Διάλεξη 6:
Διαχείριση Μνήμης & Δυναμικές Δομές Δεδομένων

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Δυναμικές Δομές Δεδομένων Γενικά

Δυναμική Δέσμευση/Αποδέσμευση Μνήμης

Δομή τύπου structure – Αυτοαναφορικές δομές

Η δήλωση typedef στη C

Δυναμικές Δομές Δεδομένων

Στατική Δομή Δεδομένων

Η απαιτούμενη μνήμη για την εκτέλεση του προγράμματος δεσμεύεται κατά την αρχικοποίηση του προγράμματος.

π.χ. struct person { char name[30], int age} costas, marios;
// πριν την έναρξη του προγράμματος δεσμεύονται 34Bytes * 2

Δυναμική Δομή Δεδομένων

- Ενα πρόγραμμα κατά την εκτέλεσή του θα μπορεί να ζητά δυναμικά μνήμη για την αποθήκευση δεδομένων καθώς και να ελευθερώνει δυναμικά τη μνήμη που δεν χρειάζεται.

- Συνεπώς το μέγιστο πλήθος στοιχείων που μπορούμε να αποθηκεύσουμε δεν χρειάζεται να είναι γνωστό εκ των πρότερων αλλά μπορεί να ορίζετε κατά την διάρκεια εκτέλεσης του προγράμματος.

Δυναμικές Δομές Δεδομένων

- Θα μάθουμε πως μπορούμε να ζητήσουμε κατά την εκτέλεση του προγράμματος μνήμη από το λειτουργικό σύστημα και να την επιστρέψουμε όταν μας είναι πλέον άχρηστη.

- Όταν το λειτουργικό σύστημα **δεν είναι σε θέση** να μας δώσει άλλη μνήμη τότε και μόνο τότε το πρόγραμμά μας θα τυπώνει **μηνύματα αδυναμίας αποθήκευσης** πληροφοριών, κάτι που συμβαίνει όμως σπάνια και συνήθως όταν κάνουμε αλόγιστη χρήση της μνήμης.

- Τα σύγχρονα λειτουργικά συστήματα κάνουν χρήση εικονικής μνήμης (**virtual memory**). Δηλαδή χρησιμοποιείται μέρος της δευτερεύουσας μνήμης (**hard disk**) σαν λογική συνέχεια της κύριας μνήμης

- Ουσιαστικά η κύρια μνήμη εξαντλείτε μόνο αν εξαντληθεί η δευτερεύουσα μνήμη.

Δυναμική Δέσμευση Μνήμης

- `void *malloc(num_of_bytes)`: η malloc() δεσμευει δυναμικά ένα συνεχόμενο block μνήμης (μεγέθους `num_of_bytes`) και επιστρέφει ένα δείκτη στο χώρο μνήμης που δέσμευσε.

- Η malloc() επιστρέφει NULL όταν η αίτηση δεν μπορεί να ικανοποιηθεί (δηλαδή δεν υπάρχει άλλη διαθέσιμη μνήμη για να δοθεί).
- Για παράδειγμα η malloc(sizeof(int)) δεσμεύει μνήμη για την αποθήκευση ενός ακεραίου και επιστρέφει ένα δείκτη (τη διεύθυνση δηλαδή) του χώρου μνήμης που δέσμευσε.
- Γιατί sizeof(int) και όχι απλά 4 => portability (αν κάποια πλατφόρμα χρησιμοποιεί 2 Bytes για αναπαράσταση ακεραίων τότε το πρόγραμμα μας εξακολουθεί να είναι σωστό!)

- Στη C μιλάμε πάντα για *δείκτες ενός συγκεκριμένου τύπου* πρέπει πάντα να κάνουμε cast τον δείκτη που επιστρέφει η malloc() στον αντίστοιχο τύπο. Δηλαδή

```
(int *)  malloc(sizeof(int))
```

Δυναμική Δέσμευση Μνήμης (συν)

- Έτσι κάνοντας τις παρακάτω δηλώσεις και κλήσεις:

```
int *nump; // δήλωση δείκτη (χωρίς δέσμευση μνήμης)

// δέσμευση μνήμης για αναπαράσταση ενός ακέραιου.
nump = (int *) malloc(sizeof(int));

// αρχικοποίηση περιεχομένου
*nump = 17;

// αποδέσμευση μνήμης
free(nump);
```

- Σημαντική είναι η χρήση της **`sizeof(type)`** η οποία μας επιστρέφει το μέγεθος αντικειμένου τύπου **`type`** και μας απελευθερώνει από τη δυσκολία εύρεσης του ποσού μνήμης που απαιτείται για την αποθήκευση ενός αντικειμένου τύπου **`type`**.

Αυτοαναφορικές δομές

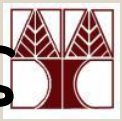
```
struct Employee {
    char                name[20];
    int                 age;
    struct Employee manager;
};
```

- Στην πιο πάνω δομή θα δοθεί "compile error", γιατί το struct Employee χρησιμοποιείται κατά την διάρκεια της δήλωσης του.
- Εντούτις μπορούμε να ορίσουμε δομές που αναφέρονται στον εαυτό τους, όπως στην προκειμένη περίπτωση δομή Employee όπου για κάθε στοιχείο της έχουμε πεδίο που αναφέρεται στον διευθυντή του εργοδοτημένου, χρησιμοποιώντας δείκτες:

```
struct Employee {
    char                name[20];
    int                 age;
    struct Employee *manager;
};
```

printf("%d", sizeof(struct Employee)); ➔ επιστρέφει 28

Δομή structure – Αυτοαναφορικές δομές



- Όταν χρησιμοποιούμε δυναμική δέσμευση μνήμης συνήθως το κάνουμε για την αποθήκευση όχι τόσο απλών τύπων δεδομένων (int, float, char κλπ.) **αλλά αντικειμένων τύπου structure.**

- Αυτό γιατί μπορούμε μέσω αντικειμένων τύπου structure να φτιάξουμε κόμβους, να τους συνδέσουμε μεταξύ τους και να δημιουργήσουμε έτσι μία συνδεδεμένη λίστα ή άλλες **εξελιγμένες δομές όπως στοίβες, ουρές, λίστες αναμονής, δέντρα κλπ.**

- Ένας απλός ορισμός κόμβου μιας **συνδεδεμένης λίστας** είναι ο εξής:

```
struct node {
        int data;
        struct node *next;
};
```

Προσέξτε ότι ένα πεδίο της δομής που υλοποιεί τον κόμβο είναι δείκτης στην ίδια δομή που ορίζεται. Αυτό το φαινόμενο όπως είπαμε ονομάζεται **αυτοαναφορική δομή (self-referential structures).**

Αυτοαναφορικές δομές (συν.)

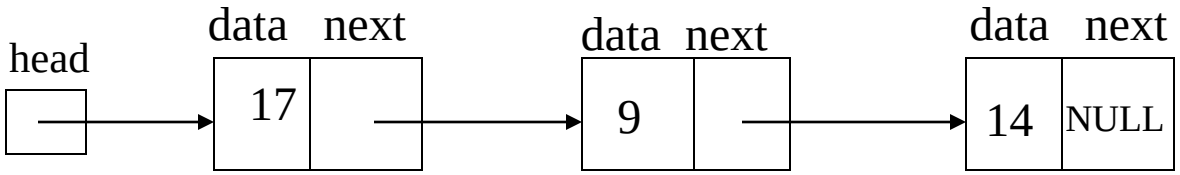
- Παρόμοια μπορούμε να ορίσουμε παραλλαγή αυτοαναφορικών δομών όπου δύο δομές αναφέρονται η μια στην άλλη.
- Αυτό αντιμετωπίζεται ως εξής:

```
struct s1 {
    . . .
    struct s2 *p;
    . . .
};

struct s2 {
    . . .
    struct s1 *q;
    . . .
};
```

Δομή τύπου structure (συν.)

- Έχοντας λοιπόν καθορίσει τη μορφή ενός κόμβου μπορούμε να φανταστούμε πως θα είναι μία συνδεδεμένη λίστα με κόμβους τύπου struct node που ορίσαμε νωρίτερα:



Δομή τύπου structure (συν.)

- Μια λίστα 10 στοιχείων θα μπορούσε να υλοποιηθεί ως εξής:

```
struct node *head, *temp;
int i;

head  = (struct node *) malloc (sizeof(struct node));
temp = head;
for(i=0; i<10; i++) {
      temp->data = i;
      if (i==9) break;
      temp->next = (struct node *) malloc(sizeof(struct node));
      temp = temp->next;
}
temp->next = NULL;
temp = head;

for(i=0; i<10; i++) {
      printf("%d %d\n", temp->data, temp->next);
      temp = temp->next;
}
```

```
struct node {
   int data;
   struct node *next;
};
```

- *Σημείωση: ο δείκτης temp είναι βοηθητικός για την υλοποίηση της λίστας.*

Η δήλωση typedef στη C

- Η C παρέχει μέσω της δήλωσης **typedef** τη δημιουργία νέων ονομάτων σε τύπους δεδομένων που ήδη υπάρχουν.
- Προσοχή: δεν δημιουργούμε νέους τύπους δεδομένων. Απλά **"βαφτίζουμε"** με νέα ονόματα τύπους που ήδη υπάρχουν. Μερικά παραδείγματα είναι τα εξής:

```
typedef int  Akeraios;
typedef struct node NODE;      /* το struct node
                                  έχει οριστεί νωρίτερα */
```

- Έτσι ορισμοί μεταβλητών μπορούν να υπάρξουν τώρα ως εξής:

```
Akeraios   len, I, arr[20];
NODE       head, tail, *temp;
```

Η δήλωση typedef στη C (συν.)

- Πολλές φορές **βαφτίζουμε** μία δομή ταυτόχρονα με τον **ορισμό** της, όπως στο παράδειγμα:

```
typedef struct node {
        int data;
        struct node *next;
  } NODE;
```

- Είναι καλή συνήθεια να χρησιμοποιούμε ως όνομα στο typedef το ίδιο όνομα της δομής αλλά με κεφαλαία γράμματα.

- Γιατί να χρησιμοποιούμε τη δήλωση `typedef` ;
  1. Γιατί δημιουργεί πιο αναγνώσιμο και πιο κατανοητό κώδικα . Το να δηλώσεις `NODE *ptr;` είναι πιο κατανοητό από το να δηλώσεις ένα δείκτη σε μία πολύπλοκη δομή (`struct node *ptr;`).
  2. Ο κώδικας γίνεται πιο λιτός. Π.χ.

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

Δομή τύπου structure (συν.)

- Τώρα μια λίστα 10 στοιχείων θα μπορούσε να υλοποιηθεί και ως εξής:

```
NODE *head, *temp;
int i;

head  = (NODE *) malloc (sizeof(NODE));
temp = head;
for(i=0; i<10; i++) {
      temp->data = i;
      if (i==9) break;
      temp->next = (NODE *) malloc(sizeof(NODE));
      temp = temp->next;
}
temp->next = NULL;
temp = head;

for(i=0; i<10; i++) {
      printf("%d %d\n", temp->data, temp->next);
      temp = temp->next;
}
```

```
typedef struct node {
   int data;
   struct node *next;
} NODE ;
```

- *Σημείωση: ο δείκτης temp είναι βοηθητικός για την υλοποίηση της λίστας.*