Προγραμματισμός II

Δυναμική Διαχείριση Μνήμης

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Ανάγκη για Δυναμική Μνήμη

Στατική Μνήμη

Μέχρι τώρα χρησιμοποιούσαμε στατική μνήμη, δηλαδή τα προγράμματα μας ήξεραν από πριν (την ώρα την μεταγλώττισης) πόση μνήμη χρειάζεται.

Τι γίνεται όμως όταν δεν ξέρουμε πόση ακριβώς μνήμη χρειαζόμαστε;

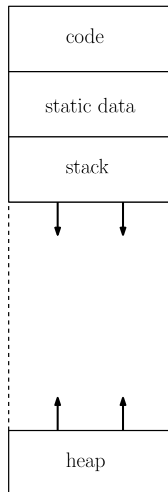
π.χ

Θέλουμε να γράψουμε ένα πρόγραμμα που να διαβάζει ένα σύνολο ακεραίων από ένα αρχείο και να τους ταξινομεί σε μη-φθίνουσα σειρά. Πόσους όμως ακέραιους;

Stack και Heap

Η μνήμη που καταλαμβάνει ένα πρόγραμμα διαχωρίζεται σε διάφορα μέρη.

- Ένα κομμάτι όπου φορτώνεται ο κώδικας του προγράμματος μας
- Ένα κομμάτι όπου βρίσκονται οι στατικές μεταβλητές
- Την στοίβα (stack) όπου αποθηκεύονται όλες οι τοπικές μεταβλητές, οι παράμετροι των συναρτήσεων, προσωρινές μεταβλητές, κ.τ.λ.
- Τον σωρό (heap) όπου είναι μνήμη που μπορούμε να την δεσμεύσουμε ή να την ελευθερώσουμε κατά την διάρκεια του προγράμματος.



Την διαχείριση μνήμης την αναλαμβάνει το λειτουργικό σύστημα.

- Η C μας παρέχει στην βιβλιοθήκη της, διάφορες συναρτήσεις που είναι υπεύθυνες ώστε να μας παραχωρήσει το λειτουργικό σύστημα μνήμη.

Η μέγιστη μνήμη που το πρόγραμμα μας μπορεί να ζητήσει έχει να κάνει με:

- την συνολική μνήμη του συστήματος,
- πόση μνήμη είναι διαθέσιμη για προγράμματα χρηστών,
- πόσα άλλα προγράμματα τρέχουν ταυτόχρονα και πόση μνήμη έχουν δεσμεύσει.

Ο Τελεστής `sizeof()`

Για να ζητήσουμε μνήμη από το σύστημα πρέπει να ξέρουμε πόση μνήμη χρειαζόμαστε.

Η C μας παρέχει τον τελεστή `sizeof()` ώστε να μπορούμε να μάθουμε πόση μνήμη σε bytes καταναλώνουν οι διάφοροι τύποι στην C.

Ο Τελεστής sizeof()

Παραδείγματα

```c
#include <stdio.h>

int main()
{
    printf("size of char in bytes = %lu\n", sizeof(char));
    printf("size of int in bytes = %lu\n", sizeof(int));
    printf("size of long in bytes = %lu\n", sizeof(long));
    printf("size of double in bytes = %lu\n", sizeof(double));
    printf("size of void* in bytes = %lu\n", sizeof(void*));

    return 0;
}
```

με έξοδο (στο σύστημα του ομιλητή)

```
size of char in bytes = 1
size of int in bytes = 4
size of long in bytes = 8
size of double in bytes = 8
size of void* in bytes = 8
```

Ο Τελεστής sizeof()

Παραδείγματα

```c
#include <stdio.h>

typedef struct {
    long AM;
    char name[56];
} student;

int main()
{
    printf("size of student in bytes = %lu\n", sizeof(student));
    return 0;
}
```

με έξοδο (στο σύστημα του ομιλητή)

```
size of student in bytes = 64
```

Ο Τελεστής `sizeof()`
Παραδείγματα

Προσοχή με το μέγεθος ενός `struct` γιατί πολλές φορές ο μεταγλωττιστής προσθέτει επιπλέον bytes στο τέλος ώστε να βελτιστοποιήσει την αντιγραφή (οι λεπτομέρειες εξαρτώνται από την CPU, το λειτουργικό και τον μεταγλωττιστή).

```c
#include <stdio.h>

typedef struct {
    long AM;
    char name[50];
} student;

int main()
{
    printf("size of student in bytes = %lu\n", sizeof(student));

    return 0;
}
```

το παραπάνω πρόγραμμα ενώ θα έπρεπε να τυπώνει 58, στο σύστημα του ομιλητή τυπώνει

```
size of student in bytes = 64
```

Ο Τελεστής sizeof()

Ως τώρα δώσαμε στον τελεστή sizeof() τύπους δεδομένων. Μπορούμε όμως να δώσουμε και μεταβλητές.

```c
#include <stdio.h>

int main()
{
    double x;
    double a[30];

    printf("size of x in bytes = %lu\n", sizeof(x));
    printf("size of a in bytes = %lu\n", sizeof(a));

    return 0;
}
```

```
size of x in bytes = 8
size of a in bytes = 240
```

Ο Τελεστής sizeof()

Προσοχή με τους δείκτες και τους πίνακες.

```
1   #include <stdio.h>
2
3   int main()
4   {
5       double a[30];
6       double *p = a;
7
8       printf("size␣of␣a␣in␣bytes␣=␣%lu\n", sizeof(a));
9       printf("size␣of␣p␣in␣bytes␣=␣%lu\n", sizeof(p));
10
11      return 0;
12  }
```

```
size of a in bytes = 240
size of p in bytes = 8
```

Ο Τελεστής sizeof()

Προσοχή με τους δείκτες και τους πίνακες.

```c
#include <stdio.h>

void printarray(double a[30]) {
    // ...

    printf("size of a in bytes = %lu\n", sizeof(a));
}

int main()
{
    double a[30];

    printf("size of a in bytes = %lu\n", sizeof(a));
    printarray(a);

    return 0;
}
```

```
size of a in bytes = 240
size of a in bytes = 8
```

Θυμηθείτε πως σε μία συνάρτηση περνάμε μόνο την διεύθυνση ενός πίνακα ως παράμετρο.

Προτεραιότητα Τελεστών

1. **παρενθέσεις:** () [] . −> `expr`++ `expr`−−
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι τελεστές:** + − ++`expr` −−`expr` ! * & `sizeof`
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** * / %
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** + −
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** < > <= >=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** == !=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **λογικό AND:** && Από αριστερά προς τα δεξιά.
8. **λογικό OR:** || Από αριστερά προς τα δεξιά.
9. **εκχώρησης:** = += −= *= /= %=
   Από δεξιά προς τα αριστερά.

Η συνάρτηση `malloc()`

Η συνάρτηση `malloc` είναι ορισμένη στο αρχείο `stdlib.h` και είναι υπεύθυνη για την δημιουργία μνήμης στον σωρό (heap).

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

- πέρνει ως είσοδο το μέγεθος μνήμης που θέλουμε να δεσμεύσουμε σε bytes,
- σε περίπτωση επιτυχίας επιστρέφει έναν δείκτη στη διεύθυνση μνήμης που δεσμεύτηκε,
- εαν η μνήμη δεν δεσμεύτηκε (π.χ δεν υπάρχει άλλη ελεύθερη μνήμη στο σύστημα) επιστρέφει `NULL`.

Δείκτης σε Τύπο void

Μετατροπή (casting)

Η συνάρτηση malloc επιστρέφει ένα δείκτη σε τύπο void.

Ο προγραμματιστής πρέπει στην συνέχεια να μετατρέψει στον τύπο που χρειάζεται (casting).

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

int main()
{
    int *x = (int*) malloc(sizeof(int));

    // ...

    return 0;
}
```

Ο παραπάνω κώδικας φτιάχνει έναν ακέραιο.

Type Casting

Η μετατροπή μιας έκφρασης ενός τύπου σε έναν άλλο.

1. έμμεση μετατροπή

   Γίνονται αυτόματα όταν μια τιμή αντιγράφεται σε έναν συμβατό τύπο.

```
short a = 2000;
int b = a;
```

   Επιτρέπεται μόνο αν δεν χάνεται πληροφορία.

2. άμεση μετατροπή

   Υποδείχνουμε εμείς την μετατροπή.

```
float a = 5.25;
int b = (int) a;
```

Προσοχή με την χρήση των typecasts, πολλές φορές οδηγούν σε πολύ δυσεύρετα λάθη.

Προτεραιότητα Τελεστών

1. **παρενθέσεις:** () [] . −> `expr++` `expr−−`
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι τελεστές:** + − `++expr` `−−expr` ! * & `sizeof` (`typecast`)
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** * / %
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** + −
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** < > <= >=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** == !=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **λογικό AND:** && Από αριστερά προς τα δεξιά.
8. **λογικό OR:** || Από αριστερά προς τα δεξιά.
9. **εκχώρησης:** = += −= *= /= %=
   Από δεξιά προς τα αριστερά.

Η συνάρτηση `free()`

Ο προγραμματιστής είναι υπεύθυνος να ελευθερώσει την μνήμη που δέσμευσε με την χρήση της συνάρτησης:

```
void free(void* ptr);
```

Η συνάρτηση ελευθερώνει την μνήμη που

- δείχνει ο δείκτης `ptr` και
- έχει δεσμευτεί με την `malloc()`.

- Για κάθε κλήση της `malloc()` πρέπει να υπάρχει και μία κλήση της `free()`.

Παράδειγμα Χρήσης της malloc()

Δημιουργία struct

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

typedef struct {
    int AM;
    char name[50];
} student;

int main()
{
    student *x = (student*) malloc(sizeof(student));

    // use x

    free(x);
    return 0;
}
```

Δημιουργία Πίνακα

Η δημιουργία δυναμικού πίνακα γίνεται ζητώντας από την `malloc()` ένα πολλαπλάσιο σε bytes του τύπου που θέλουμε.

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

int main()
{
    int n = 100, i, *array;

    array = (int*) malloc(n * sizeof(int));

    for(i = 0; i < n; i++)
        array[i] = i*i;
    for(i = 0; i < n; i++)
        printf("%d ", array[i]);

    free(array);

    return 0;
}
```

Προσοχή η μνήμη που επιστρέφει η `malloc()` δεν έχει αρχικοποιηθεί.

Παράδειγμα Χρήσης της malloc()

Δημιουργία Πίνακα

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

typedef struct {
    int AM;
    char name[50];
} student;

int main()
{
    int n = 100, i;
    student *array = (student*) malloc(n * sizeof(student));

    // use array

    free(array);
    return 0;
}
```

Διαρροή Μνήμης

Memory Leaks

Διαρροή μνήμης έχουμε εαν ξεχάσουμε να ελευθερώσουμε την μνήμη που δεσμεύσαμε.

- Η μνήμη αυτή παραμένει άχρηστη μέχρι το λειτουργικό σύστημα να την ελευθερώσει.
- Είναι συνήθης λόγος προβλημάτων προγραμματισμού.

Ένα πρόγραμμα με διαρροή μνήμης δημιουργεί προβλήματα στο σύστημα και έχει μεγάλες πιθανότητες να τερματιστεί πρόωρα.

Για να φτιάξουμε δυναμικά έναν πίνακα με δείκτες χρειαζόμαστε ένα δείκτη που να δείχνει σε τύπο δείκτη.

Όταν γράφουμε

```
int *p;
```

δηλώνουμε ένα δείκτη με όνομα `p` που δείχνει σε τύπο ακέραιο.

Αντίστοιχα γράφουμε

```
int **p;
```

για δηλώνουμε ένα δείκτη με όνομα `p` που να δείχνει σε τύπο δείκτη σε ακέραιο. Θα μπορούσαμε να το γράφαμε ως `int* *p` για να είναι πιο σαφές.

Για να φτιάξουμε δυναμικά έναν πίνακα με δείκτες χρειαζόμαστε ένα δείκτη που να δείχνει σε τύπο δείκτη.

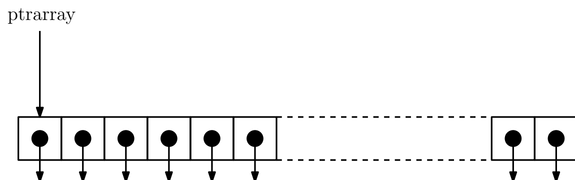
```
int **ptrarray = (int**) malloc(100 * sizeof(int*));
```

Ζητάμε από την `malloc()` να μας επιστρέψει ένα πίνακα με μέγεθος 100 επί το μέγεθος που καταναλώνει ένας δείκτης σε ακέραιο.

Κάθε στοιχείο π.χ `ptrarray[4]` είναι ένας δείκτης σε ακέραιο.

Για να φτιάξουμε δυναμικά έναν πίνακα με δείκτες χρειαζόμαστε ένα δείκτη που να δείχνει σε τύπο δείκτη.

```
int **ptrarray = (int**) malloc(100 * sizeof(int*));
```



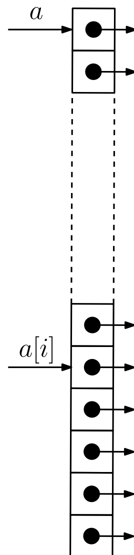
Διδιάστατος Δυναμικός Πίνακας

Για να φτιάξουμε ένα δυναμικό διδιάστατο πίνακα φτιάχνουμε πρώτα ένα πίνακα με δείκτες. Αυτός ο πίνακας αντιστοιχεί στις γραμμές του διδιάστατου πίνακα.

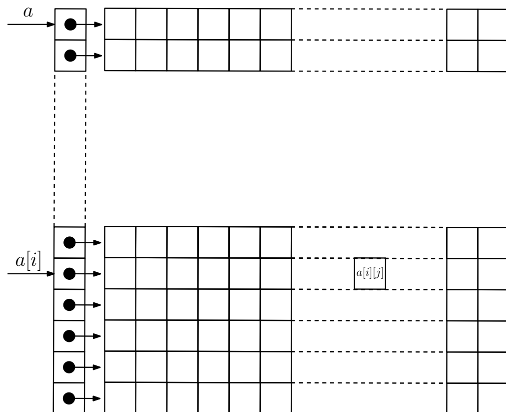
```
int **a = (int**) malloc(100 * sizeof(int*));
```

Στην συνέχεια φτιάχνουμε ένα πίνακα για κάθε γραμμή.

```
for(i = 0; i < 100; i++)
    a[i] = (int*) malloc(50 * sizeof(int));
```



Διδιάστατος Δυναμικός Πίνακας



Γράφοντας `a[i]` μας επιστρέφεται ένας δείκτης.

Γράφοντας `a[i][j]` μας επιστρέφεται ο ακέραιος που αντιστοιχεί στην θέση *i*, *j* του πίνακα.

Γιατί; Θυμηθείτε πως ο μεταγλωττιστής μετατρέπει το `a[i]` σε `*(a+i)`. Η έκφραση `a[i][j]` μετατρέπεται σε `*(*(a+i)+j)`.

Διδιάστατος Δυναμικός Πίνακας

Δημιουργία και καταστροφή διδιάστατου πίνακα.

```
#include <stdlib.h>

main() {
    int i, **a;

    a = (int**) malloc(100 * sizeof(int*));
    for(i = 0; i < 100; i++)
        a[i] = (int*) malloc(50 * sizeof(int));

    // here we can use the 100x50 array
    // a[i][j] is element at row i and column j

    for(i = 0; i < 100; i++)
       free(a[i]);
    free(a);

    return 0;
}
```

Ελευθερώνουμε την μνήμη με την αντίστροφη σειρά.

Συχνά Λάθη

Έλεγχος Τιμής Επιστροφής `malloc()`

Η `malloc()` επιστρέφει είτε την μνήμη που δεσμεύτηκε είτε `NULL` σε περίπτωση που έγινε λάθος.

Πρέπει πάντα να ελέγχουμε την τιμή επιστροφής

Σε περίπτωση που το λειτουργικό δεν έχει διαθέσιμη μνήμη η `malloc()` θα επιστρέψει `NULL`.

Εάν δεν κάνουμε έλεγχο, μόλις προσπελάσουμε αυτόν τον δείκτη το πρόγραμμα μας θα τερματιστεί βίαια.

Συχνά Λάθη

Έλεγχος Τιμής Επιστροφής malloc()

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

int main()
{
    int *x = (int*) malloc(100 * sizeof(int));
    x[3] = 15;
    free(x);
    return 0;
}
```

Εάν η malloc() επιστρέψει NULL τότε θα εκτελεστεί ο παρακάτω κώδικας που προκαλεί βίαιο τερματισμό.

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

int main()
{
    int *x = 0;  // ERROR!
    x[3] = 15;
    free(x);
    return 0;
}
```

Ο σωστός τρόπος χρήσης της συνάρτησης malloc() είναι λοιπόν:

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

int main()
{
    int *x = (int*) malloc(100 * sizeof(int));

    if (x == NULL)
    {
        fprintf(stderr, "Could not allocate memory!");
        abort();
    }

    x[3] = 15;
    free(x);
    return 0;
}
```

Συχνά Λάθη
Πολλαπλή Κλήση της `free()`

Η κλήση της συνάρτησης `free( ptr )` ελευθερώνει την μνήμη που δείχνει ο δείκτης `ptr`. Αυτή η μνήμη πρέπει να έχει πρωτύτερα δεσμευτεί με την χρήση της `malloc()`.

Σε περίπτωση που η μνήμη που δείχνει ο `ptr` έχει ήδη ελευθερωθεί από μια προηγούμενη κλήση της `free()` η συμπεριφορά του προγράμματος είναι απρόβλεπτη.

Συχνά Λάθη

Πολλαπλή Κλήση της free()

Σε περίπτωση που η μνήμη που δείχνει ο `ptr` έχει ήδη ελευθερωθεί από μια προηγούμενη κλήση της `free()` η συμπεριφορά του προγράμματος είναι απρόβλεπτη.

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

int main()
{
    int *x = (int*) malloc(100 * sizeof(int));

    if (!x) {
        fprintf(stderr, "Cannot allocate memory!");
        abort();
    }

    free(x);
    free(x);  // ERROR! undefined behavior
    return 0;
}
```