Δυναμική δέσμευση και αποδέσμευση μνήμης

Γιατί χρειάζεται η δυναμική μνήμη;

- Οι απαιτήσεις του προγράμματος σε μνήμη μπορεί να είναι **άγνωστες** την ώρα της συγγραφής του κώδικα
- Αν για «ασφάλεια» χρησιμοποιήσουμε έναν τεράστιο αριθμό μεταβλητών ή τεράστιο στατικό πίνακα για να αποθηκεύσουμε τα δεδομένα, πιθανώς αυτή η μνήμη να μην χρησιμοποιηθεί ποτέ
    - η μνήμη ουσιαστικά **χάνεται** καθώς δεν μπορεί να διατεθεί σε άλλα προγράμματα που εκτελούνται στον υπολογιστή
- Αν για «οικονομία» χρησιμοποιήσουμε ένα μικρότερο πίνακα, πιθανώς η μνήμη να μην φτάσει
    - το πρόγραμμα είτε θα αναγκαστεί να **πετάξει/χάσει** κάποια δεδομένα είτε θα αναγκαστεί να τερματίσει

Σχετικές συναρτήσεις / βιβλιοθήκες

```
#include <stdlib.h>

void *malloc(size_t size);
void *calloc(size_t n, size_t size);
void *realloc(void *p, size_t size);
void free(void *p);
```

```
#include <string.h>

void *memcpy(void *dst, const void *src, size_t n);
void *memset(void *p, int c, size_t n);
int memcmp(const void *p1, const void *p2, size_t n);

char *strdup(const char *s);
```

Βασικές λειτουργίες

- `void *malloc(size_t size);`
  - δεσμεύει ένα συνεχές κομμάτι μνήμης `size` bytes
  - επιστρέφει έναν δείκτη στην αρχή του κομματιού
  - τα περιεχόμενα της μνήμης **δεν αρχικοποιούνται**

- `void free(void *ptr);`
  - αποδεσμεύει το κομμάτι μνήμης που δείχνει ο δείκτης `ptr`
  - ο δείκτης πρέπει να έχει επιστραφεί από κάποια προηγούμενη λειτουργία δέσμευση μνήμης – **δεν** επιτρέπεται/προβλέπεται να δοθεί ως παράμετρος μια οποιαδήποτε τυχαία διεύθυνση

Βασικές λειτουργίες (2)

- `void *calloc(size_t nmemb, size_t size);`
  - δεσμεύει ένα συνεχές κομμάτι μνήμης `nmemb` x `size` bytes
  - επιστρέφει έναν δείκτη στην αρχή του κομματιού
  - τα περιεχόμενα της μνήμης **αρχικοποιούνται** σε `0`

- `void *realloc(void *ptr, size_t size);`
  - προσαρμόζει στα `size` bytes το κομμάτι μνήμης που δείχνει ο δείκτης `ptr` (το κομμάτι επεκτείνεται ή κόβεται, αναλόγως)
  - τα προηγούμενα περιεχόμενα της μνήμης **δεν αλλάζουν**
  - τυχόν επιπλέον bytes **δεν αρχικοποιούνται**
  - πιθανή **αντιγραφή** των δεδομένων σε άλλη περιοχή της μνήμης, με απελευθέρωση του προηγούμενου κομματιού

Χρήση δυναμικής μνήμης

- Η διεύθυνση που επιστρέφεται χρησιμοποιείται με αποκλειστική ευθύνη του προγραμματιστή
  - οι λειτουργίες της δυναμικής μνήμης **δεν γνωρίζουν τίποτα** σχετικά με τους τύπους των δεδομένων του προγράμματος
- Συνήθως η μνήμη που δεσμεύεται αντιστοιχεί στο (πολλαπλάσιο) μέγεθος ενός αντικειμένου τύπου `T`
- Η διεύθυνση που επιστρέφεται, ανατίθεται (με **type casting**) σε μια μεταβλητή τύπου `T*`
  - για την σωστή αποθήκευση/ερμηνεία των δεδομένων
- Ο προγραμματιστής πρέπει να αποδεσμεύει την δυναμική μνήμη που δεν χρησιμοποιεί το πρόγραμμα
  - διαφορετικά υπάρχει περίπτωση η επόμενη αίτηση δέσμευσης να αποτύχει – το πρόγραμμα να «ξεμείνει» από μνήμη

Έλεγχος τιμής που επιστρέφεται

- Οι `malloc` και `realloc` επιστρέφουν `NULL` αν δεν μπορεί να δεσμευτεί όση μνήμη ζητήθηκε

- Η τιμή επιστροφής **πρέπει να ελέγχεται**

- Διαφορετικά, αν η τιμή `NULL` ανατεθεί σε δείκτη, η επόμενη αναφορά στη μνήμη μέσω του δείκτη θα οδηγήσει σε τερματισμό του προγράμματος
  - μπορεί να μην επιθυμούμε ένα τέτοιο «απότομο» τερματισμό (και χάσιμο δεδομένων του προγράμματος)
  - δεν γνωρίζουμε σε ποιο σημείο του προγράμματος έγινε αυτή η αναφορά (υπάρχει πάντα το debugging)

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

int main(int argc, char *argv[]) {
  int a,b,*sum;

  scanf("%d %d",&a,&b);

  sum = (int *) malloc( sizeof(int) );

  if (sum == NULL) {
    printf("no memory\n");
    return(1);
  }

  *sum = a+b;

  printf("%d\n",*sum);
  free(sum);
  return(0);
}
```

δέσμευση μνήμης από `sizeof(int)` bytes

type cast σε δείκτη-σε-ακέραιο

αποδέσμευση μνήμης

Μονιμότητα δυναμικής μνήμης

- Η δυναμική μνήμη είναι **μόνιμη**, δηλαδή υφίσταται καθ' όλη τη διάρκεια της εκτέλεσης του προγράμματος
- Δυναμική μνήμη που δεσμεύεται μέσα από μια συνάρτηση **παραμένει εν ισχύ** μετά την κλήση της

- Άλλος ένας τρόπος επιστροφής αποτελεσμάτων μέσα από το (προσωρινό) περιβάλλον κλήσης συνάρτησης
  - η συνάρτηση δεσμεύει δυναμική μνήμη όπου αποθηκεύει τα δεδομένα, και επιστρέφει σαν αποτέλεσμα την διεύθυνση
  - το περιβάλλον κλήσης χρησιμοποιεί τη διεύθυνση όπως απαιτείται – ανάθεση σε κατάλληλη μεταβλητή δείκτη
  - το περιβάλλον κλήσης πρέπει να αποδέσμευσει την μνήμη μετά την χρήση

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

int *add(int a, int b) {
  int *sum = (int *)malloc(sizeof(int));
  *sum = a + b;
  return(sum);
}

int main(int argc, char *argv[]) {
  int a, b, *sum;

  scanf("%d %d", &a, &b);

  sum = add(a, b);

  printf("%d\n", *sum);
  free(sum);
  return(0);
}
```

```c
#include <stdio.h>

int *add(int a, int b) {
  int sum;
  sum = a + b;
  return(&sum); /* αυτό είναι λάθος ! */
}

int main(int argc, char *argv[]) {
  int a, b, *sum;

  scanf("%d %d",&a, &b);

  sum = add(a, b);

  printf("%d\n", *sum);
  return(0);
}
```

γιατί είναι λάθος;

Δυναμικοί πίνακες

Δυναμικοί πίνακες

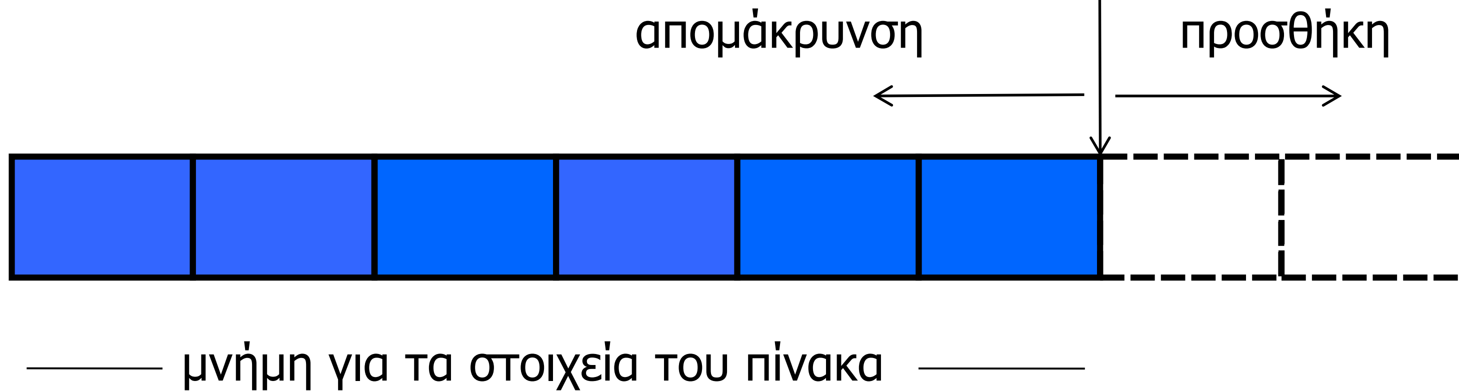
- Κλασικό παράδειγμα χρήσης δυναμικής μνήμης

1. Δεσμεύεται με `malloc` ή `realloc` δυναμική μνήμη μεγέθους `n*sizeof(T)`
   - όπου `T` ο τύπος δεδομένων των στοιχείων του πίνακα

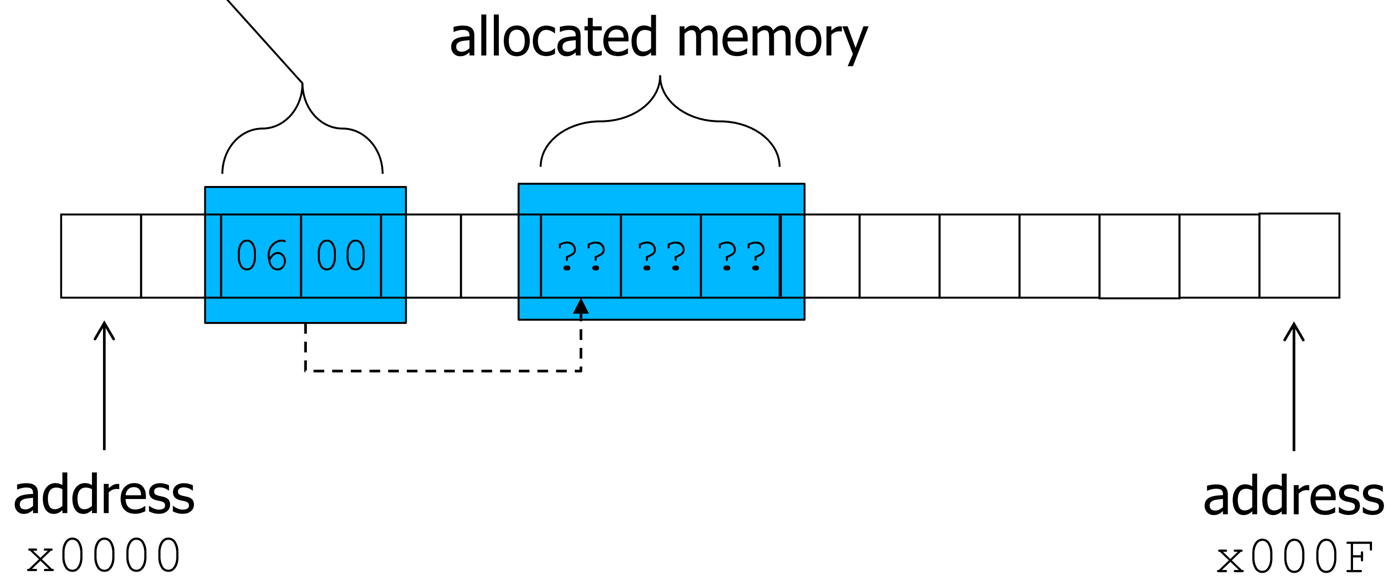
2. Αν το `n` αποδειχθεί μικρό ή μεγάλο, χρησιμοποιείται η `realloc` για την επέκταση/κόψιμο του πίνακα.

3. Όταν ο πίνακας δεν χρειάζεται άλλο, η μνήμη που καταλαμβάνει μπορεί να αποδεσμευθεί με `free`

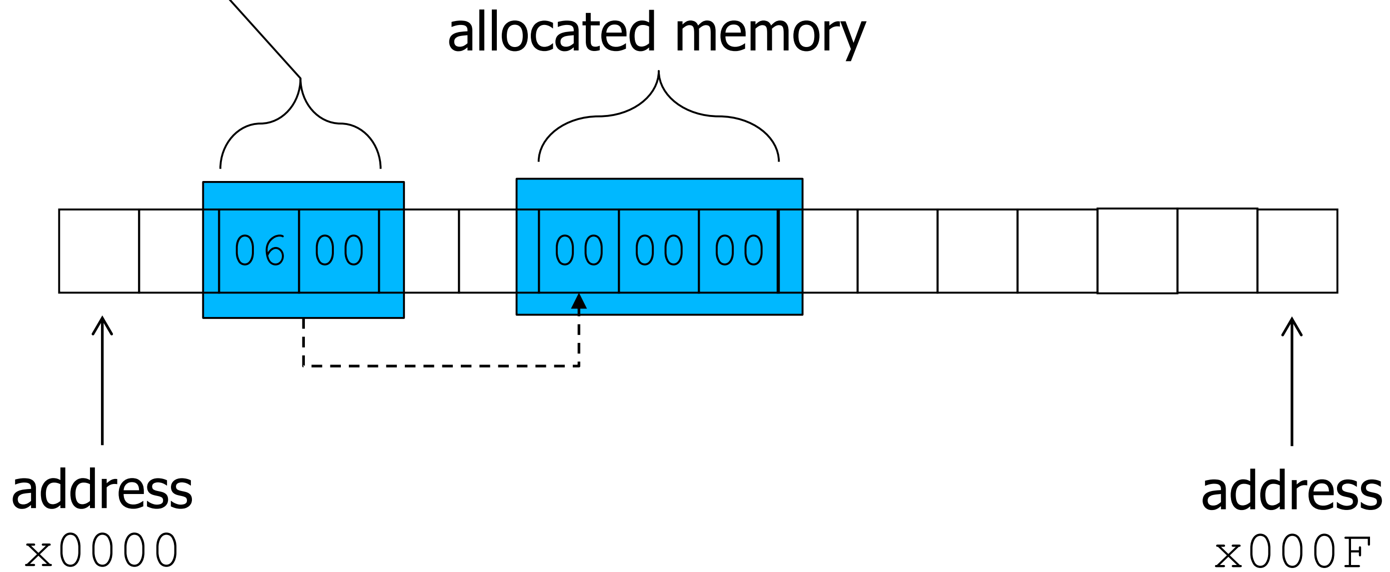
το μέγεθος του πίνακα αλλάζει **δυναμικά** ως συνέπεια των λειτουργιών προσθήκης ή/και απομάκρυνσης δεδομένων



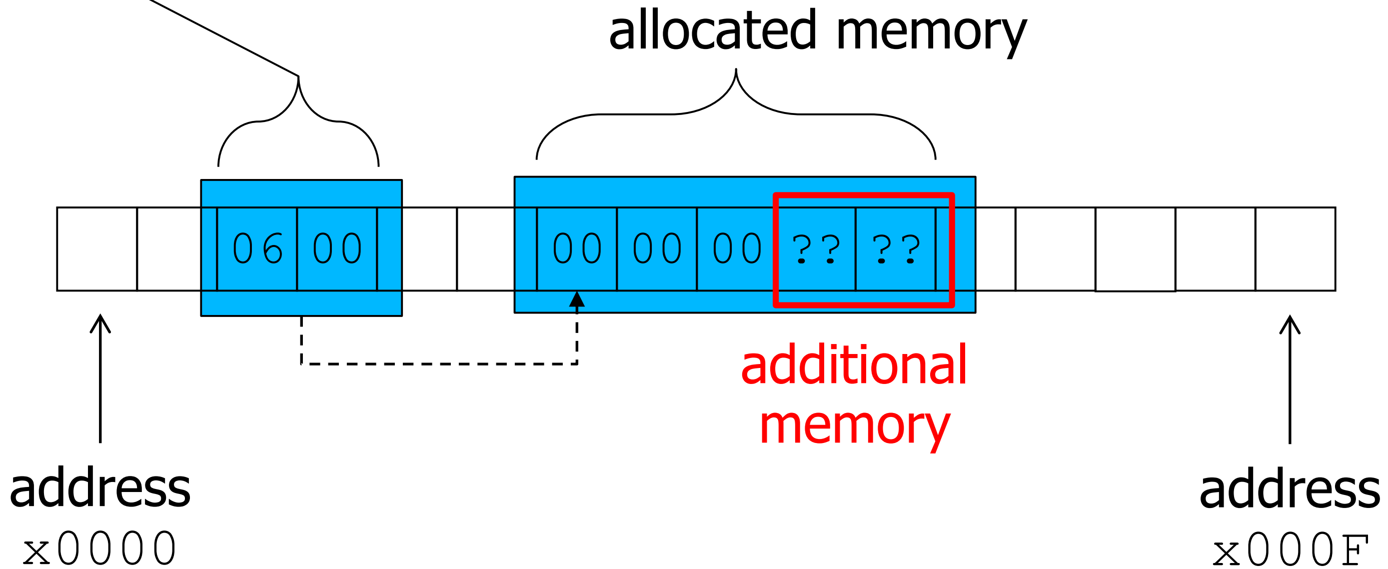
little endian



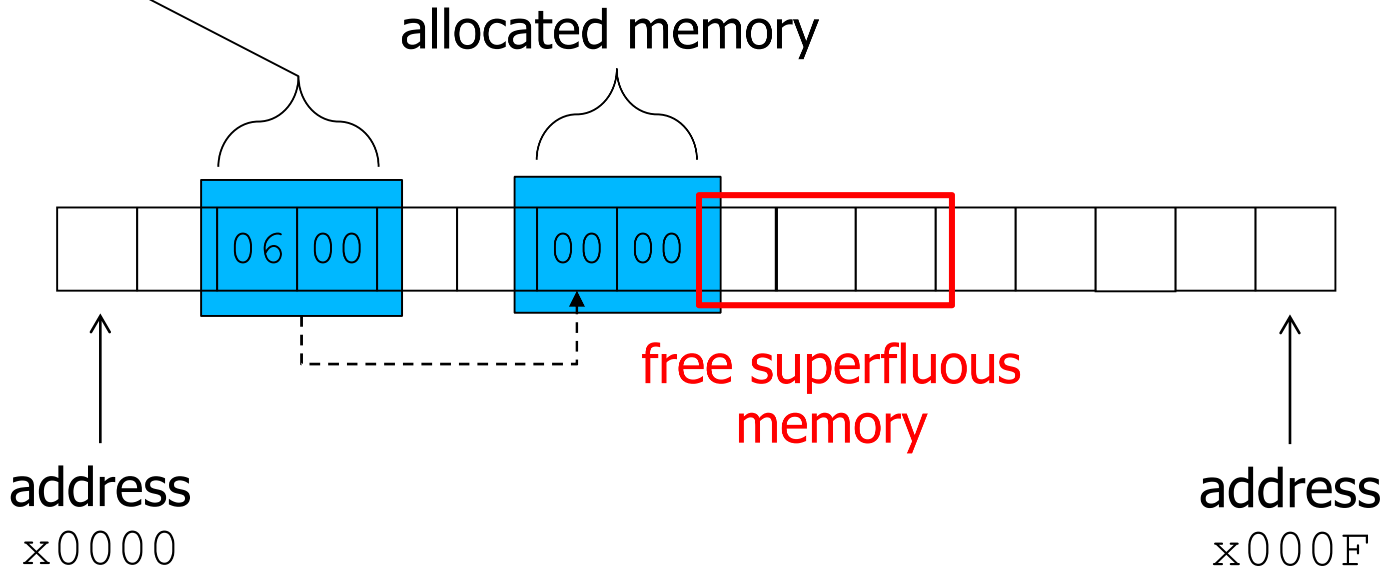
little endian



little endian

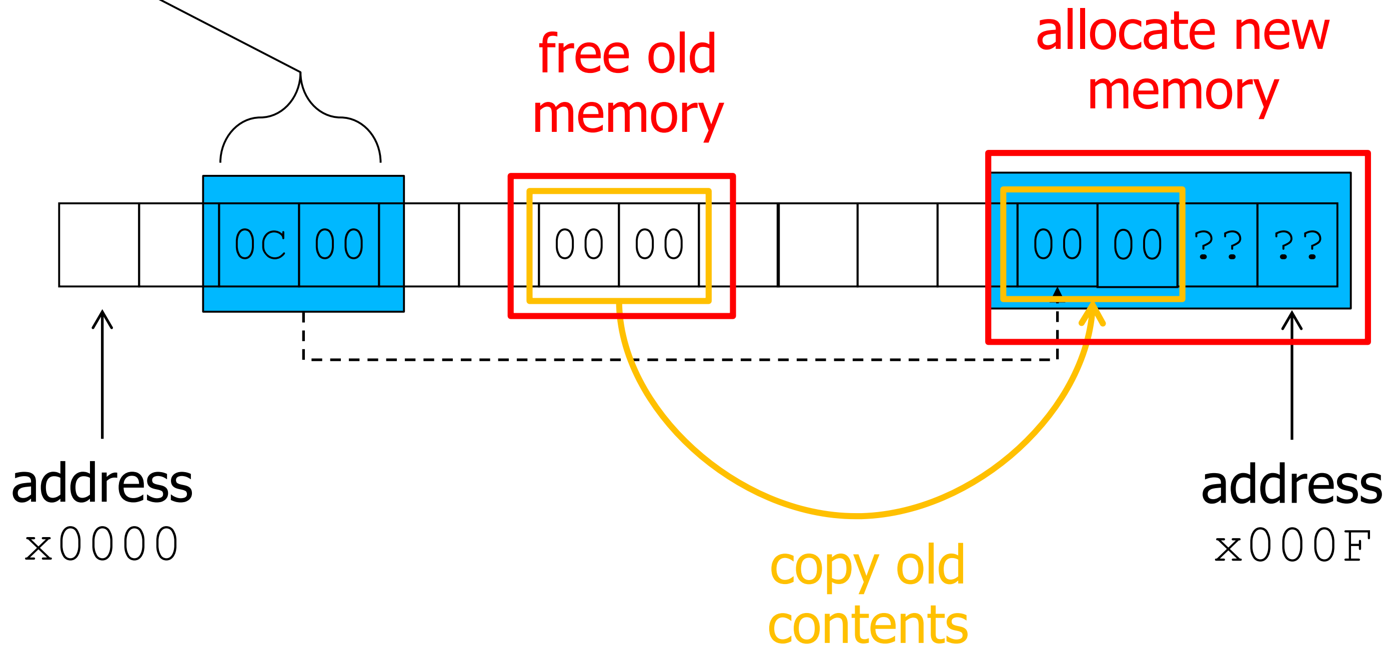


little endian



little endian

```
*str = (char *) realloc(str,4*sizeof(char));
```



Παράδειγμα: τηλεφωνική ατζέντα

- Επιθυμούμε να διαχειριστούμε τα περιεχόμενα της τηλεφωνικής μας ατζέντας, με αντίστοιχες λειτουργίες **προσθήκης**, **απομάκρυνσης** και **αναζήτησης**

- Ορίζουμε κατάλληλη δομή για την ομαδοποίηση των δεδομένων που αποτελούν μια εγγραφή
- Διατηρούμε ένα δυναμικό πίνακα από τέτοιες δομές

- Οι λειτουργίες πρέπει να υλοποιηθούν σύμφωνα με κατάλληλες **εσωτερικές συμβάσεις** για την **διαχείριση** των στοιχείων του πίνακα

```
phonebook *phonebook_create();

int phonebook_find(phonebook *b, const char name[],
                                 char phone[]);

int phonebook_add(phonebook *b, const char name[],
                                const char phone[]);

void phonebook_rmv(phonebook *b, const char name[]);

void phonebook_destroy(phonebook *b);
```

```
typedef struct {
  char name[64];
  char phone[64];
} entry;

typedef struct {
  entry* entries; /* dynamic table holding the entries */
  int size; /* the current size of the table */
} phonebook;
```

```
phonebook *phonebook_create() {
  phonebook *b;

  b = (phonebook *) malloc(sizeof(phonebook));
  b->size = 0;
  return(b);
}
```

```
void phonebook_destroy(phonebook *b) {
  if (b->size>0) {
    free(b->entries);
  }
  free(b);
}
```

```
int find0(phonebook *b, const char name[]) {
  int i;

  for (i=0; (i < b->size) &&
            (strcmp(b->entries[i].name,name)); i++);
  return(i);
}
```

```
int phonebook_find(phonebook *b, const char name[],
                                 char phone[]) {
  int pos;

  pos = find0(b,name);

  if (pos == b->size)
    return(0); /* not found */
  else {
    strcpy(phone,b->entries[pos].phone);
    return(1); /* found */
  }
}
```

```
int phonebook_add(phonebook *b, const char name[],
                                const char phone[]) {
  int pos;

  pos = find0(b,name);

  if (pos < b->size) {
    strcpy(b->entries[pos].phone,phone);
    return(-1);  /* replaced */
  }

  b->size++;
  b->entries = (entry *)
              realloc(b->entries,b->size*sizeof(entry));
  strcpy(b->entries[b->size-1].name,name);
  strcpy(b->entries[size-1].phone,phone);

 return(1); /* added */
}
```

επέκταση

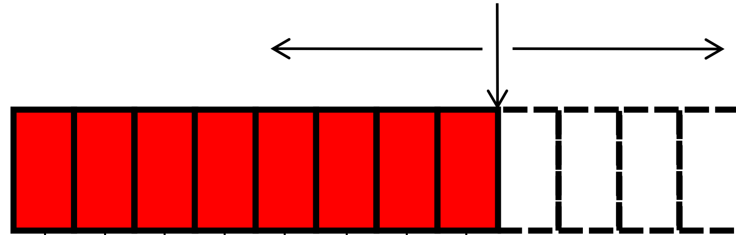
```
void phonebook_rmv(phonebook *b, const char name[]) {
  int pos;

  pos = find0(name);

  if (pos < b->size) {
    memcopy(&b->entries[pos],
            &b->entries[b->size-1],sizeof(entry));
    b->size--;
    b->entries = (entry *)
                 realloc(b->entries,b->size*sizeof(entry));
  }

}
```

κόψιμο

Παρατήρηση

- Κάθε φορά που απομακρύνεται ένα στοιχείο, γίνεται μια **αντιγραφή** δεδομένων
  - από την τελευταία θέση, στην θέση που ελευθερώθηκε
- Αυτό μπορεί να είναι ιδιαίτερα χρονοβόρο
  - αν τα περιεχόμενα της δομής είναι μεγάλα σε μέγεθος
- Μπορούμε να αποφύγουμε αυτή την αντιγραφή, χρησιμοποιώντας ένα **πίνακα από δείκτες** σε (αυτόνομα) αντικείμενα δεδομένων η μνήμη των οποίων δεσμεύεται και αποδεσμεύεται δυναμικά κατά την προσθήκη / απομάκρυνση τους
- Οι «αντιμεταθέσεις» στοιχείων γίνονται γρήγορα, **ανεξάρτητα** από το μέγεθος των αντικειμένων

memory for the
table of pointers to
data records



memory for
individual data
record