Η ΓΛΩΣΣΑ C

Μάθημα 21:

Προχωρημένα Θέματα Δεικτών

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Θεωρία

1. Δείκτες Γενικού σκοπου (void *)
   1. Δήλωση και Χρήση
   2. Παρατηρήσεις
   3. Συναρτήσεις γενικού σκοπού
2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης
   1. Η συνάρτηση malloc
   2. Η συνάρτηση calloc
   3. Η συνάρτηση realloc
   4. Παράδειγμα
3. Συναρτήσεις με Μεταβλητό Αριθμό Ορισμάτων
   1. Ορισμός Συνάρτησης
   2. Δομικά Στοιχεία
   3. Παράδειγμα

Β. Ασκήσεις

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

1. Δήλωση και Χρήση

- Ως τώρα έχουμε μελετήσει μεταβλητές, τις οποίες πάντα τις σχετίζουμε με τον τύπο δεδομένων τους.
- Ωστόσο μπορούμε να ορίσουμε έναν τύπο δείκτη, ο οποίος δεν σχετίζεται με τύπο δεδομένων:
  - Είναι ο δείκτης γενικού σκοπού (generic pointer) και είναι τύπου void *
- Ένας void * δείκτης δηλώνεται ως:

```
void *ptr_name;
```

- Και μπορούμε να τον θέσουμε να δείχνει σε μεταβλητή οποιουδήποτε τύπου δεδομένων!
  - π.χ. σε έναν ακέραιο x:

```
ptr_name = &x;
```

  - και έπειτα κάνοντας αλλαγή τύπου, μπορούμε να τυπώσουμε τα δεδομένα στα οποία δείχνει:

```
printf("%d", *(int *)ptr_name);
```

    - Πρώτα κάνουμε αλλαγή τύπου στον δείκτη (int *)ptr_name ώστε να δείχνει σε ακέραιο
    - και πάνω σε αυτό, με τον τελεστή *, μπορούμε να πάρουμε την τιμή.

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

1. Δήλωση και Χρήση (Παράδειγμα)

- Βλέπουμε ένα παράδειγμα όπου ο ίδιος δείκτης χρησιμοποιείται για να εκτυπώσουμε έναν ακέραιο και έναν πραγματικό:

```
/* generic_pointer.c: Επίδειξη ενός δείκτη γενικού σκοπού */

#include <stdio.h>

main()
{
   int x = 4;
   double d = 1.1;

   void *p;

   p=&x;
   printf("%d", *(int *)p);

   p=&d;
   printf("\n%lf", *(double *)p);

}
```

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

2. Παρατηρήσεις

- **Δεν μπορούμε να εκτυπώσουμε το περιεχόμενο ενός void * δείκτη**, αν δεν κάνουμε πρώτα αλλαγή τύπου.
- Μπορούμε ωστόσο να τυπώσουμε τη διεύθυνση που δείχνει ο δείκτης αυτός, καθώς και τη διεύθυνση του ίδιου του δείκτη.

```
/* voidptr_printing.c */

#include <stdio.h>

int main()
{
   int x = 4;

   void *p = &x;

   //printf("%d", *p); // Δεν δουλεύει
   printf("Diefthinsi toy p: %p", &p);
   printf("\nDiefthinsi poy deixnei o p: %p", p);

   return 0;
}
```

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

2. Παρατηρήσεις

- Το πρότυπο της C **δεν επιτρέπει να κάνουμε αριθμητική σε void * δείκτες.**
- Ωστόσο πολλοί μεταγλωττιστές το επιτρέπουν, θεωρώντας το μέγεθος ενός void * δείκτη ίσο με το 1.
- Βλέπουμε και ένα ενδιαφέρον παράδειγμα:

```
/* voidptr_accessing_struct.c */

#include <stdio.h>

struct point
{
   int x;
   int y;
};
```

```
int main()
{
   struct point simeio;
   void *p = &simeio;

   simeio.x = 1; simeio.y = 2;

   printf("x=%d", *(int *)p);
   p+=sizeof(int);
   printf(" y=%d", *(int *)p);

   return 0;
}
```

- Σημείωση: Αν ο μεταγλωττιστής δεν το επιτρέπει, μπορούμε να κάνουμε μετατροπή με έμμεσο τρόπο χρησιμοποιώντας δείκτη σε χαρακτήρα.

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

3. Συναρτήσεις γενικού σκοπού

- Με τους δείκτες γενικού σκοπού:
  - Μπορούμε να υλοποιήσουμε συναρτήσεις γενικού σκοπού
  - που θα δουλεύουν ανεξάρτητα από τον τύπο δεδομένων.
- Συγκεκριμένα αυτό γίνεται συνήθως ως εξής:
  - Υλοποιούμε έναν απαριθμητή (enumerator) με τους τύπους δεδομένων που θέλουμε να υποστηρίζει το πρόγραμμά μας
  - Π.χ.:

```
enum DATA_TYPE { INT, FLOAT, DOUBLE };
```

  - Έπειτα, αφού μπορούμε να ορίζουμε μεταβλητές του απαριθμητή, π.χ.:

```
DATA_TYPE x = INT;
```

  - Κατασκευάζουμε την συνάρτηση μας ώστε να παίρνει όρισμα τον τύπο δεδομένων, π.χ.:

```
void func(arg1, arg2, …, DATA_TYPE t)
```

  - και διακρίνουμε περιπτώσεις ανάλογα με την τιμή του ορίσματος.

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

3. Συναρτήσεις γενικού σκοπού

- Σημείωση:
  - Δεν είναι υποχρεωτικό να χρησιμοποιήσουμε enumerator, αλλά σίγουρα είναι πιο κομψό.

- Βλέπουμε και ένα ολοκληρωμένο παράδειγμα, στο οποίο η ίδια συνάρτηση μπορεί να χρησιμοποιηθεί για να ενεργήσει σε διαφορετικούς τύπους δεδομένων:

```
/* voidptr_inc.c */

#include <stdio.h>

enum DATA_TYPE { INT, DOUBLE, FLOAT };

void increase(void *number, enum DATA_TYPE d);
```

Α. Θεωρία

1. Δείκτες γενικού σκοπού (void *)

3. Συναρτήσεις γενικού σκοπού

```
int main()
{
   int i=5;
   float f = 3.1;
   double d = 5.2;
   void *p;

   p = &i;
   increase(p, INT);
   printf("%d\n", *(int *)p);

   p = &f;
   increase(p, FLOAT);
   printf("%f\n", *(float *)p);

   p = &d;
   increase(p, DOUBLE);
   printf("%lf\n", *(double *)p);

   return 0;
}
```

```
void increase(void *number, enum DATA_TYPE d)
{
   switch (d)
   {
      case INT:
         *(int *)number += 1;
         break;
      case FLOAT:
         *(float *)number += 1.0;
         break;
      case DOUBLE:
         *(double *)number += 1.0;
   }
}
```

Α. Θεωρία

2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης

1. Η συνάρτηση malloc

- Υπενθύμιση: Έχουμε ήδη δει τη συνάρτηση malloc της οποίας το πρωτότυπο είναι:

```
void *malloc(size_t num)
```

  - Η οποία επιστρέφει ένα void * δείκτη.
  - Και πρέπει να κάνουμε μετατροπή στον δείκτη ώστε να μπορούμε να τον χρησιμοποιήσουμε.
  - και έχει οριστεί στο `stdlib.h`
  - Π.χ. δεσμεύουμε έναν ακέραιο με την εντολή:

```
int *ptr = (int *)malloc(sizeof(int));
```

    - όπου ενσωματώνουμε τη μετατροπή του void * δείκτη που επιστρέφει η malloc σε δείκτη ακεραίου.

Α. Θεωρία

2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης

2. Η συνάρτηση calloc

- Έχει οριστεί επίσης ακόμη μία συνάρτηση που δεσμεύει χώρο.

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

  - και έχει οριστεί στο `stdlib.h`
  - Η μόνη διαφορά είναι ότι δεσμεύει χώρο για num αντικείμενα μεγέθους size

- Έτσι για παράδειγμα μπορούμε να δεσμέυσουμε έναν πίνακα n ακεραίων με την εντολή:

```
int *ptr = (int *)calloc(n, sizeof(int));
```

- και φυσικά και ο τρόπος που ξέρουμε με τη malloc είναι ισοδύναμος:

```
int *ptr = (int *)malloc(n * sizeof(int));
```

Α. Θεωρία

2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης

3. Η συνάρτηση realloc

- Μία εξαιρετικά βοηθητική συνάρτηση είναι και η

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

  - και έχει οριστεί στο `stdlib.h`
- Η realloc επαναδεσμεύει χώρο που έχει ήδη δεσμεύσει η malloc (ή η calloc)
  - Παίρνει ως όρισμα χώρο που έχει δεσμεύτει δυναμικά, επεκτείνει το χώρο (ώστε να πιάνει μέγεθος size) και επιστρέφει το δείκτη.
  - Ο χώρος είτε θα επεκταθεί, είτε θα δεσμευτεί νέος (αν δεν μπορεί να επεκταθεί) με αντιγραφή των προηγούμενων δεδομένων.
  - Επιστρέφει NULL σε περίπτωση αποτυχίας.
- Συνηθισμένο σενάριο:
  - Δεσμεύουμε κάποιο χώρο με τη malloc.
  - Ο χώρος εξαντλείται
  - Επεκτείνουμε το χώρο με τη realloc.

Α. Θεωρία

2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης

3. Η συνάρτηση realloc

- Ανάλογα με τα ορίσματα μπορεί να έχει διαφορετικές χρήσεις:
  - Για να δεσμεύσει νέο χώρο (malloc):

```
p = realloc(NULL, size);
```

  - Για να αποδεσμεύσει χώρο (free):

```
realloc(p, 0);
```

Α. Θεωρία

2. Περισσότερα για τη Δυναμική Δέσμευση Μνήμης

4. Παράδειγμα

- Βλέπουμε και ένα παράδειγμα χρήσης της realloc:

```
/* realloc.c */
#include <stdio.h>
#include <stdlib.h>

main()
{
   int n;
   int i;
   int *p;

   /* Desmeusi gia 4 akeraious */
   n=4;
   p = (int *)malloc(n*sizeof(int));
   if (!p)
      printf("Error allocating memory!");

   /* Xrisi */
   for (i=0; i<n; i++)
      p[i]=i*i;
```

```
/* Epanadesmeusi gia akomi 4 */
   n=8;
   p = (int *)realloc(p, n*sizeof(int));
   if (!p)
      printf("Error allocating memory!");

   /* Xrisi */
   for (i=4; i<n; i++)
      p[i]=i*i;

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

   /* Apodesmeusi Mnimis */
   free(p);

}
```

Α. Θεωρία

3. Συναρτήσεις με μεταβλητό αριθμό ορισμάτων

1. Ορισμός Συνάρτησης

- Η C μας παρέχει μια προγραμματιστική δυνατότητα να έχουμε συναρτήσεις που έχουν μεταβλητό αριθμό ορισμάτων:
  - Μπορούμε να έχουμε μια συνάρτηση με όνομα sum
  - Η οποία θα μπορεί να προσθέτει οσαδήποτε ορίσματα και αν βάλουμε.
- Για να χρησιμοποιήσουμε αυτήν την προγραμματιστική ευκολία πρέπει να ενσωματώσουμε την βιβλιόθηκη: `stdarg.h`
- Η δήλωση της συνάρτησης:
  - πρέπει να έχει οπωσδήποτε ένα σταθερό όρισμα τουλάχιστον
  - Και έπειτα βάζουμε τρεις τελείες (…) για να υποδείξουμε ότι ακολουθεί μεταβλητό πλήθος ορισμάτων
- Παράδειγμα: Θα ορίσουμε μία συνάρτηση που θα αθροίζει οσαδήποτε ορίσματα
  - Το πρωτότυπό της θα είναι:

```
int sum(int n, …);
```

    - όπου n είναι το πλήθος των ορισμάτων που ακολουθούν
    - και οι τρεις τελείες υποδεικνύουν ότι ακολουθεί μεταβλητό πλήθος ορισμάτων.

Α. Θεωρία

3. Συναρτήσεις με μεταβλητό αριθμό ορισμάτων

2. Δομικά Στοιχεία

- Στο σώμα της συνάρτησης χρησιμοποιούμε τα εξής στοιχεία της βιβλιοθήκης stdarg.h:
  - Ορίζουμε έναν δείκτη τύπου va_list μέσω του οποίου θα γίνει η διαχείριση των ορισμάτων:

```
va_list ptr;
```

  - Αρχικοποιούμε με την μακροεντολή va_start() με ορίσματα το τελευταίο σταθερό όρισμα και το δείκτη διαχείρισης:

```
va_start(n, ptr);
```

  - Παίρνουμε τα ορίσματα με τη σειρά με τν μακροεντολή va_arg() με ορίσματα το δείκτη διαχείρισης και τον τύπο δεδομένων του τρέχοντος ορίσματος.

```
va_arg(ptr, int);
```

  - Όταν έχουμε ολοκληρώσει, θα πρέπει να γίνουν κάποιες ενέργειες που πραγματοποιούνται με την μακροεντολή va_end

```
va_end(ptr);
```

Θα μάθουμε να ορίζουμε δικές μας μακροεντολές σε επόμενο μάθημα.

Α. Θεωρία

3. Συναρτήσεις με μεταβλητό αριθμό ορισμάτων

3. Παράδειγμα

- Βλέπουμε και ολοκληρωμένο το παράδειγμα:

```
/* va_arg.c Metavlitos arithmos orismatwn*/

#include <stdio.h>
#include <stdarg.h>

int sum(int n, ...);

main()
{
   printf("%d\n", sum(4,1,2,3,4));
}
```

```
int sum(int n, ...)
{
   int i, s;
   va_list ptr;

   va_start(ptr,n);

   s=0;
   for (i=0; i<n; i++)
       s+=va_arg(ptr,int);

   va_end(ptr);

   return s;
}
```

Β. Ασκήσεις

1. Selection Sort γενικής χρήσης

Το παρακάτω τμήμα κώδικα (Αλγόριθμοι σε C – Μάθημα 3):

```
/* Taksinomisi Selection Sort */

for (i=0; i<N; i++)
{
   pos=i;

   for (j=i+1; j<N; j++)
   {
      if (pinakas[j]<pinakas[pos])
         pos=j;
   }

   swap(&pinakas[i], &pinakas[pos]);
}
```

Υλοποιεί τον αλγόριθμο selection sort.

- Δημιουργήστε μια συνάρτηση selection sort «γενικής χρήσης» που να δουλεύει σωστά τόσο για ακέραιους, όσο και για πραγματικούς (double)

Β. Ασκήσεις
2. Παιχνίδι με τη μνήμη

Υλοποιήστε ένα πρόγραμμα το οποίο θα «παίζει με τη μνημη» και συγκεκριμένα:

1. Μία συνάρτηση double_space η οποία θα δέχεται κατάλληλα ως όρισμα έναν πίνακα ακεραίων και το πλήθος των θέσεών του. Θα διπλασιάζει τις θέσεις του πίνακα με κατάλληλη χρήση της realloc.
2. Μία συνάρτηση half_space η οποία θα δέχεται κατάλληλα ως όρισμα έναν πίνακα ακεραίων και το πλήθος των θέσεών του. Θα μειώνει τις θέσεις του πίνακα στο μισό, με κατάλληλη χρήση της realloc.
3. Μία συνάρτηση fill με τέσσερα ορίσματα (πίνακας ακεραίων, πλήθος θέσεων, αρχή, τέλος) που θα γεμίζει τον πίνακα με τυχαίους αριθμούς από το 0 έως το 99, στις θέσεις «αρχή» έως «τέλος»

Η main μέσω ενός μενού θα δίνει τις επιλογές:

- Διπλασιασμού των θέσεων (με κατάλληλο γέμισμα με τιμές των νέων θέσεων)
- Μείωσης των θέσεων στο μισό
- Εκτύπωσης του πίνακα
- Έξοδου από το πρόγραμμα

Β. Ασκήσεις
3. Συνάρτηση μέσος όρος

Κατασκευάστε μία συνάρτηση «average» η οποία θα δέχεται ως ορίσματα:

- Το πλήθος των αριθμών (1ο όρισμα)
- Ακολουθούν οσαδήποτε ορίσματα ακεραίων

Η συνάρτηση θα υπολογίζει και θα επιστρέφει το μέσο όρο αυτών των αριθμών.