Προγραμματισμός II (ECE116)

#2 δυναμικά προσαρμοζόμενοι πίνακες

Σπύρος Λάλης
lalis@uth.gr

Δυναμικοί πίνακες

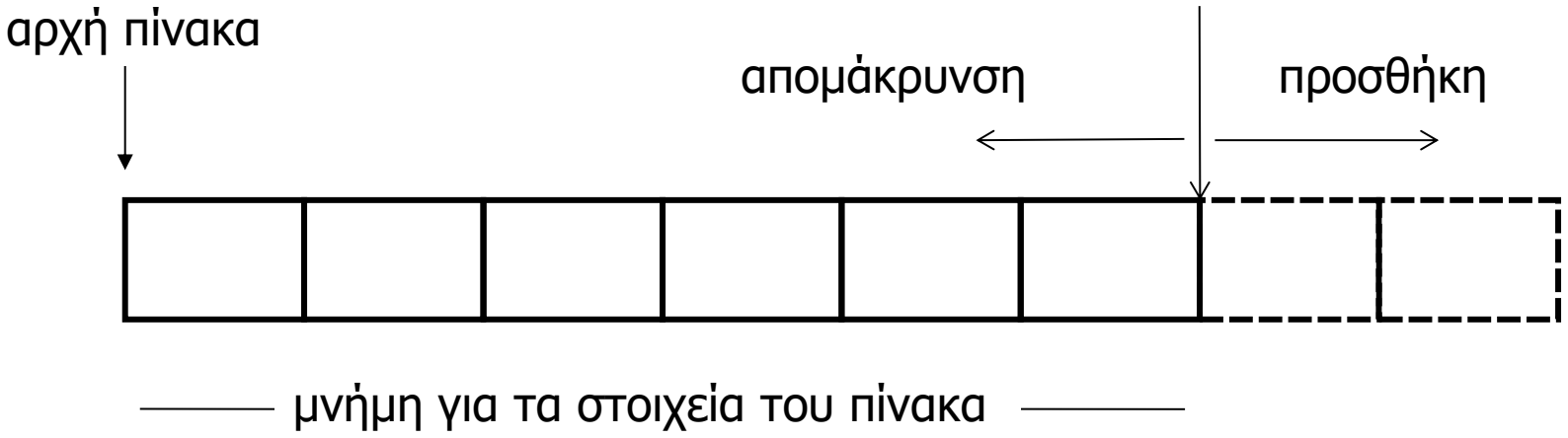
- Κλασικό παράδειγμα χρήσης δυναμικής μνήμης

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

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 αποδεσμεύεται με `free`

προσαρμόζουμε με recalloc το μέγεθος του πίνακα **δυναμικά** ανάλογα με τις προσθήκες / απομακρύνσεις εγγραφών που γίνονται κατά την εκτέλεση



Παράδειγμα: τηλεφωνικός κατάλογος

- Αρχίζουμε ορίζοντας την **προγραμματιστική διεπαφή** της λειτουργικότητας

1. Δομή για την ομαδοποίηση όλων των δεδομένων που αποτελούν μια εγγραφή
   - μπορεί να περιέχει αρκετή πληροφορία/πεδία
2. Δομή για την ομαδοποίηση της πληροφορίας που αφορά τον ίδιο τον κατάλογο
   - μπορεί να θέλουμε να διατηρούμε περισσότερους
3. Συναρτήσεις που παρέχουν την λειτουργικότητα: **προσθήκη**, **αφαίρεση**, **αναζήτηση**

- Κρατάμε την προγραμματιστική διεπαφή όσο γίνεται πιο **ανεξάρτητη** από την «εσωτερική» υλοποίηση

```
typedef struct {
  char name[64];  /* name of person */
  char phone[64]; /* phone as string for flexibility */
  …
} entry_t;

typedef struct {
  …
} phonebook_t;
```

οποιαδήποτε επιπλέον δεδομένα (δεν έχουν σημασία για το παράδειγμα)

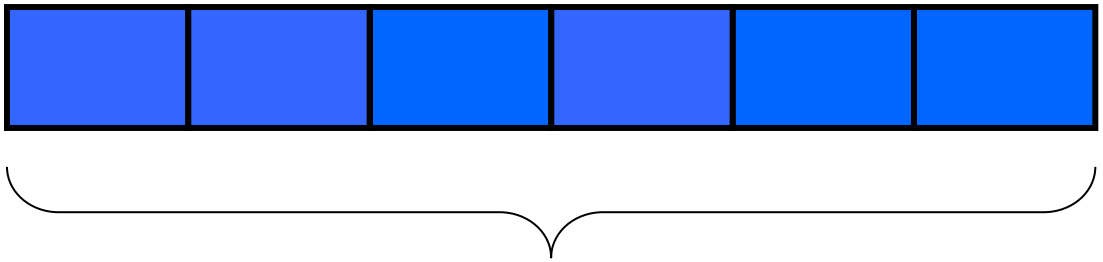
τα ακριβή περιεχόμενα θα οριστούν όταν αποφασίσουμε το πως θα λειτουργεί η εσωτερική υλοποίηση

```
/* αρχικοποιεί τον κατάλογο ως κενό/άδειο */
void phonebook_init(phonebook_t *pb);

/* προσθήκη εγγραφής με περιεχόμενα e,
   επιστρέφει 1 αν καταχωρηθεί μια νέα εγγραφή,
   2 για αλλαγή υπάρχουσας εγγραφής,
   0 για αποτυχία (δεν υπάρχει χώρος αποθήκευσης) */
int phonebook_add(phonebook_t *pb, const entry_t *e);

/* αναζήτηση με το όνομα name και αντιγραφή εγγραφής στο e,
   επιστρέφει 1 για επιτυχία, 0 για αποτυχία */
int phonebook_find(const phonebook_t *pb,
                   const char *name, entry_t *e);

/* αφαίρεση της εγγραφής με το όνομα name,
   επιστρέφει 1 για επιτυχία, 0 για αποτυχία */
int phonebook_rmv(phonebook_t *pb, const char *name);

/* διαγράφει όλα τα περιεχόμενα του καταλόγου */
void phonebook_clear(phonebook_t *pb);
```

Υλοποίηση με δυναμικό πίνακα

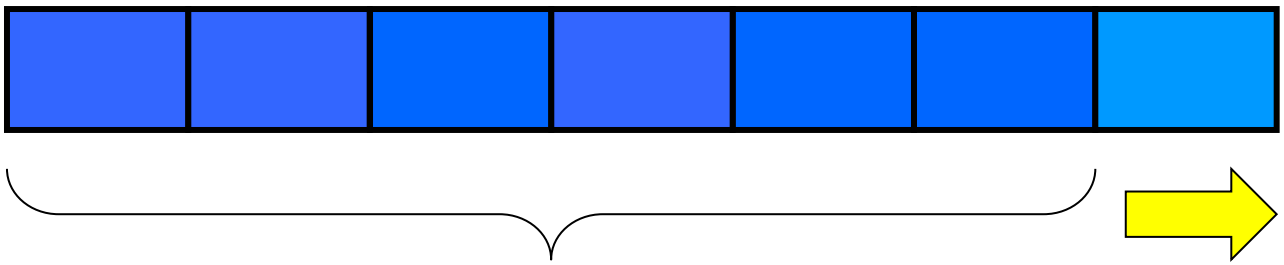
- **Σύμβαση:** το μέγεθος του πίνακα είναι **ακριβώς** ίσο με τον αριθμό των έγκυρων εγγραφών
- **Προσθήκη εγγραφής:** το μέγεθος του πίνακα πρέπει να **αυξάνεται κατά 1**
- **Αφαίρεση/διαγραφή εγγραφής:** το μέγεθος του πίνακα πρέπει να **μειώνεται κατά 1**
  - τι γίνεται αν η εγγραφή που διαγράφεται δεν βρίσκεται στο τέλος του πίνακα;
  - μια λύση: αντιγραφή της τελευταίας εγγραφής από το τέλος του πίνακα στην θέση της εγγραφής προς διαγραφή

Πίνακας εγγραφών



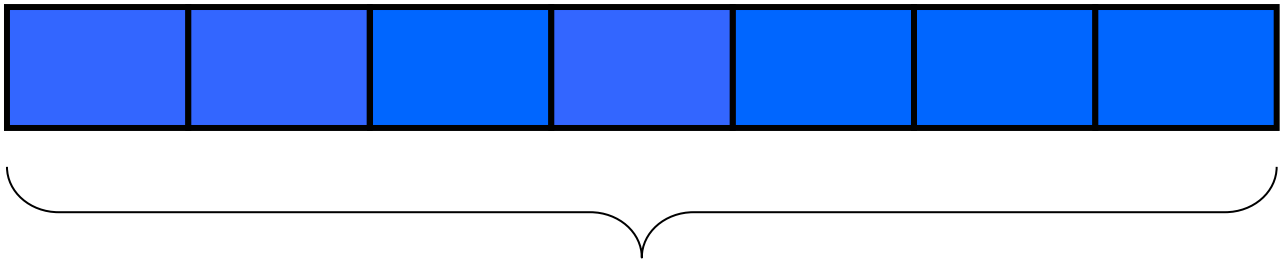
μνήμη πίνακα

Προσθήκη νέας εγγραφής



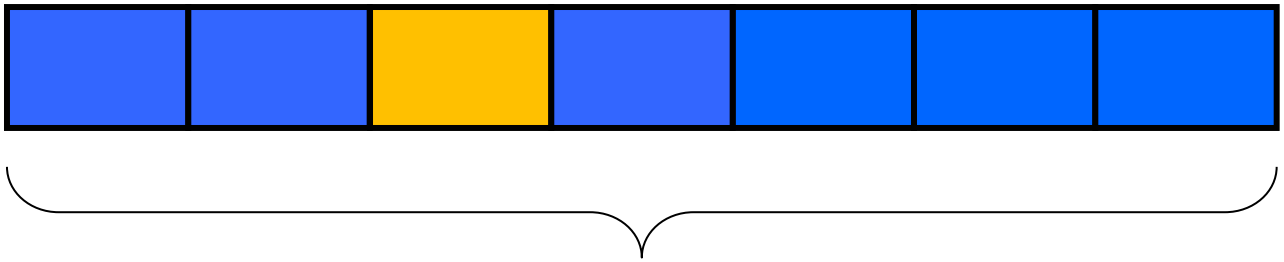
μνήμη πίνακα

Πίνακας εγγραφών



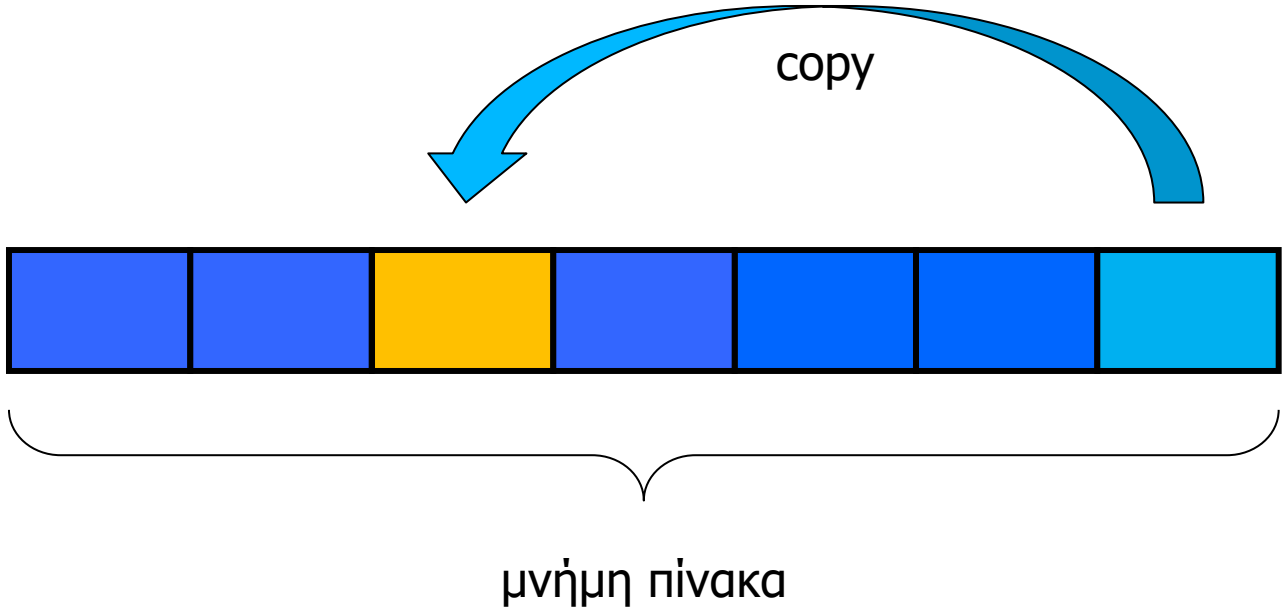
μνήμη πίνακα

Αφαίρεση υπάρχουσας εγγραφής (1)

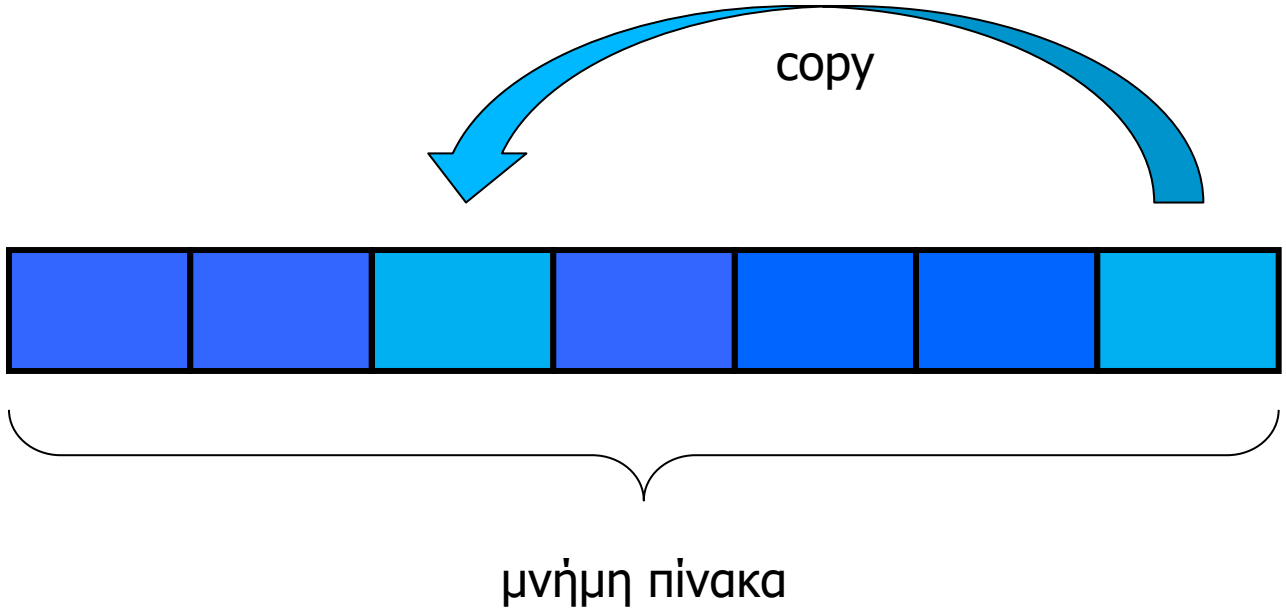


μνήμη πίνακα

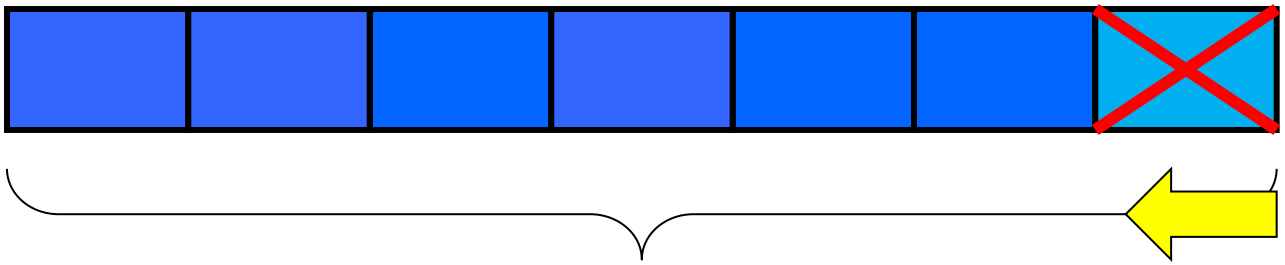
Αφαίρεση υπάρχουσας εγγραφής (2)



Αφαίρεση υπάρχουσας εγγραφής (2)

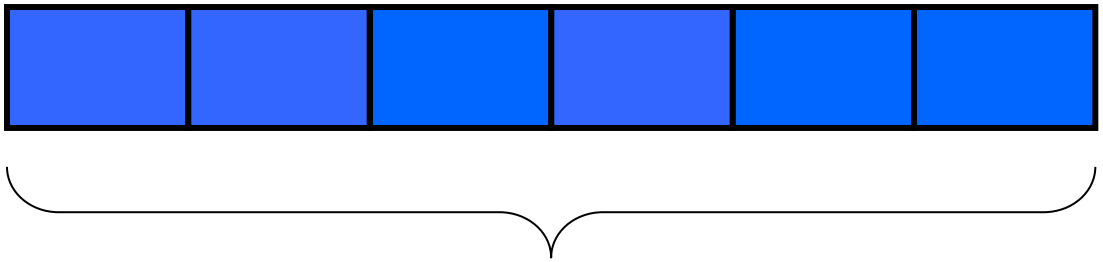


Αφαίρεση υπάρχουσας εγγραφής (3)



μνήμη πίνακα

Πίνακας εγγραφών



μνήμη πίνακα

```
typedef struct {
  char name[64];  /* name of person */
  char phone[64]; /* phone as string for flexibility */
} entry_t;

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

```
void phonebook_init(phonebook_t *pb) {
  pb->entries = NULL;
  pb->size = 0;
}
```

```
void phonebook_clear(phonebook_t *pb) {
  if (pb->size>0) {
    free(pb->entries);
    pb->entries = NULL;
    pb->size = 0;
  }
}
```

Αναζήτηση

- Η αναζήτηση είναι μια βασική λειτουργία
  - δεν εξυπηρετεί μόνο την αναζήτηση από τον χρήστη

- Αναζήτηση πρέπει να γίνει και ...
- Στην λειτουργία της **προσθήκης**
  - έλεγχος για ύπαρξη της εγγραφής (αποφυγή διπλοτύπων)
- Στην λειτουργία της **αφαίρεσης**
  - έλεγχος για ύπαρξη εγγραφής (για να αφαιρεθεί)

- **Δομημένη προσέγγιση:** υλοποιούμε την «βασική» λειτουργικότητα της αναζήτησης **μια μοναδική φορά**, σε ξεχωριστή (βοηθητική) συνάρτηση

```
/* if found, return position in table
   else return out of bounds position */

int basic_find(const phonebook_t *pb, const char *name) {
  int i;

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

  return(i);
}
```

```
int phonebook_find(const phonebook_t *pb,
                   const char *name, entry_t *e) {
  int pos;

  pos = basic_find(pb,name);

  if (pos == pb->size) {
    return(0); /* not found */
  }
  else {
    *e = pb->entries[pos];
    return(1); /* found */
  }
}
```

```
int phonebook_add(phonebook *pb_t, const entry_t *e) {
  entry_t *entries;
  int pos;

  pos = basic_find(pb,e->name);

  if (pos < pb->size) {
    pb->entries[pos] = *e;
    return(2);  /* replaced existing entry */
  }

  entries = (entry_t *)
            realloc(pb->entries,(pb->size+1)*sizeof(entry_t));

  if (entries == NULL) {
    return(0); /* out of memory */
  }

  pb->entries = entries;
  pb->entries[pb->size] = *e;
  pb->size++;

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

επέκταση πίνακα

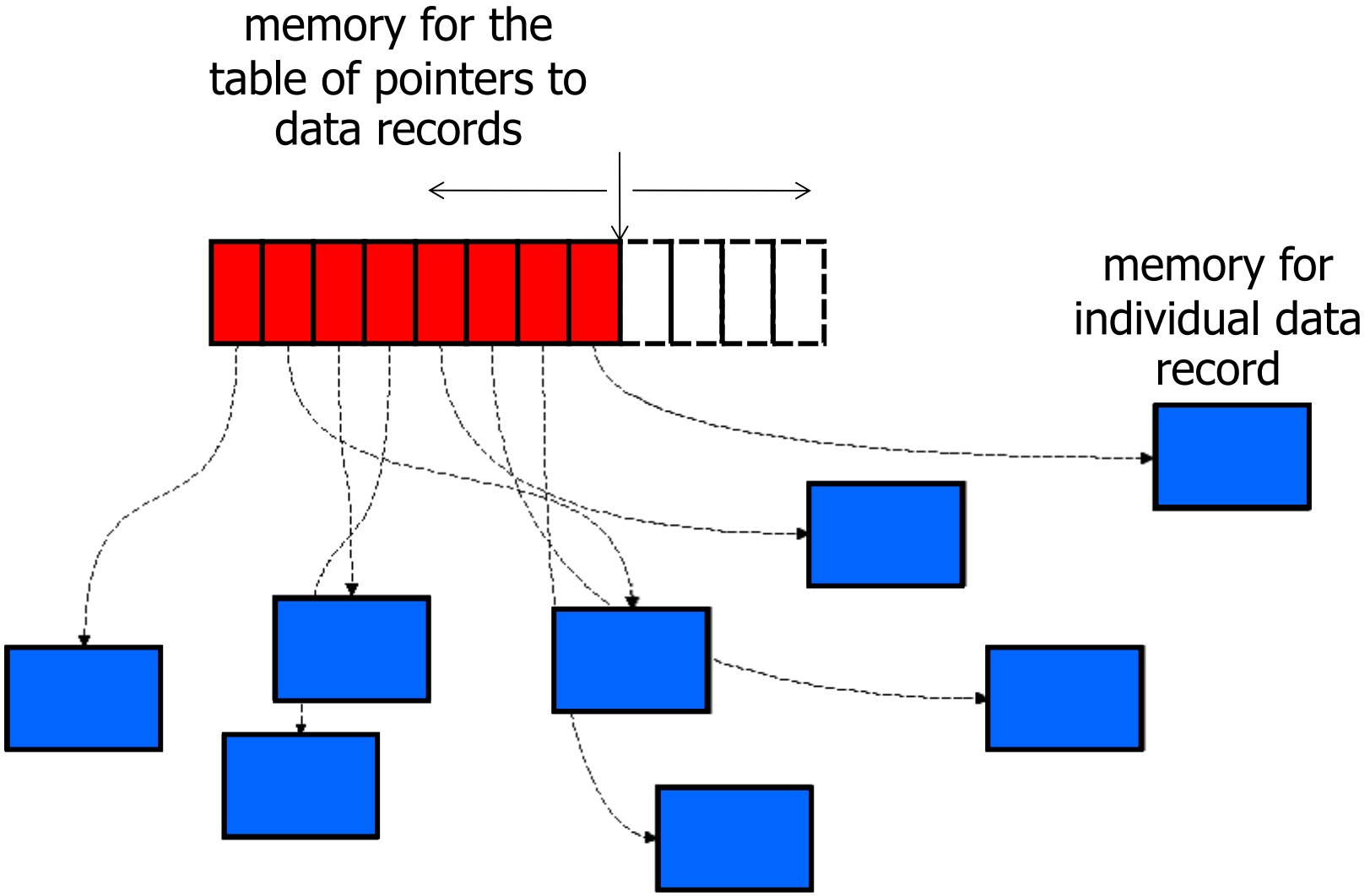
```
int phonebook_rmv(phonebook *pb_t, const char *name) {
  int pos;

  pos = basic_find(pb,name);

  if (pos == pb->size) {
    return(0); / * not found */
  }

  pb->size--;

  if (pos != pb->size) {
    pb->entries[pos] = pb->entries[pb->size];
  }

  pb->entries = (entry_t *)
                realloc(pb->entries,pb->size*sizeof(entry_t));

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

αντιγραφή τελευταίου στοιχείου στην θέση αυτού που διαγράφεται

κόψιμο πίνακα (υπόθεση ότι δεν υπάρχει περίπτωση αποτυχίας)

Παρατήρηση

- Όταν απομακρύνεται ένα στοιχείο από τον πίνακα γίνεται **αντιγραφή** δεδομένων (από την τελευταία θέση, στην θέση που ελευθερώθηκε)
  - μπορεί να είναι χρονοβόρο για μεγάλες δομές
  - η αντιγραφή γίνεται ακόμα πιο εκτενής και συχνή, αν θέλουμε τα στοιχεία να είναι ταξινομημένα – γιατί;
- Μπορούμε να αποφύγουμε αυτή την αντιγραφή;
- **Ναι**, χρησιμοποιώντας ένα **πίνακα από δείκτες** σε ξεχωριστά αντικείμενα δεδομένων (εγγραφές)
  - 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

Προσαρμογές

- Τι πρέπει να αλλάξει στους τύπους και τις συναρτήσεις της **διεπαφής προγραμματισμού**;
- Τι πρέπει να αλλάζει σε ένα **πρόγραμμα «πελάτη»** που χρησιμοποιεί αυτή την διεπαφή;
- Τι πρέπει να αλλάξει στην **υλοποίηση**;