

## Κεφάλαιο

# 17

### Δυναμική διαχείριση μνήμης



## Δυναμική διαχείριση μνήμης

Η μέχρι στιγμής εκμετάλλευση της μνήμης από τη C γίνεται είτε με τη χρήση μεταβλητών, είτε με τη χρήση πινάκων. Οι μεταβλητές και οι πίνακες μπορεί να είναι τριών κατηγοριών, ανάλογα με τον τρόπο και τη θέση που δηλώνονται:

- Καθολικές
- Στατικές
- Τοπικές

Οι καθολικές και οι στατικές μεταβλητές καταλαμβάνουν συγκεκριμένο χώρο στη μνήμη και διατηρούν τα περιεχόμενά τους σε όλη τη διάρκεια του προγράμματος.

Η κατανομή της μνήμης των καθολικών και των στατικών μεταβλητών λέγεται **στατική κατανομή** και οι θέσεις μνήμης δεσμεύονται κατά τη στιγμή της μεταγλώττισης.

Οι τοπικές μεταβλητές (λέγονται επίσης *αυτόματες* μεταβλητές) δεσμεύουν χώρο στη μνήμη και διατηρούν το περιεχόμενο τους όσο διαρκεί η εμφάνισή τους. Για παράδειγμα, όταν μια τοπική μεταβλητή δηλωθεί μέσα σε μια συνάρτηση, δεσμεύει χώρο μνήμης ανάλογο με τον τύπο της. Όταν η συνάρτηση επιστρέφει, ο χώρος που δέσμευε η μεταβλητή απελευθερώνεται και τα περιεχόμενά της χάνονται.


Η κατανομή της μνήμης των τοπικών μεταβλητών λέγεται **αυτόματη κατανομή** και οι θέσεις μνήμης δεσμεύονται προσωρινά (κατά τη δήλωσή τους) και αποδεσμεύονται μόλις λήξει η εμφάνισή τους.

Έτσι, οι προτάσεις

```
char a,b;  
int num[100];
```

δεσμεύουν δύο byte, για τις μεταβλητές **a** και **b**, και 400 byte (4 x 100) για τον πίνακα **num**. Οι θέσεις αυτές δεσμεύονται τη στιγμή της δήλωσης και αποδεσμεύονται μόλις λήξει η εμφάνιση των μεταβλητών.

Αν όμως θέλουμε π.χ. έναν πίνακα στον οποίο να καταχωρίζουμε τους αριθμούς των αυτοκινήτων που περνάνε μπροστά από ένα σημείο παρατήρησης, το ερώτημα είναι πόσο μεγάλος πρέπει να είναι αυτός ο πίνακας. Αν τον ορίσουμε μικρό (π.χ. με 100 θέσεις) είναι πιθανό να μην επαρκεί επειδή μπορεί να περάσουν περισσότερα αυτοκίνητα. Αν τον ορίσουμε αρκετά μεγάλο, προβλέποντας (αν αυτό είναι δυνατόν) τη μέγιστη κίνηση που θα μπορούσαμε να έχουμε (π.χ. με 1000 θέσεις) το πιθανότερο είναι να σπαταλήσουμε άσκοπα μνήμη που δεν θα χρησιμοποιήσουμε ποτέ.

 Πρέπει να έχουμε υπόψη ότι δεν είναι δυνατόν να μεταβάλλουμε το μέγεθος ενός πίνακα. Ο πίνακας έχει πάντα το μέγεθος που ορίζουμε με την πρόταση δήλωσης του.

Ένα άλλο πρόβλημα προκύπτει όταν δεν γνωρίζουμε το μέγεθος του πίνακα όταν γράφουμε το πρόγραμμα, αλλά το γνωρίζουμε κατά την ώρα εκτέλεσης του προγράμματος. Για παράδειγμα, αν το πρόγραμμα ζητάει τον αριθμό των μαθητών μιας τάξης και μετά θέλουμε να φτιάξουμε έναν πίνακα με τόσες θέσεις όσες ο αριθμός των μαθητών.

Στη C δεν είναι δυνατόν να δηλώσουμε ένα πίνακα που οι διαστάσεις του να καθορίζονται από μεταβλητές. Πρόταση όπως η παρακάτω:

```
int num[a]
```

δεν είναι αποδεκτή.

**Τι γίνεται λοιπόν σε αυτές τις περιπτώσεις;**

Η λύση βρίσκεται στο μηχανισμό δυναμικού καταμερισμού μνήμης που διαθέτει η C.

Η C μας εφοδιάζει με μηχανισμό και συναρτήσεις **δυναμικής κατανομής** της μνήμης. Μπορούμε να δεσμεύουμε περισσότερη μνήμη ή να απελευθερώνουμε μνήμη που δεν χρειαζόμαστε.



## Δυναμική κατανομή μνήμης


Στο διπλανό σχήμα απεικονίζεται ο τρόπος με τον οποίο διαχειρίζεται η C τη μνήμη. Ένα τμήμα μνήμης χρησιμοποιείται για την αποθήκευση του κώδικα του προγράμματος και ένα γειτονικό τμήμα καταλαμβάνουν οι καθολικές και στατικές μεταβλητές.

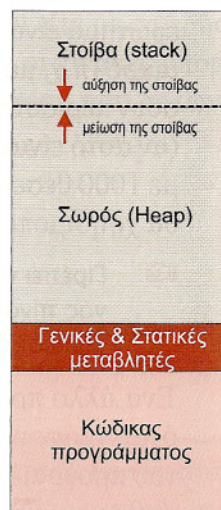
Η **στοίβα** (stack) είναι ένας χώρος μνήμης στον οποίο αποθηκεύονται οι τοπικές μεταβλητές και ο οποίος αυξάνεται ή μειώνεται ανάλογα με τις ανάγκες του προγράμματος.

Ο υπόλοιπος αδιάθετος χώρος μνήμης μεταξύ της στοίβας και του χώρου των γενικών μεταβλητών λέγεται **σωρός** (heap).

Στο δυναμικό καταμερισμό μνήμης μπορούν να διατεθούν κατά την ώρα εκτέλεσης (run-time) του προγράμματος τμήματα μνήμης (memory blocks) από το σωρό, να χρησιμοποιηθούν, και να απελευθερωθούν όταν δεν χρειάζονται.

Οι συναρτήσεις που χρησιμοποιούνται για τον δυναμικό καταμερισμό μνήμης είναι (σύμφωνα με το πρότυπο ANSI) οι `malloc()`, `calloc()`, `free()`, και `realloc()`.

 Ο μηχανισμός και οι συναρτήσεις που χρησιμοποιούνται στο δυναμικό καταμερισμό μνήμης κάνουν εκτενή χρήση των δεικτών.



### Η συνάρτηση `malloc()`

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

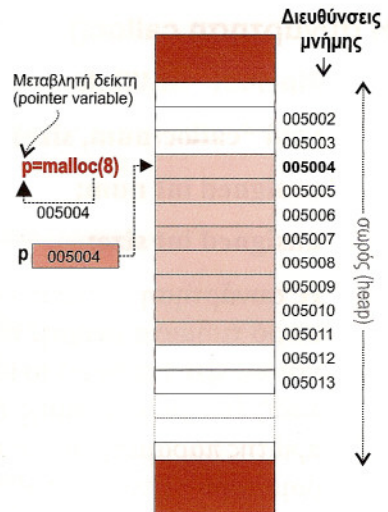
```
void *malloc(size)
```

```
unsigned int size;
```

Η συνάρτηση `malloc()` δεσμεύει από το σωρό ένα τμήμα (block) μνήμης, μεγέθους (σε byte) όσο η τιμή της παραμέτρου `size`. Η `malloc()` επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του block που δεσμευσε.

👉 Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το μέγεθος του τμήματος, η `malloc()` επιστρέφει δείκτη `NULL`. Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.

Η `malloc()` έχει μία παράμετρο τύπου `unsigned int` και η τιμή που επιστρέφει είναι ένας δείκτης (pointer) τύπου `void`.



Η συνήθης χρήση της `malloc()` φαίνεται στον επόμενο κώδικα:

```
char *p;
p=(char *)malloc(8);
if (p==NULL)
{
    puts("Δεν υπάρχει αρκετή μνήμη");
    exit(2);
}
gets(p);
putch(p[3]);
```

Μετατροπή του δείκτη `void` που επιστρέφει η `malloc()` σε δείκτη προς δεδομένα τύπου `char`.

Έλεγχος του δείκτη που επέστρεψε η `malloc()`.

Καταχώριση των χαρακτήρων που πληκτρολογούνται, στο χώρο μνήμης που δείχνει ο δείκτης `p`.

Εμφάνιση του 4ου χαρακτήρα από αυτούς που πληκτρολογήθηκαν.

👉 Αν θυμηθούμε λίγο την φιλοσοφία των δεικτών και τη σχέση τους με τους πίνακες, οι παραστάσεις `p[3]` και `*(p+3)` είναι ισοδύναμες και αναφέρονται στο τέταρτο από τα δεδομένα που ξεκινάνε από τη θέση όπου δείχνει ο δείκτης `p`.

👉 Επομένως, μπορούμε να χρησιμοποιούμε το χώρο μνήμης που δεσμεύτηκε σαν ένα μονοδιάστατο πίνακα με όνομα το όνομα του δείκτη στον οποίο έχει καταχωριστεί η διεύθυνση που επέστρεψε η `malloc()`. Απαραίτητη είναι η μετατροπή του δείκτη που επιστρέφει η `malloc()` σε δείκτη του τύπου δεδομένων που πρόκειται να καταχωριστούν μέσα στο χώρο μνήμης που δεσμεύτηκε.

## Η συνάρτηση calloc()

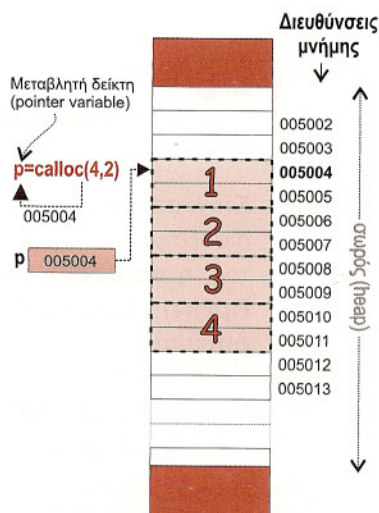
```
#include <stdlib.h>
```

```
void *calloc(num, size)
```

```
unsigned int num;
```

```
unsigned int size;
```

Η συνάρτηση `calloc()` δεσμεύει από τον σωρό τμήματα μνήμης πλήθους **num** και μεγέθους **size** (σε byte) το καθένα. Δεσμεύει δηλαδή ένα χώρο μνήμης όσο το γινόμενο της πρώτης παραμέτρου **num** και της δεύτερης παραμέτρου **size**: **num\*size** byte. Η `calloc()` επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του χώρου μνήμης που δέσμευσε.



- ☞ Με άλλα λόγια, η `calloc()` δεσμεύει την απαραίτητη μνήμη για ένα σύνολο (π.χ. ένα πίνακα) από αντικείμενα πλήθους **num** και μεγέθους **size**.
- ☞ Η μόνη διαφορά μεταξύ `malloc()` και `calloc()` είναι στον τρόπο υπολογισμού του μεγέθους του τμήματος της μνήμης που θα δεσμεύσουν. Ενώ στη `malloc()` το μέγεθος δίνεται έτοιμο σε byte, η `calloc()` το υπολογίζει πολλαπλασιάζοντας τις δύο παραμέτρους της (**num\*size**).
- ☞ Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το μέγεθος του τμήματος, η `calloc()` επιστρέφει δείκτη NULL. Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.

Η `calloc()` έχει δύο παραμέτρους τύπου **unsigned int** και η τιμή που επιστρέφει είναι ένας δείκτης τύπου **void**.



## Η συνάρτηση free()

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

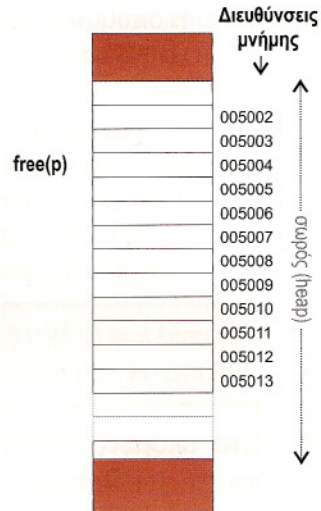
```
void free(ptr)
```

```
void *ptr;
```

Η συνάρτηση **free()** αποδεσμεύει από το σωρό το τμήμα μνήμης στο οποίο δείχνει ο δείκτης **ptr**.

👉 Η **free()** πρέπει να κληθεί με παράμετρο ένα δείκτη ο οποίος να έχει επιστραφεί από προηγούμενες κλήσεις συναρτήσεων δυναμικής κατανομής όπως η **malloc()** και η **calloc()**.

Η χρήση της **free()** με έναν αυθαίρετο δείκτη θα έχει καταστροφικά αποτελέσματα.



Η **free()** έχει μία παράμετρο που είναι δείκτης τύπου **void** και δεν επιστρέφει καμία τιμή.

## Η συνάρτηση realloc()

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

```
void *realloc(ptr, size)
```

```
void *ptr;
```

```
unsigned int size;
```

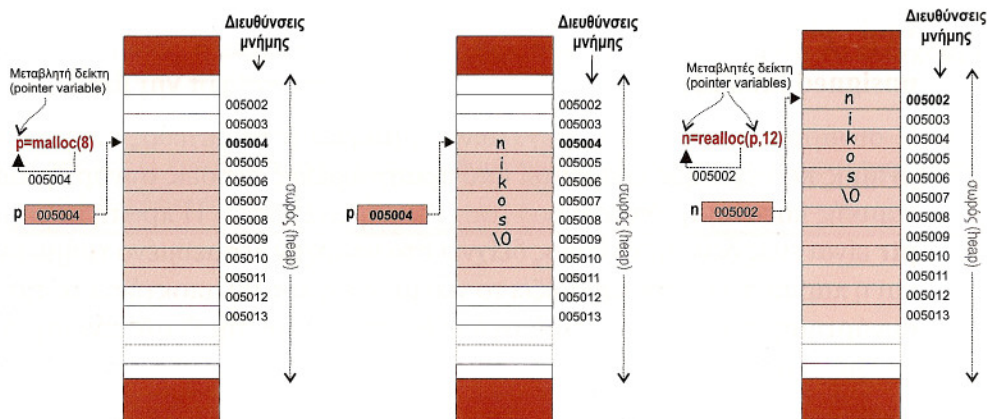
Η συνάρτηση **realloc()** μεγαλώνει ή μικραίνει το μέγεθος ενός τμήματος μνήμης που έχει ήδη δεσμευτεί από προηγούμενες κλήσεις συναρτήσεων δυναμικής κατανομής όπως η **malloc()** και η **calloc()**. Η πρώτη παράμετρος **ptr** είναι ένας δείκτης ο οποίος δείχνει στο υπάρχον δεσμευμένο τμήμα μνήμης και η παράμετρος **size** καθορίζει το νέο μέγεθος που θα αποκτήσει το τμήμα. Η συνάρτηση επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του νέου τμήματος.

- 👉 Η διεύθυνση του νέου τμήματος (δηλαδή ο δείκτης που επιστρέφεται) μπορεί να είναι διαφορετική από αυτήν του αρχικού τμήματος. Για να μπορέσει η C να δώσει στο τμήμα το νέο του μέγεθος, μπορεί να χρειαστεί να το μετακινήσει σε άλλη θέση, ακόμη και στην περίπτωση που είναι μικρότερο. Τα περιεχόμενα του αρχικού τμήματος αντιγράφονται στη νέα θέση.
- 👉 Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το νέο μέγεθος του block η **realloc()** επιστρέφει δείκτη NULL χωρίς να επηρεάσει τα περιεχόμενα του αρχικού block. **Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.**

Η **realloc()** έχει δύο παραμέτρους τύπου δείκτη **void** και **unsigned int** αντίστοιχα. Η τιμή που επιστρέφει η συνάρτηση είναι ένας δείκτης τύπου **void**.

Στο επόμενο παράδειγμα φαίνεται η λειτουργία της **realloc()**. Με τη **malloc()** δεσμεύουμε ένα τμήμα 8 θέσεων μνήμης, την αρχική διεύθυνση του οποίου (υποθέτουμε την 005002) καταχωρίζουμε στο δείκτη **p**. Με την **strcpy()** αντιγράφουμε το σύνολο χαρακτήρων "nikos" μέσα στο δεσμευμένο τμήμα μνήμης. Η κλήση της **realloc()** που ακολουθεί έχει αποτέλεσμα την αύξηση του δεσμευμένου τμήματος από 8 σε 12 θέσεις μνήμης.

```
char *p, *n;
p=(char *)malloc(8);
strcpy(p, "nikos");
n=realloc(p, 12);
```





- ☞ Παρατηρούμε τη φράση `(char *)malloc(8)`. Εδώ χρησιμοποιείται η τεχνική της μετατροπής τύπου (type casting)<sup>10</sup>, για να μετατραπεί ο δείκτης τύπου `void` που επιστρέφεται από τη `malloc()` σε δείκτη προς δεδομένα τύπου `char`.
- ☞ Παρατηρούμε ότι υπάρχει περίπτωση το νέο τμήμα να αρχίζει από διαφορετικό σημείο του σωρού. Σε αυτή την περίπτωση, η C αντιγράφει τα περιεχόμενα του αρχικού τμήματος στη νέα του θέση.

## Παραδείγματα

- Π.1** Το επόμενο πρόγραμμα δεσμεύει 256 byte μνήμης. Μετά ζητάει να πληκτρολογηθεί ένα σύνολο χαρακτήρων, οι οποίοι καταχωρίζονται στις θέσεις που δεσμεύτηκαν. Τέλος, εμφανίζει στην οθόνη το σύνολο χαρακτήρων και μετά απελευθερώνει τις θέσεις που δέσμευσε.

```
main()
{
    char *k;
    k = (char *)malloc(256);
    if (k == NULL)
    {
        puts("Δεν υπάρχει αρκετή μνήμη για δέσμευση");
        exit(2);
    }
    gets(k);
    puts(k);
    free(k);
}
```

Μετατροπή του δείκτη `void` που επιστρέφει η `malloc()` σε δείκτη προς δεδομένα τύπου `char`.

Απελευθέρωση των 256 δεσμευμένων θέσεων που δείχνει ο δείκτης `k`.

- Π.2** Το επόμενο πρόγραμμα δεσμεύει τον απαραίτητο χώρο για την καταχώριση 100 δεκαδικών αριθμών. Μετά ζητάει να πληκτρολογήσουμε 10 αριθμούς, τους οποίους καταχωρίζει στις πρώτες 10 από τις 100 θέσεις.

<sup>10</sup> Βλέπε Κεφάλαιο 8 - § Μετατροπή τύπου

```
main()
{
    float *k;
    int i;
    k=(float *)calloc(100,sizeof(float));
    if(k==NULL)
    {
        puts("Δεν υπάρχει αρκετή μνήμη για δέσμευση");
        exit(2);
    }
    for(i=0;i<10;i++)
    {
        scanf("%f",&k[i]);
    }
}
```

Χρησιμοποιούμε το χώρο μνήμης που δεσμεύτηκε σαν ένα μονοδιάστατο πίνακα float k[100].

## Ανασκόπηση Κεφαλαίου 17

- Η εκχώρηση μνήμης μέσω των μεταβλητών και των πινάκων δεν επιτρέπει την αλλαγή του μεγέθους της δεσμευμένης μνήμης κατά την ώρα εκτέλεσης του προγράμματος.
- Ο μηχανισμός δυναμικής κατανομής μνήμης της C επιτρέπει στον προγραμματιστή να δεσμεύει και να αποδεσμεύει τμήματα μνήμης κατά την ώρα της εκτέλεσης του προγράμματος.
- Όλες οι συναρτήσεις δυναμικής κατανομής μνήμης χρησιμοποιούν δείκτες προς τα τμήματα μνήμης που χειρίζονται.

## Ασκήσεις Κεφαλαίου 17

- 17.1** Να γραφεί πρόγραμμα το οποίο να διαβάζει το πλήθος των αριθμών που πρόκειται να καταχωριστούν σε έναν πίνακα, να δημιουργεί έναν πίνακα με θέσεις μνήμης τόσες ώστε να χωράνε **ακριβώς** το πλήθος των δεδομέ-

νων και μετά να διαβάσει και να καταχωρίζει αυτούς τους αριθμούς μέσα στον πίνακα. ★ ★

- 17.2 Να συμπληρωθεί το πρόγραμμα της προηγούμενης άσκησης ώστε μετά από την καταχώριση των αριθμών να αυξάνει το μέγεθος του πίνακα στο διπλάσιο του αρχικού του μεγέθους. ★

- 17.3 Να γραφεί πρόγραμμα το οποίο να διαβάσει το πλήθος των μαθητών ενός σχολείου και να δεσμεύει τόσο χώρο μνήμης όσο χρειάζεται για την καταχώριση των στοιχείων των μαθητών όπως φαίνεται στη δομή `stoxeia`. Αμέσως μετά να ζητάει και να καταχωρεί τα στοιχεία για το 10ο μαθητή. ★ ★ ★

```
struct stoxeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
};
```

- 17.4 Να συμπληρωθεί το πρόγραμμα του παραδείγματος Π2 ώστε μετά από την καταχώριση των 10 αριθμών να απελευθερώνει το μισό από το δεσμευμένο χώρο μνήμης (τις 50 από τις 100 θέσεις) χωρίς όμως να χαθούν τα δεδομένα που έχουμε καταχωρίσει. ★

- 17.5 Ποια από τα επόμενα αληθεύουν: ★

- ☐ Ο πίνακας αποτελεί μια δυναμική κατανομή μνήμης.
- ☐ Στη δυναμική κατανομή μνήμης υπάρχει περίπτωση να μην είναι δυνατή η δέσμευση της ποσότητας μνήμης που ζητάμε.
- ☐ Η `malloc(5,100)` δεσμεύει 500 byte μνήμης.
- ☐ Η `free(100)` αποδεσμεύει 100 byte από τη μνήμη.
- ☐ Η `calloc()` και η `malloc()` επιτελούν σχεδόν την ίδια λειτουργία.