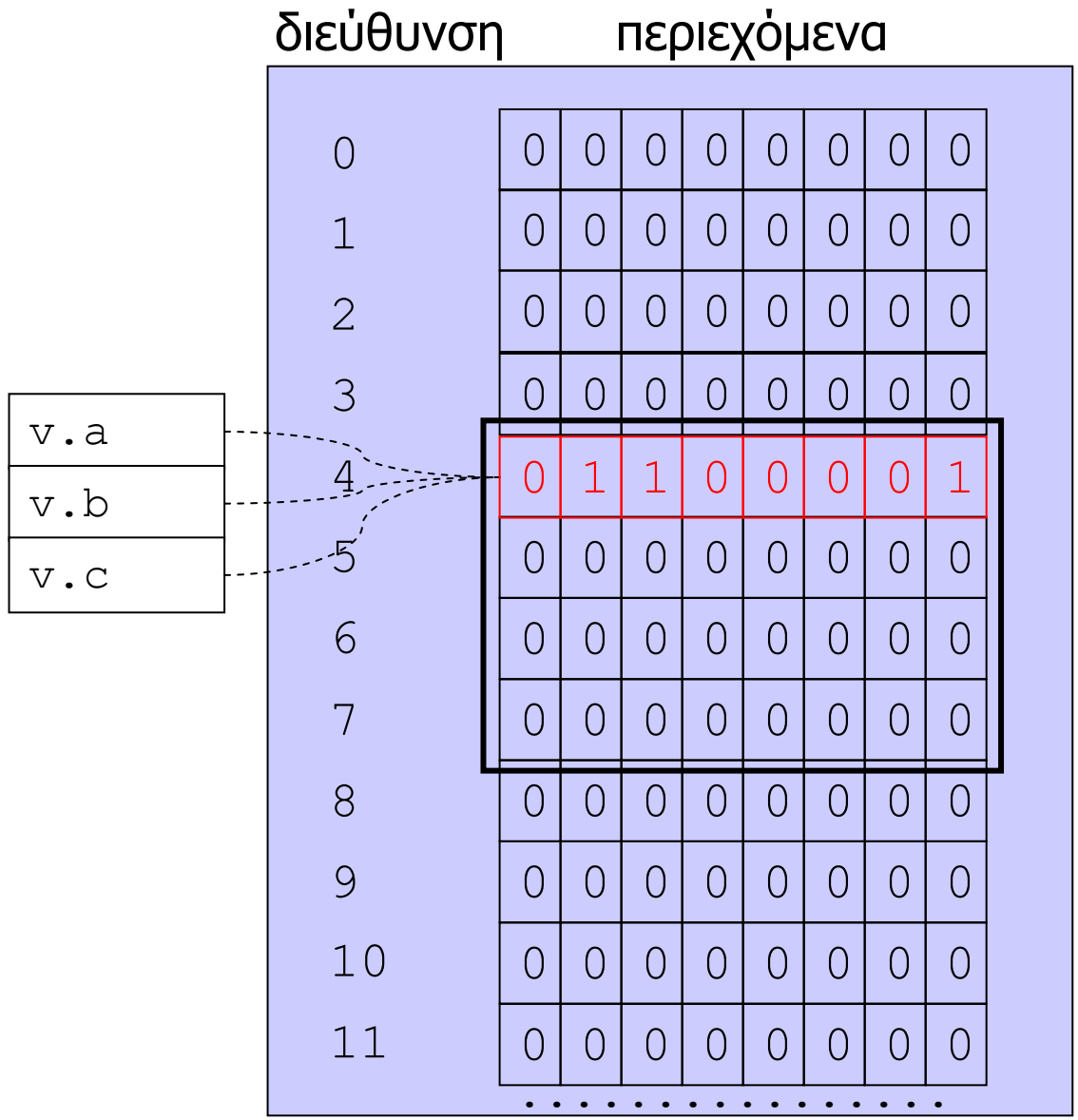
Ενώσεις - `union`

- Η ένωση (`union`) ομαδοποιεί δεδομένα διαφορετικών τύπων για την αποθήκευση των οποίων δεσμεύεται **κοινός** χώρος στη μνήμη.
- Το μέγεθος της ένωσης καθορίζεται έτσι ώστε να μπορεί να αποθηκευτεί το μεγαλύτερο πεδίο της.
- Ο προγραμματιστής είναι υπεύθυνος για την σωστή χρήση των περιεχομένων μιας ένωσης καθώς δεν υπάρχει τρόπος να διαπιστωθεί η εγκυρότητα των τιμών των πεδίων (π.χ. σε πιο πεδίο έγινε η τελευταία ανάθεση τιμής).
- Συνήθως, η ένωση χρησιμοποιείται σε συνδυασμό με την δομή – ένα πεδίο της δομής καθορίζει το πως χρησιμοποιείται η ένωση ανά πάσα χρονική στιγμή.

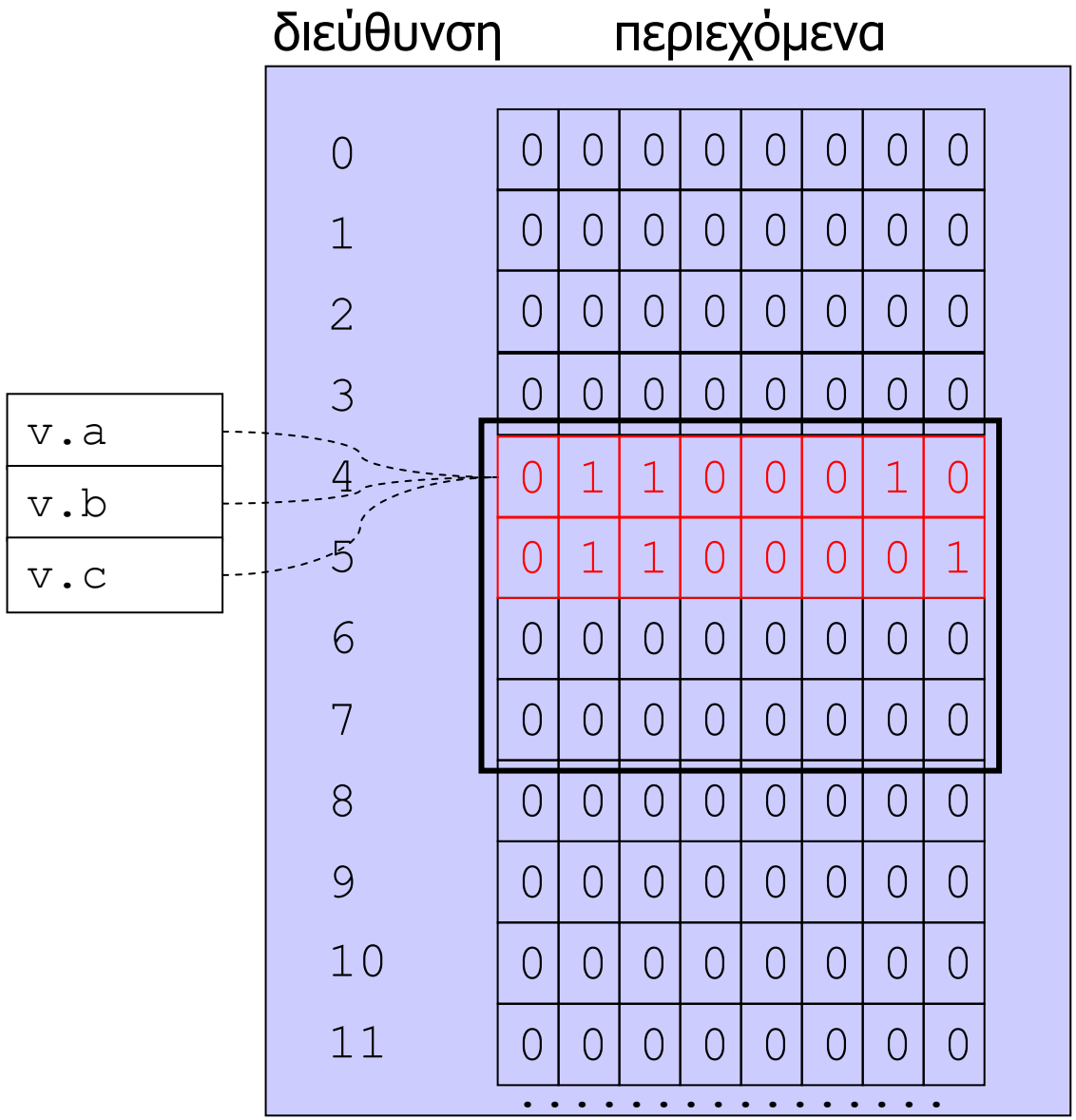
```
union abc {
   char a;
   short int b;
   int c;
};
…
union abc v;
```



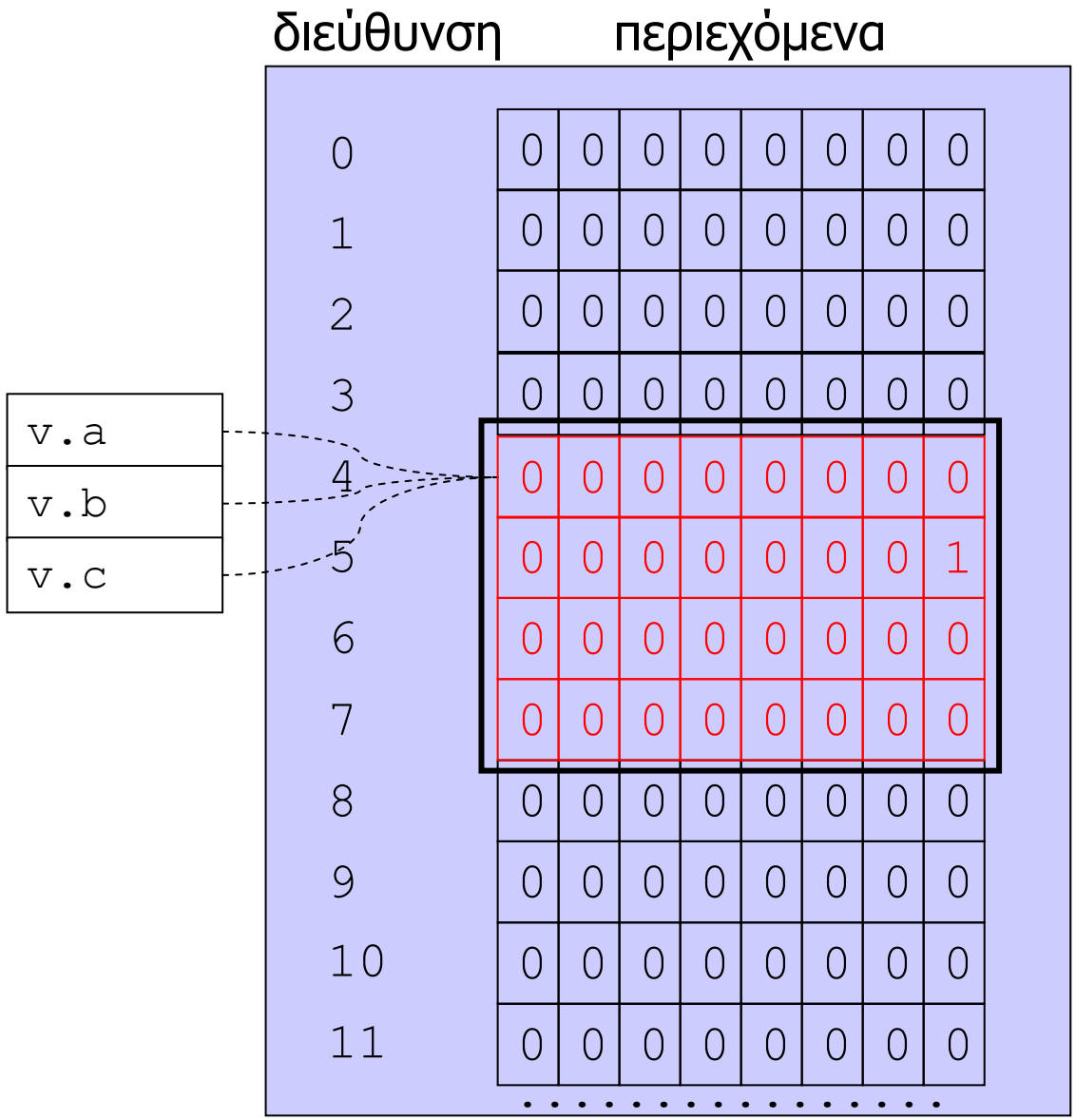
```
union abc {
   char a;
   short int b;
   int c;
};
…
union abc v;

v.a='a';
```



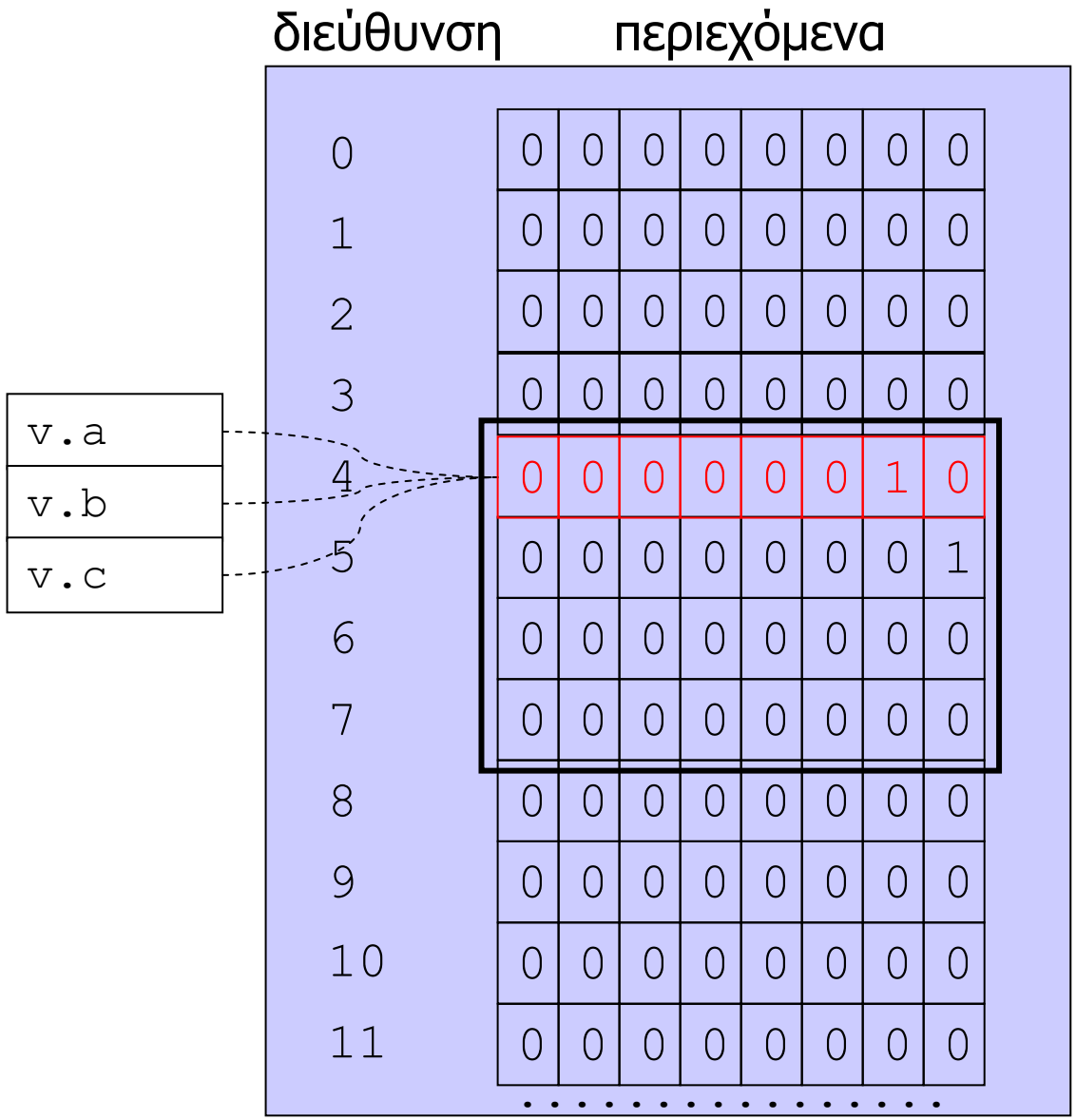
```
union abc {
   char a;
   short int b;
   int c;
};
…
union abc v;

v.a='a';

v.b=0x6162;
```



```
union abc {
   char a;
   short int b;
   int c;
};
…
union abc v;

v.a='a';

v.b=0x6162;

v.c=256;
```



```
union abc {
   char a;
   short int b;
   int c;
};
…
union abc v;

v.a='a';

v.b=0x6162;

v.c=256;

v.a=0x01;
```



```
/* δομή που αποθηκεύει τιμή σε μορφή int ή float */

struct val {
  int vtype;      /* 0 ή 1 αν ισχύει val.i ή val.f */

  union {
    int i;       /* η τιμή σε μορφή int */
    float f;     /* η τιμή σε μορφή float */
  } val;

};

…

void inc_value(struct val v) {
  switch (v.vtype) {
    case 0: { /* δούλεψε με s.val.i */ }
    case 1: { /* δούλεψε με s.val.f */ }
  }
}
```

Απαριθμήσεις - `enum`

- Με την `enum` αντιστοιχίζονται διαδοχικές ακέραιες τιμές σε ονόματα, χωρίς να ενδιαφέρει (απαραίτητα) η τιμή που δίνεται στο κάθε όνομα.
- Χρησιμοποιείται σε περιπτώσεις που μια μεταβλητή μπορεί να λάβει περιορισμένες τιμές, ιδίως όταν δεν έχει παίζει ρόλο η «απόλυτη» τιμή της μεταβλητής.
- Τα ονόματα αντιστοιχίζονται σε διαδοχικές τιμές που αυξάνουν με την σειρά που έχουν δοθεί τα ονόματα.
- Η αρίθμηση αρχίζει από το `0` εκτός και αν για κάποιο όνομα προσδιοριστεί συγκεκριμένη τιμή που οδηγεί σε διαφορετικές τιμές για τα ονόματα της απαρίθμησης.
- Μια εναλλακτική λύση είναι να οριστούν από τον προγραμματιστή σταθερές μέσω `define`.

```
enum boolean {false,true};

enum boolean b=true;

if (b) { … }
if (!b) { … }
```

```
enum weekdays {Mon=1,Tue,Wed,Thu,Fri,Sat,Sun};

enum weekdays d;

for (d=Mon; d<=Sun; d++) {
  …
}
```

Παρένθεση
(βάση δεδομένων με πίνακα από δομές)

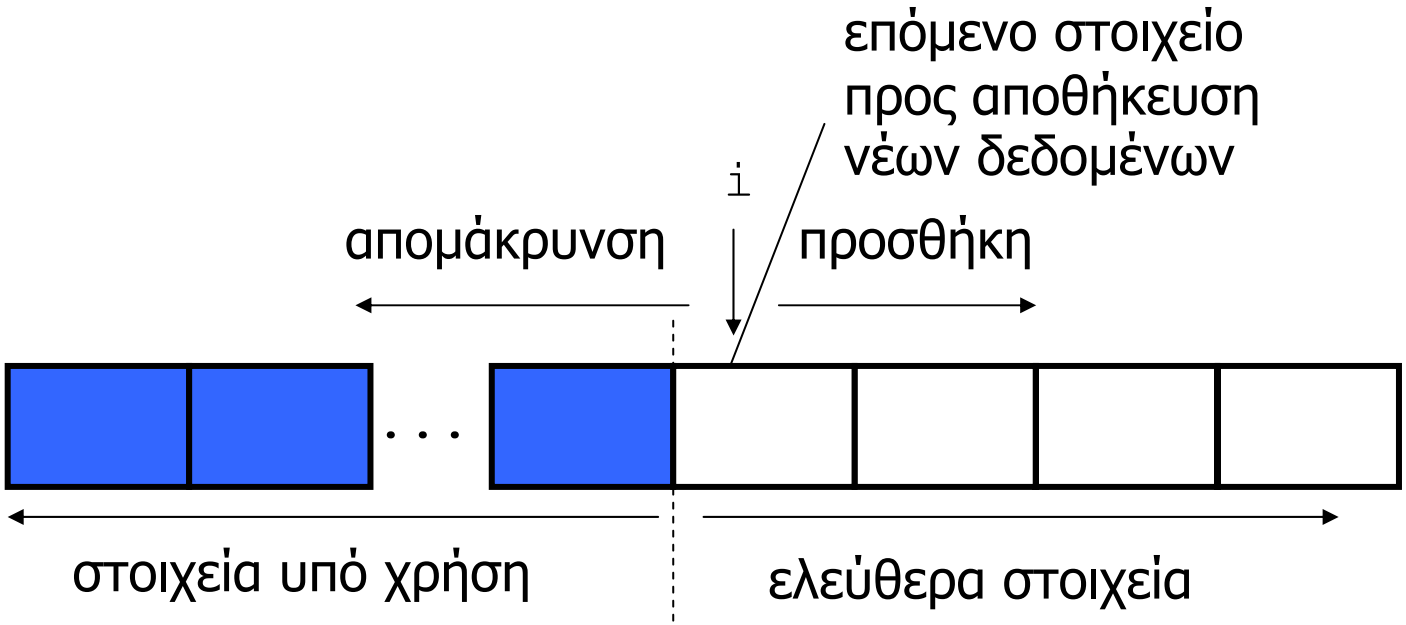
Πρόβλημα

- Ζητούμενο: επιθυμούμε να διαχειριστούμε τα περιεχόμενα της τηλεφωνικής μας αντζέντας, με βάση λειτουργίες προσθήκης, απομάκρυνσης και αναζήτησης.
- Προσέγγιση
  - ορίζουμε δομή κατάλληλη για την ομαδοποίηση των δεδομένων που ανήκουν σε μια «εισαγωγή»
  - κρατάμε τα δεδομένα σε πίνακα από δομές
- Οι λειτουργίες πρέπει να υλοποιηθούν σύμφωνα με βάση τις **εσωτερικές συμβάσεις διαχείρισης** των στοιχείων του πίνακα.

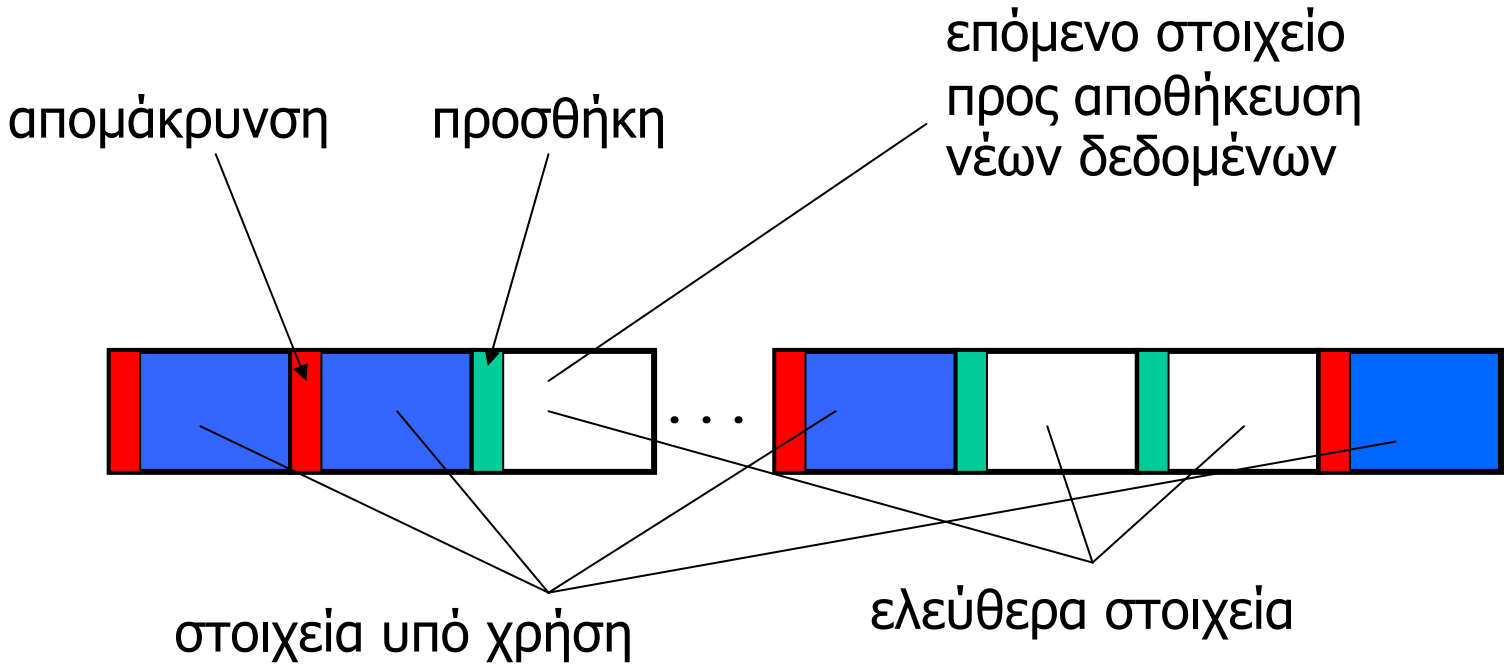
Ελεύθερα και υπό χρήση στοιχεία

- Πρέπει να γίνεται κατάλληλη διαχείριση των «υπό χρήση» / «ελεύθερων» στοιχείων, σε συνδυασμό με τις πράξεις προσθήκης, απομάκρυνσης, αναζήτησης.
- Προσέγγιση Α: όλα τα υπό χρήση (και άρα όλα τα ελεύθερα στοιχεία) βρίσκονται σε συνεχόμενες θέσεις του πίνακα και υπάρχει μεταβλητή που υποδεικνύει το σημείο διαχωρισμού (επόμενο ελεύθερο στοιχείο).
- Προσέγγιση Β: κάθε στοιχείο έχει ένα επιπλέον πεδίο μέσω του οποίου σημειώνεται κατά πόσο το στοιχείο είναι υπό χρήση ή ελεύθερο.
- Η προσέγγιση Α επιταχύνει την **αναζήτηση** ενώ η προσέγγιση Β αποφεύγει εντελώς την αντιγραφή δεδομένων κατά την **απομάκρυνση**.

Προσέγγιση Α



Προσέγγιση Β



```
void phonebook_init();
/* αρχικοποιεί τις καθολικές μεταβλητές
   ή/και δομές δεδομένων του προγράμματος */

int phonebook_find(const char name[], char phone[]);
/* δέχεται σαν πρώτη παράμετρο ένα αλφαριθμητικό όνομα και
   αποθηκεύει στην δεύτερη παράμετρο το αντίστοιχο τηλέφωνο,
   επιστρέφοντας 1 για επιτυχία και 0 για αποτυχία */

void phonebook_rmv(const char name[]);
/* δέχεται σαν παράμετρο ένα αλφαριθμητικό όνομα και
   "απομακρύνει" την αντίστοιχη εγγραφή, αν υπάρχει */

int phonebook_add(const char name[], const char phone[]);
/* επιχειρεί να εισάγει μια νέα εγγραφή με το όνομα και
   τηλέφωνο που δίνονται σαν παράμετροι, και επιστρέφει
   1 για επιτυχία, 0 για αποτυχία λόγω έλλειψης χώρου
   και -1 για αντικατάσταση υπάρχουσας εγγραφής */

int main(int argc, char *argv[]);
/* διάλογος με το χρήστη */
```

```
int main(int argc, char *argv[]) {
  int s,res; char name[64],phone[64];
  phonebook_init();
  do {
    printf("1. Add\n"); printf("2. Remove\n");
    printf("3. Find\n"); printf("4. Exit\n");
    printf("> "); scanf("%d",&s);
    switch (s) {
      case 1: {
        printf("name & phone:"); scanf("%63s %63s",name,phone);
        res=phonebook_add(name,phone); printf("res=%d\n",res);
        break;
      }
      case 2: {
        printf("name:"); scanf("%63s",name);
        phonebook_rmv(name);
        break;
      }
      case 3: {
        printf("name:"); scanf("%63s",name);
        res=phonebook_find(name,phone); printf("res=%d\n",res);
        if (res) { printf("phone: %s\n",phone); }
        break;
      }
    }
  } while (s!=4);
}
```

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

#define N 100

struct entry {
  char used;       /* 1 υπό χρήση, 0 ελεύθερο */
  char name[64];   /* όνομα ως αλφαριθμητικό */
  char phone[64]; /* τηλέφωνο ως αλφαριθμητικό */
};

struct entry entries[N];

void phonebook_init() {
  int i;
  for (i=0; i<N; i++) { entries[i].used=0; }
}
```

```
int internal_find(const char name[]) {
  int i;
  for (i=0; (i<N) && ((!entries[i].used) ||
                      (strcmp(entries[i].name,name))); i++);
  return(i);
}

int phonebook_find(const char name[], char phone[]) {
  int i;

  i=internal_find(name);
  if (i==N) { return(0); }
  else { strcpy(phone,entries[i].phone); return(1); }
}

void phonebook_rmv(const char name[]) {
  int i;

  i=internal_find(name);
  if (i<N) {
    entries[i].used=0;
  }
}
```

```
int phonebook_add(const char name[], const char phone[]) {
  int i;

  i=internal_find(name);

  if (i<N) {
    strcpy(entries[i].phone,phone);
    return(-1);  /* replace */
  }

  for (i=0; (i<N) && (entries[i].used); i++);
  if (i==N) {
    return(0);  /* no free space */
  }

  strcpy(entries[i].name,name);
  strcpy(entries[i].phone,phone);
  entries[i].used=1;
  return(1); /* done */

}
```

Παρένθεση
(βάση δεδομένων με πίνακα)

Μέγεθος αντικειμένων - `sizeof`

- Η συνάρτηση `sizeof` επιστρέφει το μέγεθος σε bytes που καταλαμβάνει η μεταβλητή ή ο τύπος δεδομένων που δίνουμε σαν παράμετρο.
- **Δεν επιστρέφει απαραίτητα την ίδια τιμή για διαφορετικούς μεταφραστές ή διαφορετικές αρχιτεκτονικές επεξεργαστών!**
- Μπορεί να υιοθετούνται διαφορετικές προσεγγίσεις για την δέσμευση μνήμης δομών `struct` και `union` ή/και να διαφέρει το μέγεθος των βασικών τύπων.
- Με το `sizeof` μπορούν να ορισθούν παραμετρικές εκφράσεις ως προς το μέγεθος των τύπων (σύνθετων και μη) που χρησιμοποιεί το πρόγραμμα.

```
#include <stdio.h>

struct entry {
  char used;       /* 1 υπό χρήση, 0 ελεύθερο */
  char name[64];   /* όνομα ως αλφαριθμητικό */
  char phone[64]; /* τηλέφωνο ως αλφαριθμητικό */
};

int main(int argc, char *argv[]) {
  struct entry e;

  printf("size of struct entry is %d\n",sizeof(struct entry));

  printf("size of variable e is %d\n",sizeof(e));

}
```

Η χρήση της `typedef`

- Με την `typedef` δηλώνονται **συμβολικά** ονόματα για τύπους που κατασκευάζει ο προγραμματιστής.
- Το συντακτικό είναι πανομοιότυπο με τον τρόπο που δηλώνονται μεταβλητές, βάζοντας το προσδιορισμό `typedef` στην αρχή της δήλωσης – το όνομα που αφορά η δήλωση είναι το όνομα του νέου τύπου.
- Χρησιμοποιούμε την `typedef` κυρίως για να διευκολύνουμε την αναγνωσιμότητα του κώδικα.
- Η `typedef` μπορεί να χρησιμοποιηθεί και για την (συντακτική) «απόκρυψη» της υλοποίησης ενός τύπου δεδομένων – έτσι ώστε ο προγραμματιστής να χρησιμοποιεί αντικείμενα ενός τύπου την υλοποίηση του οποίου δεν (είναι απαραίτητο να) γνωρίζει.

```
int a;                    /* μεταβλητή int */

int *b;                   /* μεταβλητή δείκτης σε int */

typedef int *IntPtr;      /* τύπος δείκτης σε int */

IntPtr c;                 /* μεταβλητή IntPtr */

…

a = 10;

b = &a;

*b = 11;

c = b;

*c = 12;
```

```
struct date {                   /* τύπος struct date */
  int day;
  int month;
  int year;
};

typedef struct date Date;   /* τύπος Date */

struct date d1;             /* μεταβλητή struct date */

Date d2;                    /* μεταβλητή Date */

…
d1.day = 30;
d1.month = 11;
d1.year = 2006;

d2 = d1;
d2.day++;
d2.month++;
```

Δείκτες σε δομές δεδομένων

- Η έννοια του δείκτη εφαρμόζεται και σε σύνθετους τύπους – μια μεταβλητή μπορεί να οριστεί ως δείκτης σε `struct` ή δείκτης σε `union`.
- Πρόσβαση στα πεδία των αντικειμένων τύπου `struct` και `union` μέσω μεταβλητής δείκτη επιτυγχάνεται μέσω του τελεστή `->`
- Ένας πίνακας από αντικείμενα `struct/union` θεωρείται σταθερός δείκτης στο πρώτο στοιχείο του πίνακα, και ένας δείκτης σε `struct/union` μπορεί να θεωρηθεί ως η αρχή ενός τέτοιου πίνακα.
- Ισχύουν οι παρατηρήσεις που έχουν γίνει για δείκτες σε βασικούς τύπους: μη ελεγχόμενη πρόσβαση μνήμης, αριθμητικές πράξεις με δείκτες κλπ.

```
struct date {
  int day,month,year;
};

typedef struct date *datePtr;

struct date d[2];
datePtr dp;
…
d[0].day = 1;
d[0].month = 1;
d[0].year = 2007;
d[1] = d[0];
d[1].day = 31;

dp = d;                 /* dp is address of d[0] */
dp->year++;             /* d[0].year is 2008 */
(*dp).month = 12;       /* d[0].month is 12 */
dp++;                   /* dp is address of d[1] */
dp->year++;             /* d[1].year is 2008 */
(*dp).year++;           /* d[1].year is 2009 */
```

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

#define N 100

typedef struct {
  char used;
  char name[64];
  char phone[64];
} Entry;

typedef Entry *EntryPtr;

Entry book[N];

void phonebook_init() {
  EntryPtr p;

  for (p=book; p<book+N; p++) { p->used=0; }
}
```