Προγραμματισμός Υπολογιστών

Δυναμική δέσμευση της μνήμης

Νικόλαος Ζ. Ζάχαρης
Καθηγητής

Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Δυναμική Δέσμευση Μνήμης (Dynamic Memory Allocation)

Στα περισσότερα εκπαιδευτικά προγράμματα χρησιμοποιούμε πίνακες οι οποίοι έχουν σταθερό μήκος, όπως παρακάτω :

Ένας πίνακας ακεραίων με μέγεθος 5 στοιχείων

```
int main(int argc, char *argv[])
{
  int pinakas[5];

  for(int i = 0; i < 5; i++)
    scanf("%d", &pinakas[i]);

  int sum = 0;
  for(int i = 0; i < 5; i++)
    sum += pinakas[i];

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

Στο πρόγραμμα δηλώνουμε ένα πίνακα 5 ακεραίων και γεμίζουμε τις αντίστοιχες θέσεις από το πληκτρολόγιο. Εν συνεχεία υπολογίζουμε το άθροισμα των στοιχείων του πίνακα.

Αν αργότερα θέλουμε να τρέξουμε το ίδιο πρόγραμμα για 7 στοιχεία θα πρέπει να πάμε σε 3 διαφορετικά σημεία στο πρόγραμμα μας και να κάνουμε το 5 σε 7.

```
#define pinLen 7

int main(int argc, char *argv[])
{
  int pinakas[pinLen ];

  for(int i = 0; i < pinLen; i++)
     scanf("%d", &pinakas[i]);

  int sum = 0;
  for(int i = 0; i < pinLen ; i++)
     sum += pinakas[i];

  printf("sum = %d",sum);
  return 0;
}
```

Μερική λύση στο πρόβλημα είναι να δηλώσουμε μια σταθερά π.χ pinLen με το αντίστοιχο μήκος του πίνακα. Εν συνεχεία όλες οι αναφορές δεν γίνονται βάση ενός ακεραίου αριθμού αλλά βάση του ονόματος της σταθεράς.

Έτσι σε περίπτωση που θέλουμε να τρέξουμε το ίδιο πρόγραμμα για 15 στοιχεία τότε θα κάνουμε μια μικρή αλλαγή στην τιμή της σταθεράς από 7 σε 15.

Τι γίνεται όμως στην περίπτωση που δεν γνωρίζουμε εξαρχής το πλήθος των στοιχείων του πίνακα;

Στα περισσότερα προβλήματα χρησιμοποιούμε δυναμικούς πίνακες με σκοπό να χρησιμοποιούμε ακριβώς την μνήμη που χρειαζόμαστε, όπως παρακάτω :

Δείκτης = (ΤύποςΔεδομένων *) malloc (Πλήθος Bytes);

```
int *array;
array = (int *) malloc(4 * sizeof(int));
if(array == NULL){
  printf("Not enough memory");
  exit(0);
}
else {
   int i;
   for(i = 0; i < 4; i++)
        scanf("%", &array[i]);
   free(array);
}
```

Η συνάρτηση **malloc** δεσμεύει δυναμικά χώρο στην μνήμη, ικανό να αποθηκεύσει **Πλήθος Bytes** που συνήθως υπολογίζονται από το γινόμενο (**Πλήθος Στοιχείων**) * (**Μέγεθος Τύπου δεδομένων**)

Η συνάρτηση **free** δέχεται σαν είσοδο ένα δείκτη και αποδεσμεύει την μνήμη όταν δεν την χρειαζόμαστε πλέον.

Η **exit(**Επιστρεφόμενη τιμή**)** τερματίζει την εκτέλεση του προγράμματος.

Το **NULL** είναι μια ειδική τιμή που ορίζεται στις περισσότερες βιβλιοθήκες των μεταγλωττιστών, όπως παρακάτω :

#define NULL 0 η εναλλακτικά σαν #define NULL -1

και χρησιμοποιείται για να συμβολίζει δείκτες οι οποίοι δεν έχουν καμία αναφορά στην μνήμη του συστήματος. Για παράδειγμα, στις παρακάτω εντολές :

```
int *array;
array = (int *) malloc( 4 * sizeof(int));
```

κατά την εκτέλεση της δεύτερης γραμμής, το πρόγραμμα αιτείται από το λειτουργικό σύστημα να διαθέσει συνεχόμενες θέσεις μνήμης, ώστε να μπορούν να αποθηκευτούν 4 ακέραιοι αριθμοί. Αν υπάρχει ελεύθερη μνήμη, τότε το λειτουργικό σύστημα θα επιστρέψει την διεύθυνση της πρώτης θέσης, το A002, στο διπλανό παράδειγμα, και μετά όλη η αιτούμενη μνήμη θα οριστεί σαν Δεσμευμένη αφού πλέον θα γίνεται χρήση από το πρόγραμμα μας, μέχρι να την αποδεσμεύσουμε με τη free.

array

Διεύθυνση	Δέσμευση
A000	Δεσμευμένη
A001	Δεσμευμένη
A002	Ελεύθερη
A003	Ελεύθερη
A004	Ελεύθερη
……..	Ελεύθερη

Αν δεν υπάρχει διαθέσιμη συνεχόμενη μνήμη για 4 ακεραίους αριθμούς τότε το λειτουργικό σύστημα επιστρέφει την τιμή NULL δηλαδή το 0 (ή το -1), η οποία είναι μια μη έγκυρη τιμή, στην μνήμη του συστήματος.

Γιατί χρειαζόμαστε μόνο την αρχή της δεσμευμένης μνήμης;

Επειδή όλα τα στοιχεία του πίνακα είναι του ιδίου τύπου (π.χ. int, long, char, float, double) και είναι σε συνεχόμενες θέσεις , μπορούμε από τη σχέση

Αρχή_Μνήμης_Πίνακα + (Στοιχείο * Μέγεθος Στοιχείου)

μπορούμε να προσπελάσουμε όλα τα στοιχεία του πίνακα. Αν θεωρήσουμε ότι υπάρχει ο πίνακα CharArray ο οποίος έχει 4 στοιχεία όπου το κάθε στοιχείο χρειάζεται 1 Byte μνήμης. Τότε αν γράψουμε CharArray[2] = 'b' θα αποθηκεύσουμε την τιμή 'b' στην θέση μνήμης A004

Αρχή Πίνακα CharArray →

Array[2] = A002 + (2 * 1 Byte) →

Διεύθυνση	Τιμή	Δέσμευση
A000		Χρησιμοποιείται
A001		Χρησιμοποιείται
A002		Χρησιμοποιείται
A003		Χρησιμοποιείται
A004	'b'	Χρησιμοποιείται
A005		Χρησιμοποιείται

Στη συνέχεια θα δούμε ένα ολοκληρωμένο το πρόγραμμα όπου δηλώνουμε ένα πίνακα ακεραίων μεταβλητού πλήθους και γεμίζουμε τις αντίστοιχες θέσεις από το πληκτρολόγιο. Εν συνεχεία υπολογίζουμε το άθροισμα των στοιχείων του πίνακα.

```
int main(int argc, char *argv[]){
  int *pin;
  int pLen, sum = 0;

  printf(" Type number of elements : ");
  scanf("%d", &pLen);


  if((pin = (int *)malloc(pLen * sizeof(int)) == NULL)
     printf("Error in memory allocation\n");
  else {
     for(int i = 0; i < pinLen; i++)
         scanf("%d", &pin[i]);

     for(int i = 0; i < pinLen ; i++)
         sum += pin[i];

      printf("sum = %d\n" sum);

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

Η δήλωση του δείκτη για το πίνακα των ακεραίων.

Το επιθυμητό πλήθος των στοιχείων του πίνακα. Αυτή η μεταβλητή δεν αλλάζει τιμή καθ' όλη την διάρκεια του προγράμματος.

Η δέσμευση της μνήμης για το συγκεκριμένο πλήθος στοιχείων. Αν δεν υπάρχει διαθέση μνήμη τότε το σύστημα επιστρέφει την τιμή NULL.

Είναι αναγκαίο να αποδεσμεύσουμε τη μνήμη που δεν χρειάζεται πλέον στο πρόγραμμα μας.

Η δέσμευση της μνήμης μπορεί να γίνει και μέσα σε μια συνάρτηση η οποία μπορεί να επιστρέφει σαν τιμή το δείκτη της δεσμευμένης μνήμης.

```
int *readPinaka(int plithos) {
   int *pin;

   if((pin = (int *) malloc(plithos *
sizeof(int)) == NULL)   {
       return NULL;
   }
   else
   {
       for(int i = 0; i < plithos; i++)
           scanf("%d", &pin[i]);

       return pin;
   }
}
```

```
int main(int argc, char *argv[]){
   int *pinakas;

   if((pinakas = readPinaka(5)) == NULL)
   {
      printf("Error in memory allocation.");
      exit(0);
   }

   for(int i = 0; i < 5; i++)
      printf("%d\t",pinakas[i]);

   free(pinakas);

    return 0;
}
```

Θα πρέπει να ΜΗΝ ΑΠΟΔΕΣΜΕΥΣΟΥΜΕ (free) την μνήμη μέσα στην συνάρτηση.

Η επιστροφή ενός δείκτη από μια συνάρτηση είναι πολύ συνηθισμένη διαδικασία στις συμβολοσειρές (strings).

```
char *myStrCpy(char *source)
{
   char *dupl;
   int i, strPlithos = strlen(source);

   if((dupl = (char *) malloc(strPlithos+1)) == NULL)   {
      return NULL;
   }
   else
   {
      for(i = 0; i < strPlithos; i++)
         dupl[i] = source[i];

      dupl[i] = '\0';

      return dupl;
   }
}
```

```
int main(int argc, char *argv[]){
  char *copyString;

  if((copyString = nzStrCpy("Hello")) == NULL)
  {
      printf("Error in memory allocation.");
  }

  printf("%s\n", copyString);
  printf("%d\n", strlen(copyString));

  free(copyString);
  return 0;

}
```

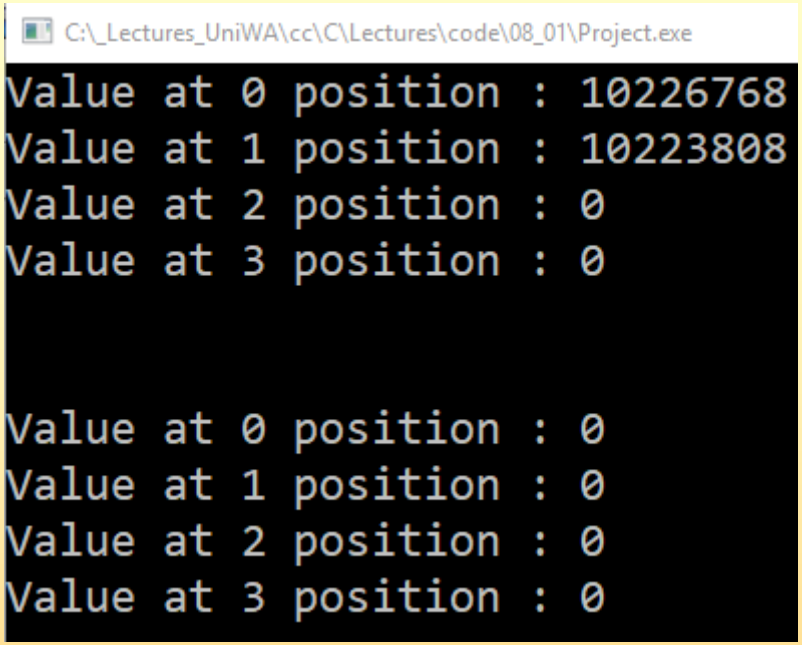
Η επίδειξη της δυναμικής δέσμευσης της μνήμης έγινε κατά κύριο λόγο με την χρήση των πινάκων. Αυτό δεν σημαίνει ότι δεν μπορούμε να δεσμεύσουμε έναν απλό τύπο δεδομένων όπως για παράδειγμα ένα ακέραιο ή ένα χαρακτήρα.

```
int main(int argc, char *argv[]) {
  int *ptrInt = (int *)malloc(sizeof(int));
  char *ptrChar = (char *)malloc(sizeof(char));
  *ptrInt = 5;
  *ptrChar = 'M';

  printf("%d\n", *ptrInt);
  printf("%c\n", *ptrChar);

  free( ptrInt);
  free( ptrChar);
  return 0;
}
```

Αρχικοποίηση της δεσμευμένης μνήμης

Η συνάρτηση malloc επιτρέπει τη δυναμική δέσμευση μνήμης κατά τη διάρκεια εκτέλεσης ενός προγράμματος αλλά δεν αρχικοποιεί τη μνήμη, οπότε σε αυτές τις θέσεις μέχρι να αναθέσουμε κάποια τιμή δεν πρέπει να θεωρούμε ότι είναι 0.

Στο διπλανό παράδειγμα οι 4 πρώτες γραμμές εκτυπώνουν τα περιεχόμενα ενός πίνακα που έχει δεσμευτεί με τη malloc



Εν αντιθέσει η συνάρτηση calloc κάνει ότι και η malloc και επιπλέον αρχικοποιεί τη μνήμη με την τιμή 0. Στο παράδειγμα οι 4 τελευταίες γραμμές εκτυπώνουν τα περιεχόμενα ενός πίνακα που έχει δεσμευτεί με τη calloc. Η σύνταξη της συνάρτησης είναι :

void *calloc(size_t Πλήθος, size_t Μέγεθος ενός στοιχείου)

Η συνάρτηση επιστρέφει τη διεύθυνση της νέας δεσμευμένης μνήμης ή NULL

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


int main(int argc, char *argv[]) {
int *array = NULL;
int i;

array = (int *) malloc(4 * sizeof(int));

if(array == NULL){
  printf("Not enough memory");
  exit(0);
}

for(i = 0; i < 4; i++) {
   printf("Value at %d position : %d\n", i , array[i]);
}

free(array);

printf("\n\n");
```

Η συνάρτηση malloc

```c
array = (int *) calloc(4, sizeof(int));

if(array == NULL){
  printf("Not enough memory");
  exit(0);
}

for(i = 0; i < 4; i++) {
   printf("Value at %d position : %d\n", i ,
array[i]);
}

free(array);

  return 0;
}
```

Η συνάρτηση calloc

Και στις δύο περιπτώσεις, η διαχείριση είναι ή ίδια, η διαφορά είναι στο τρόπο λειτουργίας των δύο συναρτήσεων.

Αλλαγή μεγέθους της δεσμευμένης μνήμης

Η συνάρτηση malloc επιτρέπει τη δυναμική δέσμευση μνήμης κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Εάν σε περίπτωση χρειάζεται μετά τη δέσμευση μνήμης να μεγαλώσουμε η να μικρύνουμε το μέγεθός της, τότε μπορούμε να κάνουμε ένα από τα δύο :

α) να δημιουργήσουμε ένα δεύτερο δείκτη με το επιθυμητό μέγεθος, να αντιγράψουν σε αυτόν τα στοιχεία από τον πρώτο δείκτη, να απελευθερώσουμε τη μνήμη του πρώτου δείκτη, να δεσμεύσουμε μνήμη για το πρώτο δέκτη ίση με το επιθυμητό μέγεθος, να αντιγράψουμε τα στοιχεία από το δεύτερο δείκτη στον πρώτο δείκτη, και τέλος να ελευθερώσουμε τη μνήμη του δεύτερου δείκτη.

β) α χρησιμοποιήσουμε τη συνάρτηση realloc η οποία συντάσσεται όπως παρακάτω :

void *realloc(void *ptr, size_t size)

ptr – ο δείκτης του οποίου που πρόκειται να αλλάξουμε το μέγεθος.

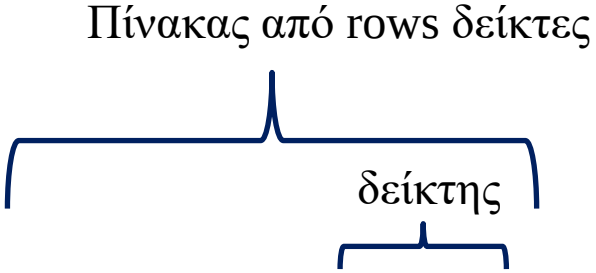
size – το επιθυμητό μέγεθος μνήμης (είναι 0 ή θετικός αριθμός)

Η συνάρτηση επιστρέφει τη διεύθυνση της νέας δεσμευμένης μνήμης ή NULL

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

int main(int argc, char *argv[]) {
int *array = NULL;

array = (int *) malloc(4 * sizeof(int));

if(array == NULL){
  printf("Not enough memory");
  exit(0);
}

printf("Address of %p\n", array);

int i;
for(i = 0; i < 4; i++) {
   printf("Type %d value : ", i + 1);
   scanf("%d", &array[i]);
}

array = (int *) realloc(array, sizeof(int) * 6);
printf("Address of %p\n", array);

for(i = 4; i < 6; i++) {
   printf("Type %d value : ", i + 1);
   scanf("%d", &array[i]);
}

printf("\n\n");

for(i = 0; i < 6; i++) {
   printf("Value at %d position : %d\n",
                                i , array[i]);
}

free(array);

  return 0;
}
```

Δυναμική δημιουργία πίνακα 2 διαστάσεων (1-2)

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

int main(int argc, char *argv[]) {
  int **pin2D;
  int i,j,  rows, cols;

   printf("Type rows : ");
   scanf("%d", &rows);
   printf("Type cols : ");
   scanf("%d", &cols);

   pin2D = (int **) malloc(rows * sizeof(int *));

   if(pin2D == NULL) { printf("Not enough memory."); exit(0); }

   for (i=0; i < rows; i++) {
      pin2D[i] = (int *) malloc(cols * sizeof(int));

      if(pin2D[i] == NULL) { printf("Not enough memory."); exit(0); }
   }
```



Δυναμική δημιουργία πίνακα 2 διαστάσεων (2-2)

```
    for (i=0; i<rows; i++) {
        for (j=0; j<cols; j++) {
            printf("Type value pin[%d][%d]", i, j);
            scanf("%d", &pin2D[i][j]);
        }
    }

    int sum = 0;
    for (i=0; i<rows; i++)  {
        for (j=0; j<cols; j++) {
            sum += pin2D[i][j];
        }
    }

    printf("\nSum = %d\n", sum);

    for (i=0; i<rows; i++)
    {
        free(pin2D[i]);
    }

    free(pin2D);

    return 0;
}
```

```
Type rows : 2
Type cols : 3
Type value pin[0][0]23
Type value pin[0][1]7
Type value pin[0][2]5
Type value pin[1][0]6
Type value pin[1][1]7
Type value pin[1][2]3

Sum = 51
```

Στο τέλος θα πρέπει να αποδεσμεύσουμε όλη τη μνήμη :

Α) για τη κάθε μια θέση του πίνακα

Β) το συνολικό πίνακα

Δυναμική δημιουργία πίνακα 3 διαστάσεων (1-2)

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

int main(int argc, char *argv[]) {
  int ***pin3D;
  int i,j,k,  rows, cols, height;

   printf("Type rows : ");
   scanf("%d", &rows);
   printf("Type cols : ");
   scanf("%d", &cols);
   printf("Type height : ");
   scanf("%d", &height);

   pin3D = (int ***) malloc(rows * sizeof(int **));

   if(pin3D == NULL) { printf("Not enough memory."); exit(0); }

   for (i=0; i < rows; i++) {
        pin3D[i] = (int **)malloc(cols * sizeof(int *));

        if(pin3D[i] == NULL) { printf("Not enough memory."); exit(0); }

        for (j =0; j < cols; j++) {
            pin3D[i][j] = (int *) malloc(height * sizeof(int));

            if(pin3D[i][j] == NULL) { printf("Not enough memory."); exit(0); }
        }
    }
```

Δυναμική δημιουργία πίνακα 3 διαστάσεων

```
  for (i=0; i < rows; i++) {
     for (j=0; j < cols; j++){
        for (k=0; k < height; k++){
           printf("Type value pin[%d][%d][%d]", i, j, k);
           scanf("%d", &pin3D[i][j][k]);
        }
     }
  }

  int sum = 0;
  for (i=0; i<rows; i++)  {
     for (j=0; j<cols; j++) {
       for (k=0; k< height; k++) {
         sum += pin3D[i][j][k];
       }
     }
  }
  printf("\nSum = %d\n", sum);

  for (i=0; i < rows; i++) {
       for (j =0; j < cols; j++) {
           free( pin3D[i][j] ) ;
       }
       free(pin3D[i]);
   }

   free(pin3D);
  return 0;
}
```

```
Type rows : 2
Type cols : 2
Type height : 2
Type value pin[0][0][0]2
Type value pin[0][0][1]3
Type value pin[0][1][0]4
Type value pin[0][1][1]5
Type value pin[1][0][0]6
Type value pin[1][0][1]7
Type value pin[1][1][0]8
Type value pin[1][1][1]9

Sum = 44
```