Διάλεξη 7: Διαχείριση Μνήμης, Δυναμικές Δομές Δεδομένων, Αναδρομή

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:
Εισαγωγή στις έννοιες:

- Δυναμικές Δομές Δεδομένων Γενικά
- Δυναμική Δέσμευση/Αποδέσμευση Μνήμης
- Αυτοαναφορικές δομές
- Αναδρομή

Διδάσκων: Παναγιώτης Ανδρέου

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 (συν.)

- 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:

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


// 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`**.

Αυτο-αναφορικές δομές

- Στην δήλωση της δομής δεξιά θα δοθεί "compile error", γιατί το struct Employee χρησιμοποιείται κατά την διάρκεια της δήλωσης του.
- Επίσης, αν επιτρεπόταν, το φώλιασμα της συγκεκριμένης δομής θα επεκτεινόταν μέχρι το άπειρο.
- Εντούτοις 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 όπως φαίνεται δεξιά.

Παράδειγμα Δομής Employee

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

Παράδειγμα Δομής Employee

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

Αυτο-αναφορικές δομές (συν.)

- 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:

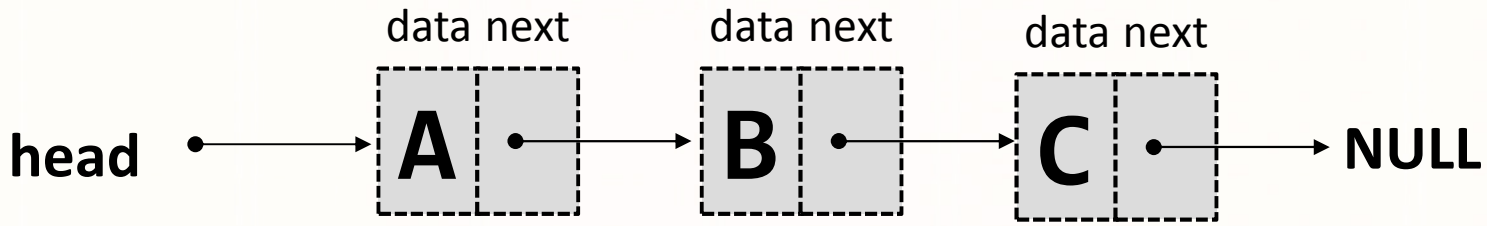
```
struct node{
   char         data;
   struct node * next;
} node;
```

```
typedef struct node{
   char data;
   node * next;
} node;
```

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).**

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:



- Υλοποίηση Παραδείγματος

Στατική Δέσμευση Μνήμης

```
struct node head, n1, n2, n3;
n1.data = 'A';
n2.data = 'B';
n3.data = 'C';
head    = &n1;
n1.next = &n2;
n2.next = &n3;
n3.next = NULL;
```

Δυναμική Δέσμευση Μνήμης

```
  struct node *head, *n1, *n2,
*n3;
  n1 = malloc(sizeof( node ));
  n2 = malloc(sizeof( node ));
  n3 = malloc(sizeof( node ));
  head    = &n1;
  //Τα υπόλοιπα όπως αριστερά
από n1.data = 'A';
```

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{
   char data;
   node * next;
} node;
```

- 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 ) );
```

Αναδρομή

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Η έννοια της αναδρομής
- Μη-αναδρομικός / Αναδρομικός Ορισμός Συναρτήσεων
- Παραδείγματα Ανάδρομης: Παραγοντικό, Δύναμη, Αριθμοί Fibonacci
- Αφαίρεση της Αναδρομής

Διδάσκων: Παναγιώτης Ανδρέου

Μη αναδρομικές συναρτήσεις

- Προτού δούμε τι είναι αναδρομή θεωρήστε το πρόβλημα εύρεσης του παραγοντικού κάποιου αριθμού (factorial).

  0! = 1 , **1!** = 1 **2!** = 1x2 = 2 **3!** = 1x2x3 = 6

  4! = 1x2x3x4=24 **5!** = 1x2x3x4x5 = 120

  Ζητούμενο: *Να υλοποιήσουμε την factorial(int n) η οποία μας επιστρέφει το παραγοντικό κάποιου θετικού ακεραίου n.*

  Λύση

```
int factorial(int n) {
        int i, result=1;
        for (i=1; i<=n; i++)  {
                result *= i;
        }
        return result;
}
```

Αναδρομή

- Βασική έννοια στα Μαθηματικά και στην Πληροφορική.

- Στην πληροφορική η αναδρομή χρησιμοποιείται σαν *τεχνική προγραμματισμού* και σαν *μέθοδος σχεδιασμού αλγορίθμων*.

- Στον προγραμματισμό η αναδρομή εμφανίζεται με την **κλήση ενός υποπρογράμματος από τον εαυτό του**. Ένα αναδρομικό υποπρόγραμμα αποτελείται από:
  - ένα **βήμα τερματισμού**, όπου ορίζεται η εκτέλεση του υποπρογράμματος για κάποιες "μικρές "τιμές των παραμέτρων του, και
  - ένα **αναδρομικό βήμα**, κατά το οποίο η εκτέλεση του υποπρογράμματος ορίζεται ως συνδυασμός κλήσεων του υποπρογράμματος σε άλλες "μικρότερες" τιμές των παραμέτρων.

Παράδειγμα 1: Παραγοντικός Αριθμός με Αναδρομή

- Ας ορίσουμε τώρα τον αναδρομικό ορισμό της factorial.

  0! = 1 , **1!** = 1 **2!** = 1x2 = 2 **3!** = 1x2x3 = 6

  4! = 1x2x3x4=24 **5!** = 1x2x3x4x5 = 120

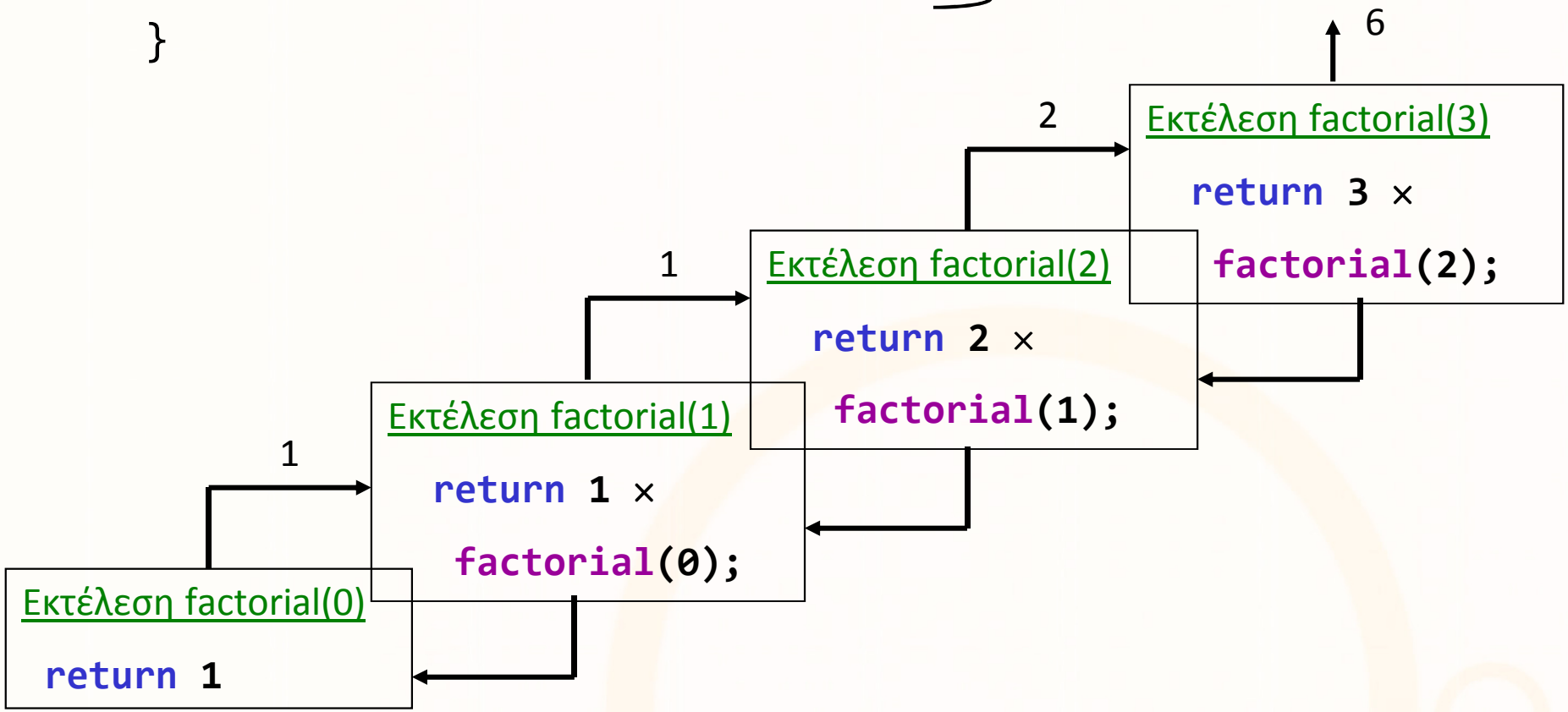
- Ας ορίσουμε τώρα τον αναδρομικό ορισμό της factorial.
- **Αναδρομικός Ορισμός συνάρτησης <factorial>**
  - Βήμα Τερματισμού: **0! = 1**
  - Αναδρομικό Βήμα: **n! = n x (n-1)!**

Παραγοντικός Αριθμός με Αναδρομή

```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n x factorial(n-1);
        }
```

Βήμα Τερματισμού

Αναδρομικό Βήμα



Υλοποίηση αναδρομής

- Σε κάθε κλήση οποιασδήποτε συνάρτησης ένα σύνολο από λέξεις **(stack frame) φυλάσσεται σε μια στοίβα (την στοίβα του προγράμματος)**, από όπου μπορεί να ανασυρθεί.

- Όταν μια συνάρτηση διακόψει την εκτέλεσή της με την κλήση μιας άλλης συνάρτησης ***οι παράμετροι της συνάρτησης***, *η* ***διεύθυνση επιστροφής*** και *οι* ***τοπικές μεταβλητές*** της **καλούσας συνάρτησης** φυλάσσονται μέσα στη στοίβα του προγράμματος.

- Έτσι όταν η κληθείσα συνάρτηση τερματίσει το περιβάλλον την καλούσας συνάρτησης **ανασύρεται από τη στοίβα** για να συνεχιστεί κανονικά η εκτέλεσή της.

- Αφού κάθε κλήση μιας διαδικασίας εκτελείται στο δικό της περιβάλλον, είναι επιτρεπτή και **η κλήση συναρτήσεων από τον εαυτό** τους (αναδρομή).

Παράδειγμα 2: Δύναμη Αριθμού με Αναδρομή

2) Δύναμη (Power)

$a^0 = 1$

$a^n = a^{n-1}.a \qquad (n>=1)$

```
int mpower(int a, int n) {
        if (n==0) return 1;
        return a*mpower(a, n-1);
}
```

Παράδειγμα

mpower(2,3) => 2*mpower(2,2)

= 2*2*mpower(2,1)

= 2*2*2*mpower(2,0)

= 2*2*2*1= 8

Παράδειγμα 3: Fibonacci Numbers

3) Αριθμοί Fibonacci (Leonardo of Pisa – 1202μΧ)

Χρησιμοποιήθηκαν για να εκφράσουν την αύξηση κουνελιών!

- Στον μήνα 0 έχουμε 0 ζεύγη , και στον μήνα 1 έχουμε 1 ζεύγος (η Γένεσης!)
- Το ζεύγος γονιμοποιείται μετά τον πρώτο μήνα (δηλ. στον δεύτερο).
- Κάθε μήνα, Κάθε ζεύγος παράγει ένα νέο ζεύγος
- Έστω ότι είμαστε στον μήνα **n** και έχουμε ένα πληθυσμό **F(n) ζευγών**. Αυτή την στιγμή μόνο κουνέλια που ήταν ζωντανά την στιγμή **n-2** παράγουν ένα νέο ζεύγος.
- Επομένως **F(n-2)** ζευγάρια προστίθεται στον παρόν πληθυσμό των F(n-1).
- Ο ολικός πληθυσμός την στιγμή F(n) είναι επομένως **F(n) = F(n-1) + F(n-2)**

0,1,1,2,3,5,8,13,21,34,55,89,….

$$F_n := F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

```
int fibonacci(int a) {
  if (a==0) return 0;
  else if (a==1) return 1;
  return fibonacci(a-1) +
                 fibonacci(a-2);
}
```

Αφαίρεση της Αναδρομής

- Η χρήση της αναδρομής επιτρέπει την επίλυση πολύπλοκων προβλημάτων με **άμεσο και σαφή τρόπο**. Συχνά όμως **υστερεί από άποψη αποδοτικότητας.**

- Η **αφαίρεση της αναδρομής από μια συνάρτηση**, δηλαδή, η μετατροπή της σε επαναληπτική συνάρτηση χωρίς αναδρομή, είναι δυνατή (κάτω από κάποιες συνθήκες).

- Συχνά προϋποθέτει τη χρήση κάποιων βοηθητικών δομών (π.χ. στοίβα ή ουρά – θα μελετηθούν στην συνέχεια του μαθήματος).

- Επίσης στις περισσότερες περιπτώσεις μπορούμε να επιλύσουμε μια αναδρομική εξίσωση και να βρούμε την λύση της σε κλειστή μορφή (με την χρήση Αναδρομικών Σχέσεων – Recurrence Relations).

  Π.χ. η λύση της εξίσωσης Fibonacci είναι :

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$