Δομημένος προγραμματισμός

Δυναμική διαχείριση μνήμης

Περίγραμμα της ύλης που θα καλυφθεί

- Δυναμική Διαχείριση Μνήμης
- Συναρτήσεις για δυναμική διαχείριση μνήμης
  - malloc(),
  - free(),
  - calloc() και
  - realloc()
- Δεσμεύοντας μνήμη για
  - Δομές και
  - Πίνακες

Κατηγορίες μνήμης εκτελέσιμου προγράμματος

- Στις *καθολικές* και *στατικές* μεταβλητές οι χώροι μνήμης δεσμεύονται κατά την διάρκεια της μεταγλώττισης.
- Οι τοπικές μεταβλητές δεσμεύονται προσωρινά όσο διαρκεί η εκτέλεση της συνάρτησης που τις περιέχει.
- Όταν εκτελείται ένα πρόγραμμα η μνήμη που καταλαμβάνει χωρίζεται τυπικά σε 4 μέρη:
  - *Στη μνήμη που καταλαμβάνει ο εκτελέσιμος κώδικας (code area).*
  - *Στη μνήμη για τις καθολικές και στατικές μεταβλητές (global area).*
  - *Στη μνήμη όπου αποθηκεύονται οι τοπικές μεταβλητές και οι παράμετροι συναρτήσεων. Ονομάζεται «στοίβα» (stack).*
  - *Στη μνήμη για τις μεταβλητές που δεσμεύονται με δυναμική διαχείριση. Πρόκειται για ελεύθερη μνήμη και ονομάζεται «σωρός» (heap).*

Ο σωρός (heap)

- Ελεύθερη μνήμη που δεν χρησιμοποιείται από κανένα άλλο πρόγραμμα (ούτε από το λειτουργικό σύστημα).
- Παρέχει τμήματα μνήμης όποτε ζητηθούν από το πρόγραμμα (*δυναμική εκχώρηση*).
- Αξιοποιώντας τον *σωρό* μπορεί κάποιος να κάνει εξοικονόμηση μνήμης.
- Ο χώρος που καταλαμβάνει ο *σωρός* εξαρτάται από τον μεταγλωττιστή. Συνήθως είναι αρκετά μεγάλος, αλλά όχι απεριόριστος.
- Η πρόσβαση στο τμήμα του *σωρού* είναι πιο αργή σε σχέση με αυτή της *στοίβας*.
- Είναι ευθύνη του προγραμματιστή η αποτελεσματική διαχείριση μνήμης.
- Είναι δυνατή η αλλαγή μεγέθους μεταβλητής.

Δυναμική διαχείριση μνήμης

- Μέχρι τώρα στα προγράμματα μας αναθέταμε τη μεγαλύτερη μνήμη που ενδεχόμενα χρειαζόταν οι μεταβλητές μας:

```
#define N 100
……..
……..
int Pin[N]; //δήλωση πίνακα. Το μέγεθος δεν μπορεί ν' αλλάξει
```

Στατική δέσμευση μνήμης (στη στοίβα)

- Η δήλωση της μεγαλύτερης πιθανής μνήμης έχει επιπτώσεις στην εξοικονόμηση μνήμης και σε ταχύτητα εκτέλεσης.
- Η λύση σε ένα τέτοιο ενδεχόμενο (δηλαδή όταν είναι άγνωστο το μέγεθος της μνήμης που θα χρειαστούμε) είναι η *δυναμική δέσμευση μνήμης* (*dynamic memory allocation*).
- Στη βιβλιοθήκη *stdlib.h*, υπάρχουν συναρτήσεις με τις οποίες μπορούμε να ζητήσουμε δυναμικά την παραχώρηση και την αποδέσμευση χώρου μνήμης από τον *σωρό*.
- Πρόκειται για πολύ χρήσιμη δυνατότητα σε περιπτώσεις που δεν γνωρίζουμε εκ των προτέρων το μέγεθος της μνήμης που χρειαζόμαστε.

Συναρτήσεις για δυναμική διαχείριση μνήμης

malloc ()	Δυναμική δέσμευση συγκεκριμένου πλήθους bytes
calloc ()	Δυναμική δέσμευση πλήθους τμημάτων μνήμης
realloc ()	Αλλαγή μεγέθους τμήματος δεσμευμένης μνήμης
free ()	Απελευθέρωση μνήμης που δεσμεύτηκε δυναμικά

- Η διαχείριση μνήμης γίνεται με αίτημα προς το λειτουργικό σύστημα.
- Η μέγιστη μνήμη που μπορούμε να ζητήσουμε εξαρτάται από:
  - *την συνολική μνήμη του συστήματος,*
  - *πόση μνήμη είναι διαθέσιμη για προγράμματα χρηστών,*
  - *πόσα άλλα προγράμματα τρέχουν ταυτόχρονα και πόση μνήμη έχουν δεσμεύσει.*
- Με την δυναμική διαχείριση μνήμης, εκτός του ότι δεν χρειάζεται να γνωρίζουμε το μέγεθος της μνήμης που θα χρειαστεί, μπορούμε επίσης ν' αλλάξουμε αυτό το μέγεθος.

Η συνάρτηση malloc() – memory allocation

- Για τη δέσμευση τμήματος μνήμης στο *σωρό* (*heap*). Δέχεται σαν όρισμα το μήκος του τμήματος σε Bytes.
- Το πρωτότυπο της συνάρτησης:

*void *malloc(size_t size);*

- Παράμετρος *size*: πόσα Bytes θα δεσμευτούν
- Τύπος *size_t*: συνώνυμο του *unsigned integer*.
- Η *malloc()* επιστρέφει ένα δείκτη στη μνήμη που δεσμεύτηκε (στο 1ο Byte). Αυτός ο δείκτης είναι τύπου void (*void *).
- Αν δεν υπάρχει επαρκής μνήμη, τότε επιστρέφει ένα κενό (*NULL*) δείκτη. Πρέπει πάντα να γίνεται αυτός το έλεγχος:

Χρήση του NULL δείκτη ίσως προκαλέσει κατάρρευση του προγράμματος

```
p = (float *) malloc(40);
if (p==NULL)  {          //if (!p)
        printf("Allocation error");
exit (1);
 }
```

Πρόγραμμα με χρήση της malloc()

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main() {
  char *str;
  str = (char *) malloc(15);
  if (!str){
        printf("Allocation error");
        exit (1);
  }
  strcpy(str, “ELMEPA");
  puts(str);
  printf("%c\n", str[0]);
  putchar(str[1]);
  free(str);
}
```

Στο πρόγραμμα δεσμεύονται 15 χαρακτήρες (Bytes)

Καλύτερα:
str = (char *) malloc(15*sizeof(char));

Μπορείτε και:
gets(str)

Στη πραγματικότητα δημιουργείται δυναμικά ένας μονοδιάστατος πίνακας χαρακτήρων!

Το τμήμα μνήμης αποδεσμεύεται

Η συνάρτηση free()

- Η συνάρτηση *free()* καλείται για την απελευθέρωση κάποιας δεσμευμένης μνήμης.
- Πρωτότυπο της συνάρτησης:

*void free(void *ptr);*

Χρήση:
```
free(ptr);
```

- Η παράμετρος *ptr* δείχνει στη αρχή του τμήματος της μνήμης που πρόκειται ν' απελευθερωθεί με τη *free().*
- Όταν τελειώνει ένα πρόγραμμα, η δεσμευμένη μνήμη αποδεσμεύεται αυτόματα.
- Όμως εάν δεν χρειαζόμαστε την μνήμη που δεσμεύσαμε τότε καλύτερα να την αποδεσμεύουμε με τη συνάρτηση *free().*
- Αν ο δείκτης-παράμετρος της *free()* ΔΕΝ δείχνει σε δυναμικά δεσμευμένη μνήμη, τότε προκαλείται πρόβλημα στην εκτέλεση του προγράμματος (όπως επίσης αν αποπειραθούμε να ελευθερώσουμε μνήμη που έχει ήδη αποδεσμευτεί).

Ο δείκτης τύπου void (void *)

- Ο *δείκτης void* υποδηλώνει ότι *δεν είναι γνωστό* το είδος της μνήμης που δεσμεύεται (γενικός δείκτης).
- Επιτρέπει στη συνάρτηση να επιστρέφει δείκτη σε μνήμη, χωρίς να προσδιορίζεται ο τύπος δεδομένων που θα φιλοξενηθούν.
- Στο προηγούμενο πρόγραμμα στη εντολή:

  *str = (char *) malloc(15);*

  μετατρέπεται ο δείκτης *void* σε δείκτη δεδομένων τύπου χαρακτήρα. *Αν και δεν είναι απαραίτητο στην ANSI C, η πρακτική αυτή προτείνεται για λόγους συμβατότητας με τη C++.*
- Είναι διαφορετικός από το void:
  - *void* - δεν επιστρέφει τίποτα η συνάρτηση.
  - *void* * - απλά επιστρέφεται μια διεύθυνση μνήμης, χωρίς να είναι γνωστό τι τύπου δεδομένα θα αποθηκευτούν εκεί.

Η συνάρτηση calloc() – clear memory allocation

- Η *calloc()* όπως και η *malloc()* μας δίνει την δυνατότητα να δεσμεύσουμε μνήμη.
- Το πρωτότυπο:

*void *calloc(size_t num, size_t size);*

- Το ποσό της μνήμης που δεσμεύεται είναι: *num*size* (*ουσιαστικά μια διάταξη από num στοιχεία που το καθένα τους έχει μέγεθος size*).
- Παράδειγμα χρήσης:

Όπως και στη *malloc()* αν δεν υπάρχει επαρκής μνήμη επιστρέφει ένα *NULL* (κενό) δείκτη.

Δέσμευση 60 bytes

```
ptr = (int *) calloc(15, sizeof(int));
  if (!ptr){
        printf("Allocation error");
        exit (1);
  }
```

- Η *calloc()* σε αντίθεση με την *malloc()* δίνει μηδενικές αρχικές τιμές στη μνήμη που δεσμεύει.

Πρόγραμμα με χρήση της calloc()

```
#include <stdio.h>
#include <stdlib.h>
main() {
  char *str;
  str = (char *) calloc(15, sizeof(char));
  if (!str){
        printf("Allocation error");
        exit (1);
  }
  printf("String is zero: ");  puts(str);
  printf("Give me a string:");   gets(str);
  printf("New string: "); puts(str);
  printf("%c\n", str[0]);
  putchar(str[1]);
  free(str);
}
```

Στο πρόγραμμα δεσμεύονται 15 χαρακτήρες (Bytes)

Με malloc:
str = (char *) malloc(15*sizeof(char));

Προσοχή:
αν δώσουμε
putchar(str[20]);
δεν θα βγει προφανώς σωστό αποτέλεσμα (εκτός ορίων!)

Η συνάρτηση realloc() – [memory] reallocation

- Με την *realloc()* μπορούμε να αλλάξουμε το μέγεθος μιας ήδη δεσμευμένης μνήμης.
- Πρωτότυπο:

*void *realloc(void ptr, size_t size);*

- Η παράμετρος *ptr* δείχνει στη αρχή του τμήματος της μνήμης που έχει ήδη δεσμευτεί (pointer).
- Η τιμή της παραμέτρου *size* μπορεί να είναι μεγαλύτερη ή μικρότερη από το αρχικό μέγεθος μνήμης που δεσμεύτηκε αρχικά.
- Η συνάρτηση επιστρέφει ένα δείκτη (pointer) σε περιοχή μνήμης.
- Αυτός ο pointer μπορεί να δείχνει σε διεύθυνση διαφορετική από την αρχική - αν δεν βρεθεί αρκετός χώρος στη αρχική διεύθυνση, η *realloc()* μεταφέρει το τμήμα της μνήμης σε άλλη διεύθυνση.

Πρόγραμμα με τη συνάρτηση realloc()

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main() {
   char *str;
   str = (char *) malloc(15*sizeof(char));
    if (!str){   printf("Allocation error");
                 exit (1);
    }
    strcpy(str, “ELMEPA");
    printf("String = %s,  physical address = %p\n", str, str);
    str = (char *) realloc(str, 50*sizeof(char));
    strcat(str, ", Electrical Engineering");
    printf("String = %s,  physical address = %p\n", str, str);
    free(str);
}
```

Στο πρόγραμμα δεσμεύονται αρχικά 15 Bytes. Στη συνέχεια 50.

Αρχική δέσμευση

Και εδώ πρέπει να γίνεται έλεγχος

Αλλαγή του μεγέθους της μνήμης (μεγαλύτερο)

Αποδέσμευση μνήμης

Προσοχή με την επιστροφή της realloc()

- Κανονικά η τιμή που επιστρέφει η *realloc()* δεν πρέπει ν' ανατίθεται στον ίδιο δείκτη της μνήμης που δεσμεύτηκε αρχικά.
- Αυτό διότι σε περίπτωση αποτυχίας της *realloc*() θα επιστραφεί η τιμή *NULL*, με αποτέλεσμα να χαθούν τυχόν δεδομένα που αποθηκεύτηκαν.
- Είναι καλύτερα ν' αναθέτουμε επιστροφή της *realloc()* σε έναν άλλο (προσωρινό) pointer και αν ο έλεγχος είναι επιτυχής, τότε η διεύθυνση του νέου χώρου εκχωρείται στον αρχικό δείκτη:

```
char *str, *tmpstr;
str = (char *) malloc(15*sizeof(char));
……….
tmpstr = (char *) realloc(str, 50*sizeof(char));
if (tmpstr ==NULL) printf ("realloc: Allocation error");
else str=tmpstr;
```

Δεσμεύοντας μνήμη για δομές

```
struct stud {
        int id;
        char name[30];
};
main(){
   struct stud *ptr;
   int i,n;
   printf("Enter n: ");   scanf("%d",&n);
   ptr=(struct stud*)malloc(n*sizeof(struct stud));
   for(i=0;i<n;++i){
        printf("Student %d ===\n",i+1);
        printf("    Enter name:");  scanf("%s",&(ptr+i)->name);
        printf("    Enter ID:");    scanf("%d",&(ptr+i)->id);
    }
 printf("\nDisplaying Info:\n");
 for(i=0;i<n;++i)
         printf("%s\t%d\t\n",(ptr+i)->name,(ptr+i)->id);
  free(ptr);
```

Στο πρόγραμμα δεσμεύεται μνήμη για n δομές. Το n δίνεται από τον χρήστη

Δέσμευση μνήμης για n δομές τύπου stud

Δυναμική δημιουργία μονοδιάστατων πινάκων

```
int *p, i, N;
 printf("How many? ");
 scanf("%d", &N);
 p = (int *) malloc(N*sizeof(int));
 if (!p){
        printf("Allocation error");
        exit (1);
 }
 for (i=0;i<N;i++) {
    printf("Please give %d:",i+1);
    scanf("%d",&p[i]);
 }
 printf("Allocated: ");
 for (i=0;i<N;i++) {
      printf("%d ",p[i]);
 }
 free(p);
```

Στο πρόγραμμα δεσμεύονται Ν ακέραιοι

Η δέσμευση Ν ακεραίων. Ουσιαστικά έχει δημιουργηθεί δυναμικά ένας μονοδιάστατος πίνακας Ν ακεραίων!

Εναλλακτικά:
scanf("%d", p+i);

Εναλλακτικά:
printf("%d ", *(p+i));

Το τμήμα μνήμης αποδεσμεύεται

Δυναμική δημιουργία 2Δ πινάκων

```
int **p, i, j, M, N;
printf("Line size:");  scanf("%d", &M);
p = (int **) malloc(M*sizeof(int *));  //δέσμευση M γραμμών (δεικτών)
if (p == NULL){ printf("malloc lines: Error\n"); exit(1); }
printf("Column size:"); scanf("%d", &N);
for (i=0;i<M;i++) {
     p[i]= (int *) malloc(N*sizeof(int *)); //δέσμευση N στηλών ανά γραμμή
     if (p == NULL){ printf("malloc column:%d Error\n",i); exit(1); }
}
for (i=0;i<M;i++)
    for (j=0;j<N;j++)
           p[i][j]=rand()%20;   //δίδονται τυχαίες τιμές στο πίνακα
for (i=0;i<M;i++){
       for (j=0;j<N;j++)      printf("%5d", p[i][j]);
       printf("\n");
}
for (i=0;i<M;i++)  free(p[i]);
free(p);
```

Προσέξτε την αποδέσμευση

str[] ή *str (διαφορές) και εγγραφή σε αρχείο

```
main () {
  FILE *fp;
  int c;
  char str []="Hello!";
  if ((fp=fopen("store.txt", "w+"))==NULL){
        printf("Error opening file!");
        exit(1);
   }
  fputs(str, fp);
  fclose(fp);
  if ((fp=fopen("store.txt", "r"))==NULL) {
        printf("Error opening file!");
        exit(1);
   }
  if( fgets (str, 7, fp)!=NULL ) puts(str);
  fclose(fp);
}
```

```
main () {
  FILE *fp;
  char *str;
  if ((fp=fopen("store.txt", "w+"))==NULL) {
        printf("Error opening file!");
        exit(1);
   }
  str="Hello!";
  fputs(str, fp);
  fclose(fp);
  if ((fp=fopen("store.txt", "r"))==NULL) {
        printf("Error opening file!");
        exit(1);
   }
  str = (char*)malloc(7);  //το str είναι pointer
  if( fgets (str, 7, fp)!=NULL ) puts(str);
  fclose(fp);
}
```

Πόση μνήμη έχω ακόμα?

```
#include <stdio.h>
#include <stdlib.h>
main()
{
  char *str;
  long MEM = 1000;

do {
        str = (char *) malloc(MEM);
      // printf("mem:%u\n", MEM); //προαιρετικό
        if (str) MEM += 1000; //Αυξάνω τη μνήμη
  } while (str);
  printf ("About %ld bytes of free memory! ",MEM);
  free(str);
}
```