

Προγραμματισμός I (HY120)

Διάλεξη 14:
Δυναμική Μνήμη



Δυναμική μνήμη προγράμματος



2

- Πολλές φορές, δεν γνωρίζουμε εκ των προτέρων πόση μνήμη θα χρειαστεί το πρόγραμμα μας.
 - Αν δεσμεύσουμε περισσότερη μνήμη από αυτή που θα χρειαστεί, καταναλώνουμε άσκοπα πόρους του Η/Υ
 - Διαφορετικά περιορίζουμε την λειτουργικότητα του προγράμματος μας.
- Εκτός από την στατική (καθολική) μνήμη (και την στοίβα), υπάρχει και η **δυναμική** μνήμη.
 - Ο προγραμματιστής μπορεί να **δεσμεύσει** και να **αποδεσμεύσει** δυναμική μνήμη **κατά την διάρκεια της εκτέλεσης**, ανάλογα με τις (μεταβαλλόμενες) τρέχουσες απαιτήσεις του προγράμματος.

Κύριες συναρτήσεις



3

- **`void *malloc(size_t n)`**

δέσμευση ενός συνεχόμενου τμήματος μνήμης και επιστροφή διεύθυνσης της αρχής του τμήματος (αν δεν υπάρχει αρκετή μνήμη, επιστρέφεται `NULL`)

- **`void free(void *adr)`**

αποδέσμευση της περιοχής μνήμης που αρχίζει στη διεύθυνση που δίνεται

- **`void *realloc(void *adr, size_t n)`**

αναπροσαρμογή μεγέθους του τμήματος μνήμης (με πιθανή αντιγραφή των περιεχομένων) η διεύθυνση του οποίου δίνεται σαν παράμετρος, και επιστροφή διεύθυνσης της αρχής του (νέου) τμήματος (αν δεν υπάρχει αρκετή μνήμη, επιστρέφεται `NULL`)

Σχετικές συναρτήσεις και βιβλιοθήκες



4

```
#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);
```

Χρήση δυναμικής μνήμης



5

- Οι ρουτίνες διαχείρισης δυναμικής μνήμης δεν γνωρίζουν **τίποτα** σχετικά με τους τύπους των δεδομένων που χρησιμοποιεί το πρόγραμμα.
- Η διεύθυνση που επιστρέφεται χρησιμοποιείται με αποκλειστική ευθύνη του προγραμματιστή.
- Κλασική χρήση: δέσμευση χώρου που αντιστοιχεί στο **μέγεθος** ενός αντικειμένου **τύπου T** και ανάθεση της διεύθυνσης (με type casting) σε μια μεταβλητή τύπου **δείκτη-σε-T** για ελεγχόμενη πρόσβαση στην μνήμη.
- Ο προγραμματιστής είναι υπεύθυνος για την **αποδέσμευση** δυναμικής μνήμης που δεν χρησιμοποιεί το πρόγραμμα
 - Διαφορετικά υπάρχει περίπτωση τερματισμού λόγω μη επάρκειας μνήμης.

Έλεγχος τιμής που επιστρέφεται

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



6

- Η τιμή επιστροφής πρέπει να ελέγχεται
 - Αυτό δεν γίνεται (κακώς) σε κάποια παραδείγματα των διαφανειών για οικονομία χώρου.
- Διαφορετικά, αν η τιμή `NULL` ανατεθεί σε δείκτη, η επόμενη αναφορά στη μνήμη μέσω του δείκτη θα οδηγήσει σε τερματισμό του προγράμματος.
 - **Πρόβλημα 1:** μπορεί να μην επιθυμούμε ένα τέτοιο «απότομο» τερματισμό (π.χ. χάσιμο δεδομένων).
 - **Πρόβλημα 2:** δεν γνωρίζουμε σε ποιο σημείο του προγράμματος έγινε αυτή η αναφορά.



```
#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



8

- 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** στο περιβάλλον κλήσης μιας συνάρτησης.
 - Η συνάρτηση δεσμεύει δυναμική μνήμη όπου αποθηκεύει τα δεδομένα που επιθυμεί να επιστρέψει.
 - Η συνάρτηση επιστρέφει σαν αποτέλεσμα την διεύθυνση του μπλοκ της μνήμης.
 - Το περιβάλλον κλήσης χρησιμοποιεί τη διεύθυνση όπως απαιτείται – ανάθεση σε κατάλληλη μεταβλητή δείκτη, αποδέσμευση μετά την χρήση, κλπ.



```
#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);
}
```

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

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

```
int f(int a, int b) {  
    int sum = 0;  
}
```

```
int main(int argc, char *argv[]) {  
    int a, b, *sum;  
  
    scanf("%d %d",&a, &b);  
  
    sum = add(a, b);  
  
    f(a, b);  
    printf("%d\n", *sum);  
    return(0);  
}
```



10



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

char *strAppend(const char *s1, const char *s2) {
    int i,j,len1,len2; char *s3;

    len1 = strlen(s1);
    len2 = strlen(s2);
    s3=(char *)malloc((len1+len2+1)*sizeof(char));
    strcpy(s3, s1);
    strcat(s3, s2);

    return(s3);
}

int main(int argc, char *argv[]) {
    char s1[64],s2[64],*s3;

    scanf("%63s %63s",s1,s2);
    s3=strAppend(s1,s2);
    printf("%s plus %s is %s\n",s1,s2,s3);
    free(s3);
    return(0);
}
```

Δυναμικοί πίνακες



12

- Με χρήση δυναμικής μνήμης μπορεί να υλοποιηθούν **δυναμικοί πίνακες**, το μέγεθος των οποίων ορίζεται (αλλάζει) **κατά την διάρκεια** της εκτέλεσης.
 1. Δεσμεύεται με `malloc` ή `realloc` δυναμική μνήμη μεγέθους $n * \text{sizeof}(T)$, και η διεύθυνση που επιστρέφεται αποθηκεύεται σε μεταβλητή δείκτη σε `T`.
 2. Αν το `n` αποδειχθεί μικρό ή μεγάλο, χρησιμοποιείται η **`realloc`** για την επέκταση / αποκοπή του πίνακα.
 3. Όταν ο πίνακας δεν χρειάζεται, αποδεσμεύεται η μνήμη που χρησιμοποιείται με την `free`.

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

```
char *t;
```

```
int tlen;
```

```
void init_array() {
```

```
    t=NULL;
```

```
    tlen=0;
```

```
}
```

```
void trim_array(int len) {
```

```
    t=(char*)realloc(t, len*sizeof(char));
```

```
    tlen=len;
```

```
}
```

```
void write_array(int pos, char v) {
```

```
    if (pos >= tlen)
```

```
        trim_array(pos+1);
```

```
    t[pos]=v;
```

```
}
```

```
char read_array(int pos) {
```

```
    return(t[pos]);
```

```
}
```

```
void destroy_array() { /* Do not forget to free */
```

```
    trim_array(0);
```

```
    t=NULL;
```

```
}
```



```

#include <stdlib.h>
void **t;
int tlen;

void init_array() {
    t=NULL;
    tlen=0;
}

void trim_array(int len) {
    t=(void**)realloc(t, len*sizeof(void*));
    tlen=len;
}

void write_array(int pos, void *v) {
    if (pos >= tlen)
        trim_array(pos+1);
    t[pos]=v;
}

void *read_array(int pos) {
    return(t[pos]);
}

void destroy_array() { /* Do not forget to free any */
    trim_array(0);    /* dynamically allocated */
    t=NULL;           /* elements pointed to by the */
}                    /* array before destroying it*/

```





```
int main(int argc, char *argv[]) {
    char s[256], *p; int i;

    init_array(); i = 0;

    do {
        scanf("%255s", s);
        write_array(i, (void*)strdup(s));
        i++;
    } while (strcmp(s, "end"));

    for (--i; i >= 0; i--) {
        p = (char*)read_array(i);
        printf("%s\n", p);
        free(p);
    }

    destroy_array();
    return 0;
}
```

δημιουργεί αντίγραφο
του αλφαριθμητικού (σε
δυναμική μνήμη)

type cast (από `char *`
που επιστρέφεται) σε
`void *` που αναμένει η
συνάρτηση `write`

type cast (από `void *`
που επιστρέφεται) σε
`char *` που είναι ο τύπος
της `p`

ελευθερώνει τη (δυναμική)
μνήμη που δεσμεύτηκε

Δυναμική δέσμευση δομών δεδομένων



16

- Με χρήση δυναμικής μνήμης μπορεί να δεσμευτούν χώροι μνήμης για την αποθήκευση `struct/union` κατά την διάρκεια της εκτέλεσης του προγράμματος.
 - Η δέσμευση και αποδέσμευση δυναμικής μνήμης ακολουθεί τους ίδιους κανόνες που ισχύουν και για τους βασικούς τύπους δεδομένων.
 - Κατά την δέσμευση πρέπει να δίνεται το επιθυμητό μέγεθος, μέσω `sizeof`, και η δεσμευμένη μνήμη πρέπει να αποδεσμεύεται όταν δεν είναι αναγκαία.
- Πρόσβαση στην δυναμικά δεσμευμένη μνήμη γίνεται μέσω μεταβλητών δεικτών που πρέπει να αρχικοποιούνται κατάλληλα.

Παρένθεση (βάση δεδομένων με δυναμικό πίνακα)

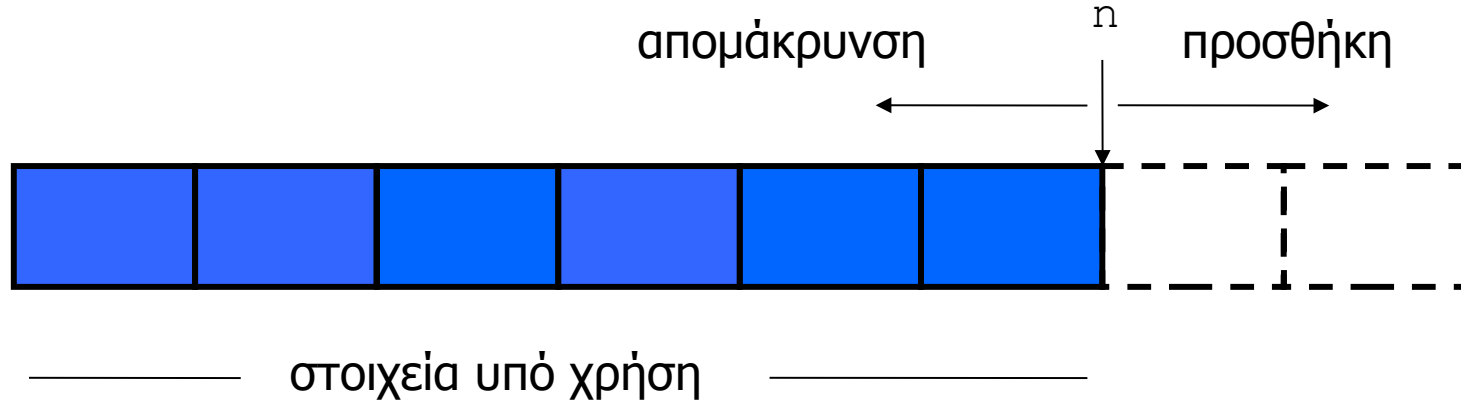




- Ζητούμενο: επιθυμούμε να διαχειριστούμε τα περιεχόμενα της τηλεφωνικής μας αντζέντας, με αντίστοιχες λειτουργίες προσθήκης, απομάκρυνσης και αναζήτησης.
- Προσέγγιση
 - ορίζουμε δομή κατάλληλη για την ομαδοποίηση των δεδομένων που ανήκουν σε μια «εισαγωγή»
 - κρατάμε τα δεδομένα σε **ανοιχτό** πίνακα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



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



```

int main(int argc, char *argv[]) {
    int sel ,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",&sel);
        switch (sel) {
            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 (sel!=4);
    return(0);
}

```





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

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

```
Entry *entries;
int size;
```

```
void phonebook_init() {
    entries=NULL;
    size=0;
}
```

η μεταβλητή entries είναι **δείκτης** σε Entry, και χρησιμεύει ως η αρχή ενός δυναμικού πίνακα από δομές Entry

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



22

```
int phonebook_find(const char name[], char phone[]) {
    int pos;
    pos=internal_find(name);
    if (pos == size)
        return(0);
    else {
        strcpy(phone,entries[pos].phone);
        return(1);
    }
}
```

```
void phonebook_rmv(const char name[]) {
    int pos;
    pos=internal_find(name);
    if (pos < size) {
        memcpy(&entries[pos],&entries[--size],sizeof(Entry));
        entries=(Entry *) realloc(entries,size*sizeof(Entry));
    }
}
```



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

    pos=internal_find(name);

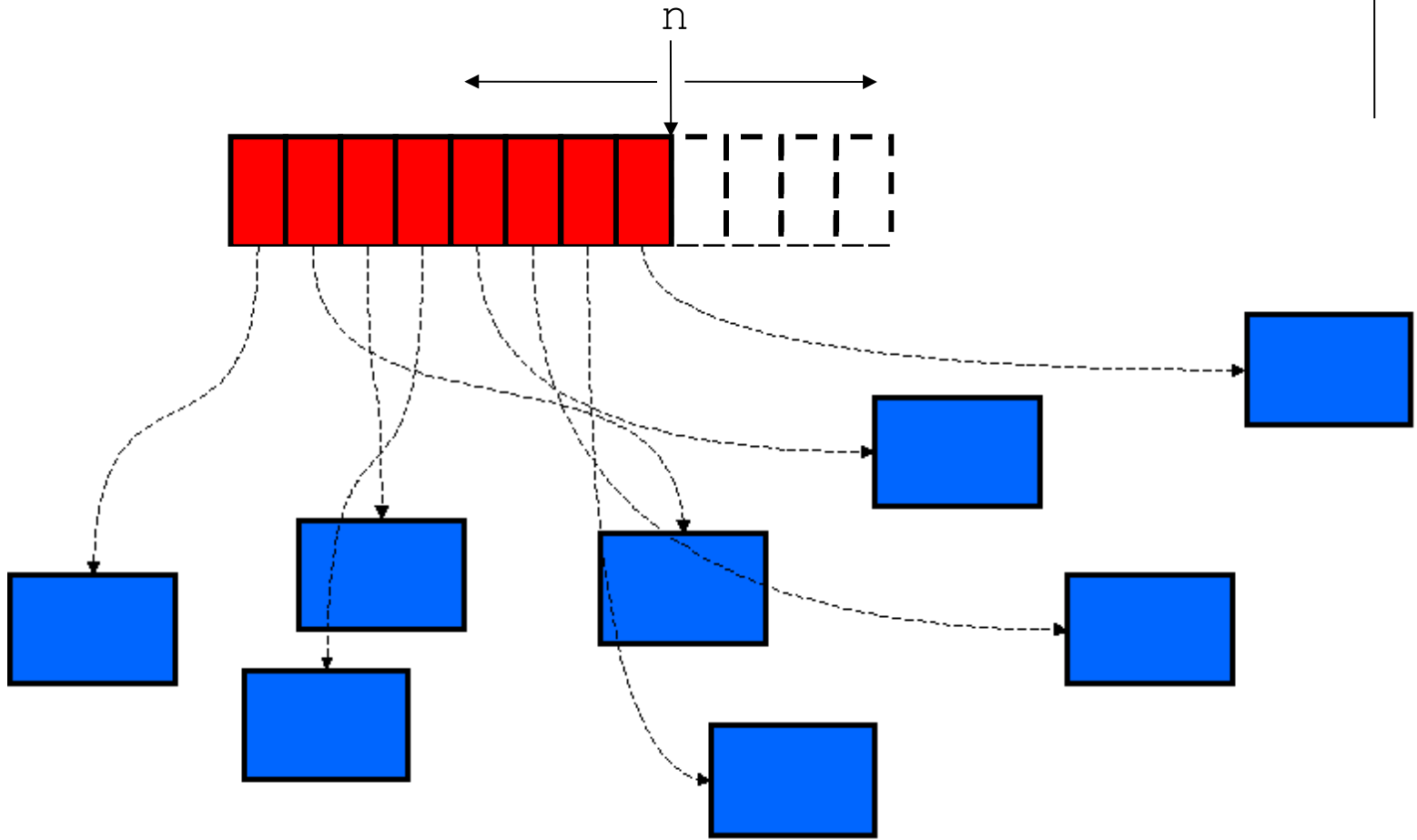
    if (pos < size) {
        strcpy(entries[pos].phone,phone);
        return(-1); /* replace */
    }

    size++;
    entries=(Entry *) realloc(entries,size*sizeof(Entry));

    strcpy(entries[size-1].name,name);
    strcpy(entries[size-1].phone,phone);
    return(1); /* done */
}
```



- Κάθε φορά που απομακρύνεται ένα στοιχείο, γίνεται μια αντιγραφή δεδομένων από την τελευταία θέση του πίνακα 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 δεδομένων, χρησιμοποιώντας ένα πίνακα **από δείκτες** σε (αυτόνομα) αντικείμενα δεδομένων η μνήμη των οποίων δεσμεύεται και αποδεσμεύεται δυναμικά κατά την προσθήκη / απομάκρυνση τους.
- Με αυτό το τρόπο ο πίνακας μετατρέπεται σε ένα ευρετήριο από δείκτες σε αντικείμενα, επιτρέποντας πολύ γρήγορες «αντιμεταθέσεις» στοιχείων.





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

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

η μεταβλητή entries είναι **δείκτης σε δείκτη** σε Entry, και χρησιμεύει ως η αρχή ενός δυναμικού πίνακα από **δείκτες** σε Entry

```
Entry **entries;
int size;
```

```
void phonebook_init() {
    entries=NULL;
    size=0;
}
```

```
int internal_find(const char name[]) {
    int i;
    for (i=0; (i<size) && (strcmp(entries[i]->name,name) != 0); i++)
        return(i);
}
```



27

```
int phonebook_find(const char name[], char phone[]) {
    int pos;
    pos=internal_find(name);
    if (pos == size)
        return(0);
    else {
        strcpy(phone,entries[pos]->phone);
        return(1);
    }
}
```

```
void phonebook_rmv(const char name[]) {
    int pos;
    pos=internal_find(name);
    if (pos < size) {
        free(entries[pos]); entries[pos] = entries[--size];
        entries=(Entry **) realloc(entries,size*sizeof(Entry *));
    }
}
```



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

    pos=internal_find(name);

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

    size++;
    entries=(Entry **) realloc(entries, size*sizeof(Entry *));

    entries[size-1] = (Entry *)malloc(sizeof(Entry));
    strcpy(entries[size-1]->name, name);
    strcpy(entries[size-1]->phone, phone);
    return(1); /* done */
}
```

Παρένθεση (βάση δεδομένων με δυναμικό πίνακα)

