ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

12η Εβδομάδα: Δυαδικά Δένδρα,

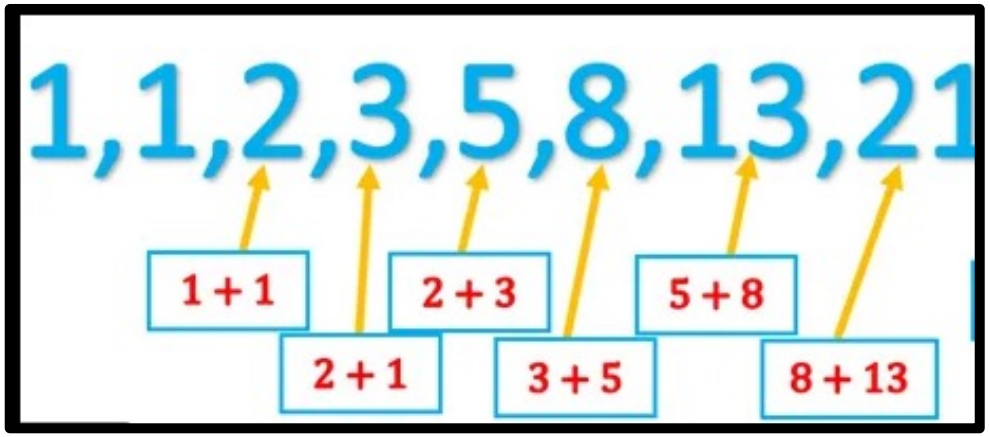
Αναφορές

Οι διαφάνειες της διάλεξης στηρίζονται, εν μέρει, σε υλικό παραδόσεων παλαιότερων ετών του **Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογία Υπολογιστών του Πανεπιστημίου Πατρών** καθώς και του **Τμήματος Πληροφορικής του Πανεπιστήμιο Κύπρου.**

Από την προηγούμενη διάλεξη: Αναδρομή και Ακολουθία Fibonacci

Αριθμοί Fibonacci (Leonardo of Pisa - 1202μΧ)

Χρησιμοποιήθηκαν για να εκφράσουν την αύξηση κουνελιών!



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

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

Ακολουθία Fibonacci Μεγάλων Αριθμών

Πως μπορώ να υπολογίσω μιας ακολουθία **FIBONACCI N=100**

32-bit: 2.147.483.647

Υπάρχουν Βιβλιοθήκες για Υπολογιστική Αριθμητική Αυθαίρετης Ακρίβειας π.χ. GMP Library

Όποιος ενδιαφέρεται να ασχοληθεί με εξωτερικές βιβλιοθήκες να επικοινωνήσει μαζί μου για την αποστολή εκπαιδευτικού υλικού! Δεν αφορά την εξεταστέα ύλη του μαθήματος.

```
Fibonacci Series (n=100):0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,25
84,4181,6765,10946,17711,28657,46368,75025,121393,196418,317811,514229,832040,134
6269,2178309,3524578,5702887,9227465,14930352,24157817,39088169,63245986,10233415
5,165580141,267914296,433494437,701408733,1134903170,1836311903,-1323752223,5
9680,-811192543,-298632863,-1109825406,-1408458269,1776683621,3
3,-1781832971,363076002,-1418756969,-1055680967,1820529360,764848393,-1709589543,
-944741150,1640636603,695895453,-1958435240,-1262539787,1073992269,-188547518,885
444751,696897233,1582341984,-2015728079,-433386095,1845853122,1412467027,-1036647
147,375819880,-660827267,-285007387,-945834654,-1230842041,2118290601,887448560,-
1289228135,-401779575,-1691007710,-2092787285,511172301,-1581614984,-1070442683,1
642909629,572466946,-2079590721,-1507123775,708252800,-798870975,-90618175,-88948
9150,-980107325


Fibonacci Series (n=100):0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,25
84,4181,6765,10946,17711,28657,46368,75025,121393,196418,317811,514229,832040,134
6269,2178309,3524578,5702887,9227465,14930352,24157817,39088169,63245986,10233415
5,165580141,267914296,433494437,701408733,1134903170,1836311903,2971215073,48
6976,7778742049,12586269025,20365011074,32951280099,53316291173,
3862445,225851433717,365435296162,591286729879,956722026041,1548008755920,2504730
781961,4052739537881,6557470319842,10610209857723,17167680177565,27777890035288,4
4945570212853,72723460248141,117669030460994,190392490709135,308061521170129,4984
54011879264,806515533049393,1304969544928657,2111485077978050,3416454622906707,55
27939700884757,8944394323791464,14472334024676221,23416728348467685,3788906237314
3906,61305790721611591,99194853094755497,160500643816367088,259695496911122585,42
0196140727489673,679891637638612258,1100087778366101931,1779979416004714189,28800
67194370816120,4660046610375530309,7540113804746346429,12200160415121876738,19740
274219868223167,31940434634990099905,51680708854858323072,83621143489848422977,13
530185234470674604 9,218922995834555169026,354224848179261915075
```

Ακόμη ένα παράδειγμα – Πρώτοι Αριθμοί – Συμβατικές Βιβλιοθήκες και GMP

Προσπαθήστε να γράψετε πρόγραμμα το οποίο να βρίσκει τους πρώτους αριθμούς στο διάστημα [MAX_INT-2, MAX_INT+13].

[32-bit:2147483645 32-bit: 2147483660]

```
int start;
int end;
int k;
int c;
start=1;
end=100;
start=INT_MAX-10;
 end=INT_MAX;
 for (k = start; k <= end; k++) {
   if (k == 0 || k == 1) continue;

   for (c = 2; c <= k ; c++){
        if (k%c == 0) break;
    }
    if (c == k){
        printf("%d\n", k);
     }
}
```

```
2147483647
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
```

...

k: Integer overflow

```
2147483647
k=-2147483648
k=-2147483647
k=-2147483646
k=-2147483645
```

...

```
2147483645      not prime
2147483646      not prime
2147483647      PRIME!
2147483648      not prime
2147483649      not prime
2147483650      not prime
2147483651      not prime
2147483652      not prime
2147483653      not prime
2147483654      not prime
2147483655      not prime
2147483656      not prime
2147483657      not prime
2147483658      not prime
2147483659      PRIME!
2147483660      not prime
```

GMP

1° Παράδειγμα με τη GMP

Όποιος ενδιαφέρεται να ασχοληθεί με εξωτερικές βιβλιοθήκες να επικοινωνήσει μαζί μου για την αποστολή εκπαιδευτικού υλικού! Δεν αφορά την εξεταστέα ύλη του μαθήματος.

Πρόγραμμα το οποίο σας επιτρέπει να προσδιορίζεται αυθαίρετα τον αριθμό σε bits που θέλετε να δεσμεύεστε για την αποθήκευση ενός ακεραίου αριθμού και να εκτυπώνει τον μέγιστο αριθμό

```c
#include <stdio.h>
#include "gmp.h"
int main()
{
    int bits_for_int;
    printf("Enter an arbitrary number of bits
    scanf("%d",&bits_for_int);
    bits_for_int=bits_for_int-1;

    mpz_t res;
    mpz_init_set_ui(res,1);

    mpz_mul_2exp(res, res , bits_for_int);
    mpz_sub_ui (res, res, 1);

    mpz_out_str(stdout,10,res);
    printf("\n");
    mpz_clear(res);
    return 0;
}
```

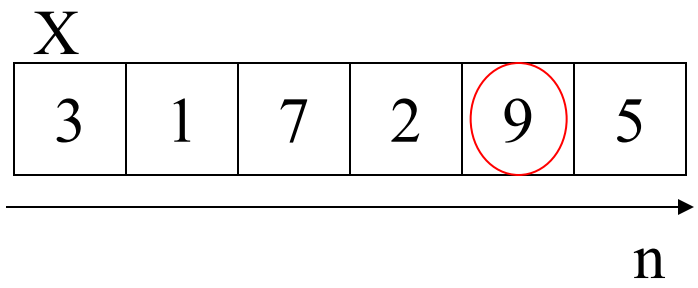
```
Enter an arbitrary number of bits for a signed integer: 32
2147483647
```

```
Enter an arbitrary number of bits for a signed integer: 1024
89884656743115795386465259539451236680898848947115328636715040578866337902750481
56635423866120376801056005693993569667882939488440720831124642371531973706218888
39467124327426381511098006230470597265414760425028844190753411712314407369565552
70413618581675255342293149119973622969239858152417678164812112068607
```

```
Enter an arbitrary number of bits for a signed integer: 2054
10341441942819522336228760540374384627342112854308954890281710536807889644437725
08582310445168733230838038974749500736623811825892815709475042820876366345059860
18807575363187493600632801246886741747384442829016218485735099656383219717881901
88312374550586126672651617613724024285293141058754416767719657723367012586684693
86957461919193170066223835397633130442305858489781098592838166670730052808274971
65366773523019531899077943544314723687365266894091832362346519139926802191074624
67112050272552618023510297133625810143339214719125872959564228968053244619947373
95268333863347348650191315358468742193787315553907079380991
```

Αλγόριθμοι Αναζήτησης - Πολυπλοκότητα

Υποθέστε ότι θέλετε να βρείτε το μεγαλύτερο στοιχείο σε έναν πίνακα θετικών ακεραίων.




```
int largest( int X[], int n){
        int current=0, i=0;
        while ( i < n ){
            if ( X[i] > current){
               current = X[i];     }
            i = i+1;
         }
     return current;
}
```

Βασική πράξη θεωρούμε ότι είναι οποιαδήποτε πράξη της οποίας ο χρόνος εκτέλεσης είναι φραγμένος από κάποια σταθερά

***Μέγεθος δεδομένων εισόδου*: n**

Στόχος: υπολογισμός του αριθμού βασικών πράξεων

Χειρότερη Περίπτωση: εξέταση όλων των στοιχείων $t(n) = n$

Βέλτιστη Περίπτωση: εξέταση όλων των στοιχείων $t(n) = n$

Αλγόριθμοι Αναζήτησης - Πολυπλοκότητα

Αν ήταν ταξινομημένη η λίστα (σε αύξουσα σειρά) τότε το μεγαλύτερο στοιχείο μπορεί να βρεθεί σε σταθερό χρόνο (δηλαδή πάντα το τελευταίο στοιχείο).

```
int largest2( int X[], int n){
      if (n<0)
            return -1;
      return X[n-1];
}
```

Μέγεθος δεδομένων εισόδου: n

Χειρότερη Περίπτωση: εξέταση τελευταίου στοιχείου $t(n) = 1$

Βέλτιστη Περίπτωση: εξέταση τελευταίου στοιχείου $t(n) = 1$

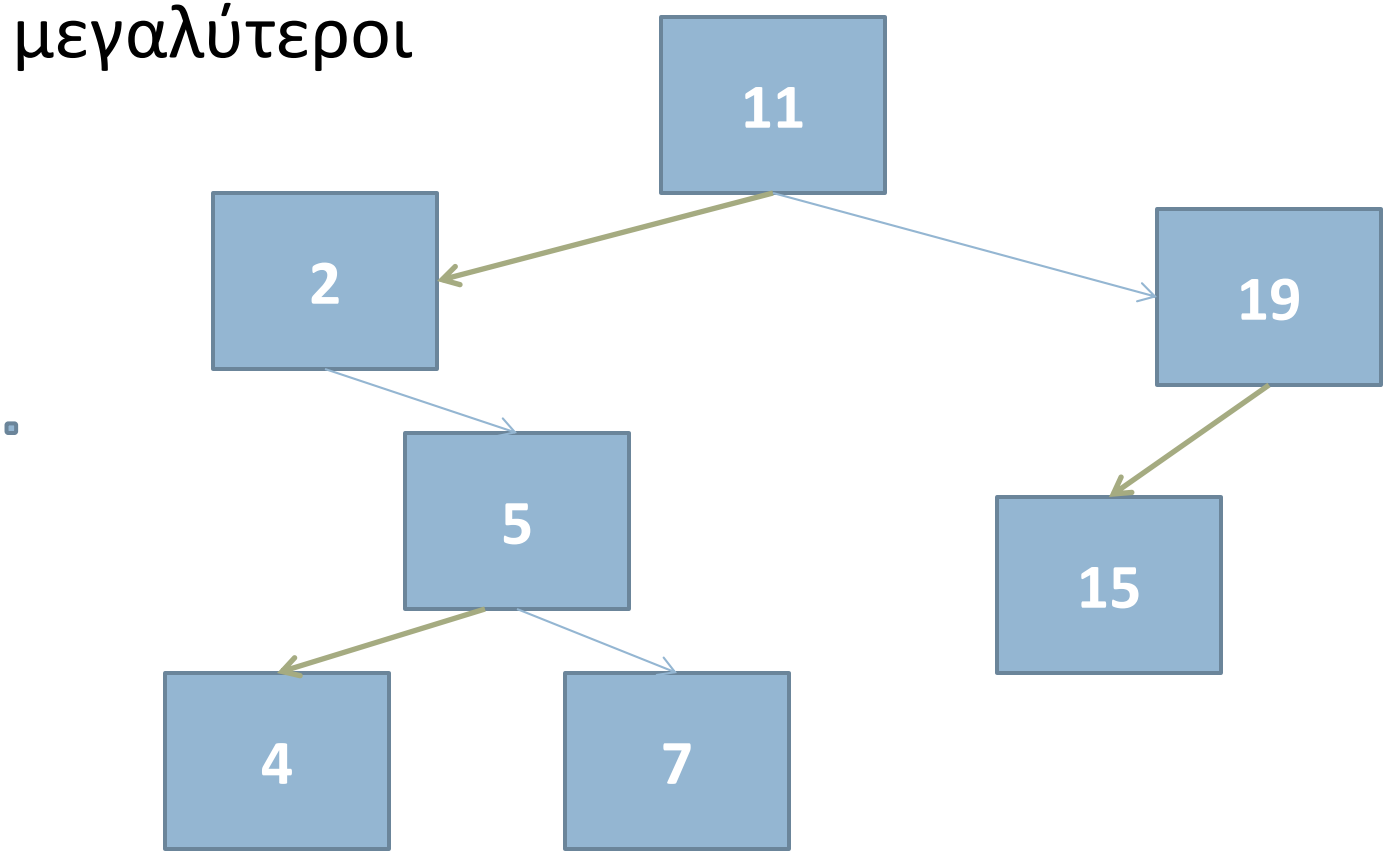
Είδη αλγορίθμων

t(n)	Όνομα
1 (ένα)	Σταθερός (constant)
n	Γραμμικός (linear)
n^k	Πολυωνυμικός (polynomial)
n^2	Τετραγωνικός – δευτεροβάθμιος (quadratic), π.χ., Shortest Path Dijkstra
n^3	Κυβικός - τριτοβάθμιος (cubic), π.χ., All-Pair-Shortest Path Floyd-Warshall
c^n , n^n $(2^n,3^n)$	Εκθετικός (exponential)
$\log_k n$	Λογαριθμικός (logarithmic)

Δυαδικά Δένδρα

- **Το πιο σημαντικό πλεονέκτημα της χρήσης δυαδικών δένδρων η αποδοτική αναζήτηση σε ένα σύνολο στοιχείων.**

- Παράδειγμα: 11, 2, 5, 19, 7, 15, 4
- Κανόνας: Αριστερά οι μικρότεροι, δεξιά οι μεγαλύτεροι



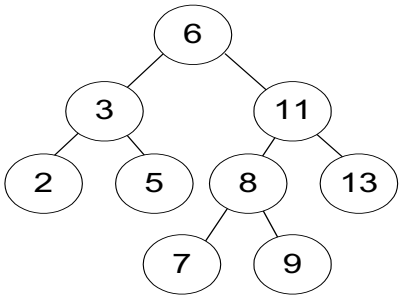
Δυαδικά Δένδρα

- Ένα δένδρο είναι **δυαδικό** αν όλοι οι κόμβοι του έχουν **βαθμό** ≤ 2.

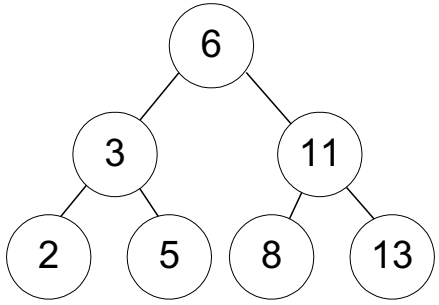
Ορισμός: Δυαδικό δένδρο λέγεται ένα δένδρο το οποίο :

- είτε είναι κενό,
- ή αποτελείται από μια **ρίζα** και **δύο δυαδικά υπόδενδρα**. Αναφερόμαστε στα δύο υπόδενδρα ως το **αριστερό** και το **δεξιό υπόδενδρο**.

- Το ύψος ενός δυαδικού δένδρου με **n** κόμβους μπορεί να είναι το πολύ : **n-1 (συνδεδεμένη λίστα)** και το λιγότερο **$\log_2(n)$.**
- Ένα δυαδικό δένδρο είναι **γεμάτο (full),** αν κάθε εσωτερικός του κόμβος έχει δύο απογόνους.
- Ένα δυαδικό δένδρο είναι **τέλειο (perfect),** αν είναι γεμάτο και όλοι του οι κόμβοι που βρίσκονται στο ίδιο επίπεδο έχουν το ίδιο βάθος.

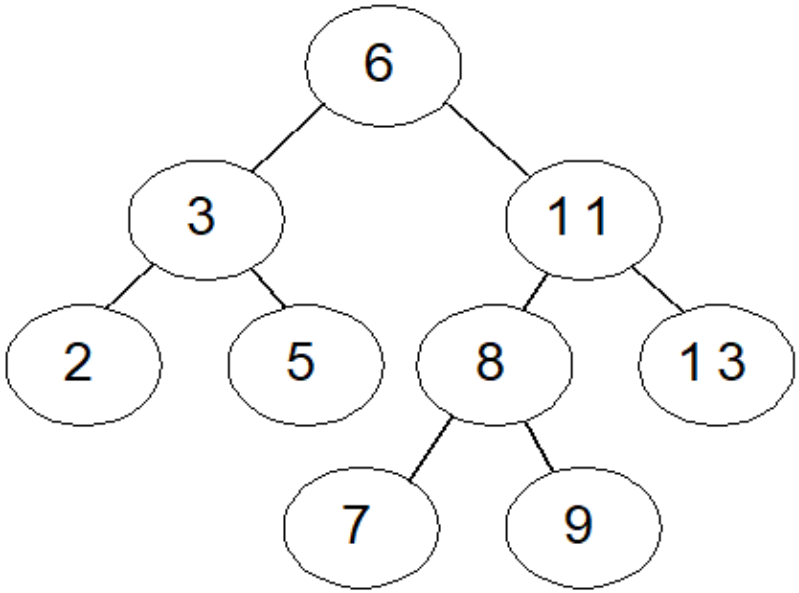


FULL

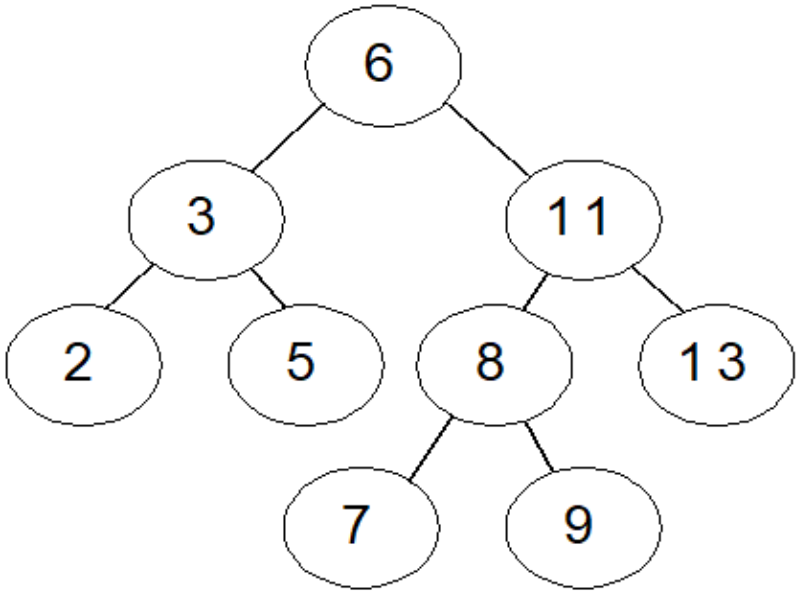


PERFECT

Δυαδικά δένδρα

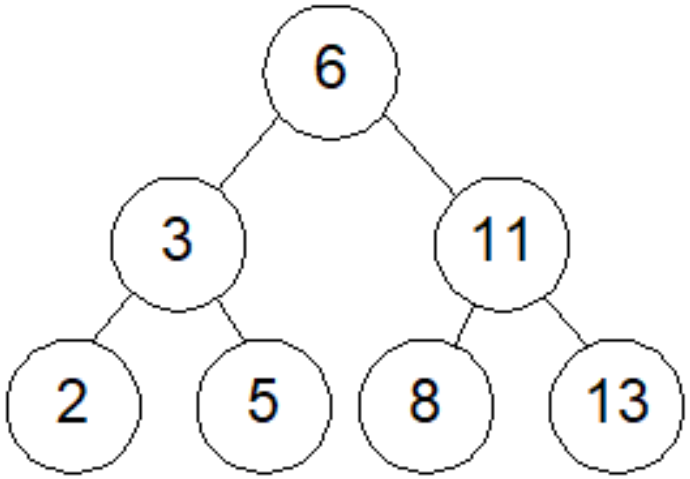


Δυαδικά δένδρα παραδείγματα



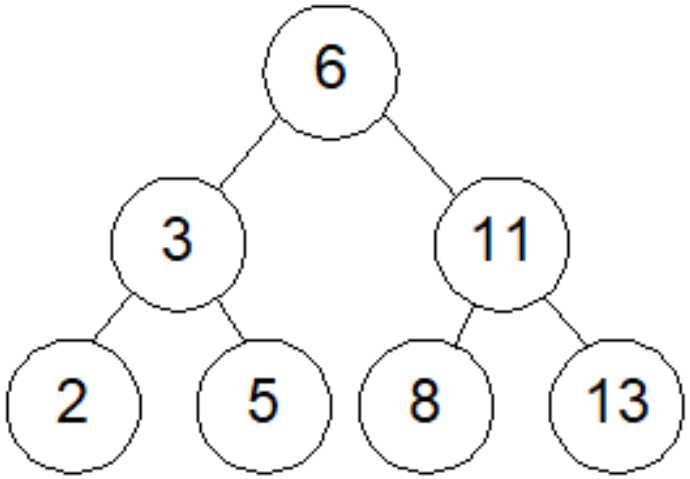
ΓΕΜΑΤΟ (FULL)

Δυαδικά δένδρα παραδείγματα



?

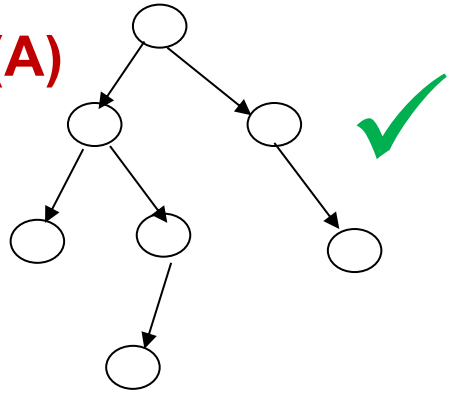
Δυαδικά δένδρα παραδείγματα



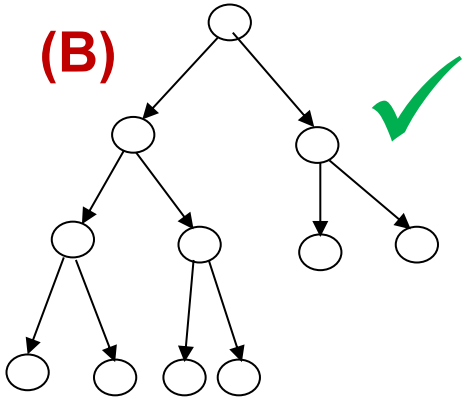
ΤΕΛΕΙΟ (PERFECT)

Δυαδικά δένδρα παραδείγματα

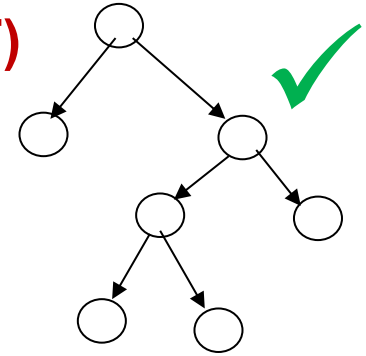
(Α)



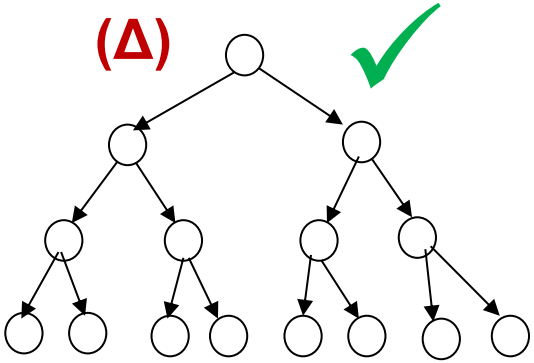
(Β)



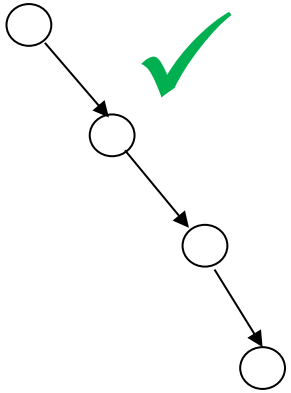
(Γ)



(Δ)



(Ε)



Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60

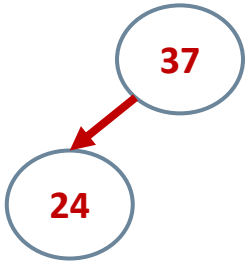
ΣΧΕΔΙΑΣΤΕ ΈΝΑ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΕΙΣΑΓΩΝΤΑΣ ΜΕ ΤΗΝ ΣΕΙΡΑ ΤΑ ΠΑΡΑΔΙΠΛΑ ΣΤΟΙΧΕΙΑ

Δυαδικά δένδρα - Υλοποίηση

» ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **37**

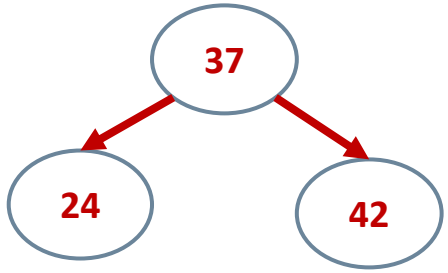
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **24**



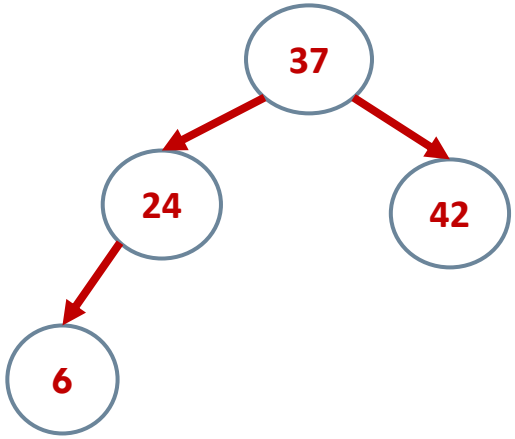
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **42**



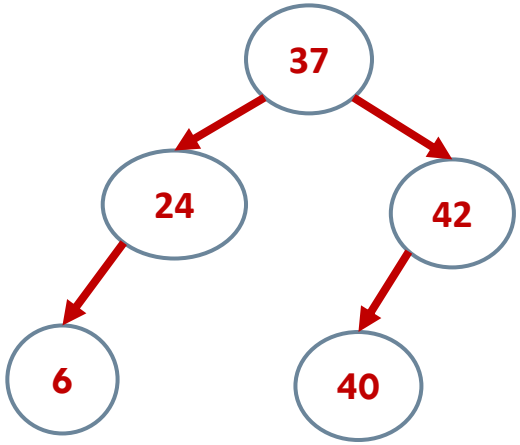
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **6**



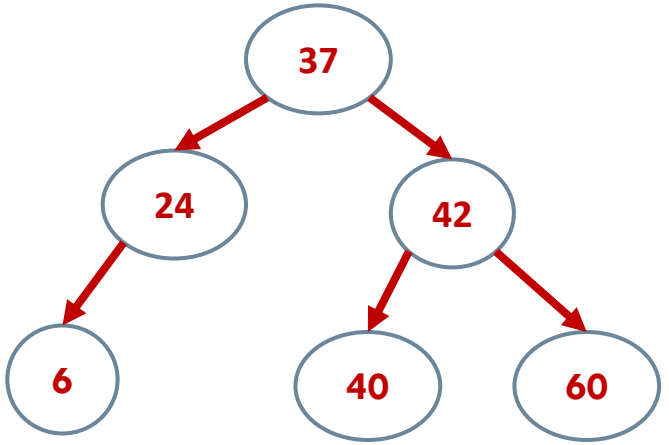
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **40**



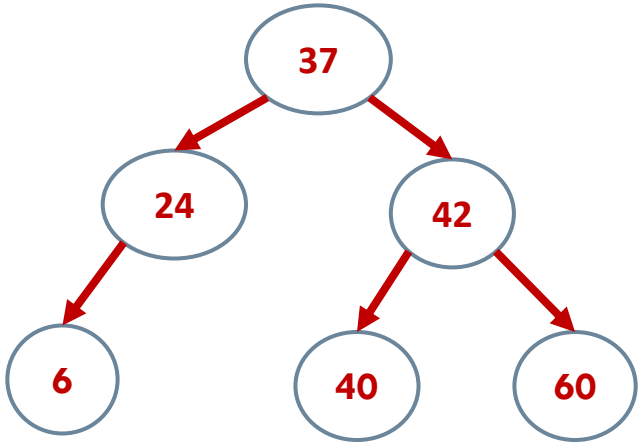
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ **60**



Δυαδικά δένδρα - Υλοποίηση

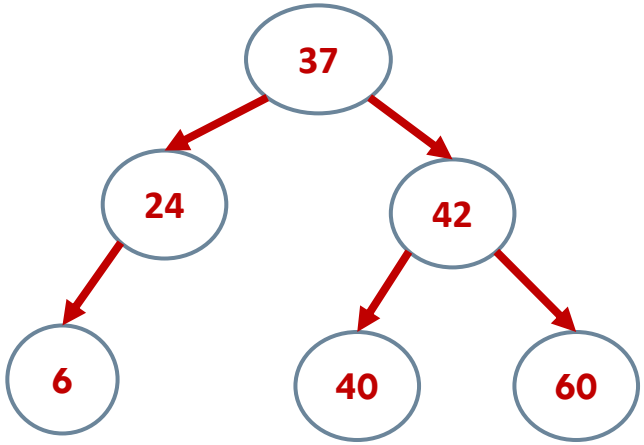
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60



ΕΙΝΑΙ ΤΕΛΕΙΟ;

Δυαδικά δένδρα - Υλοποίηση

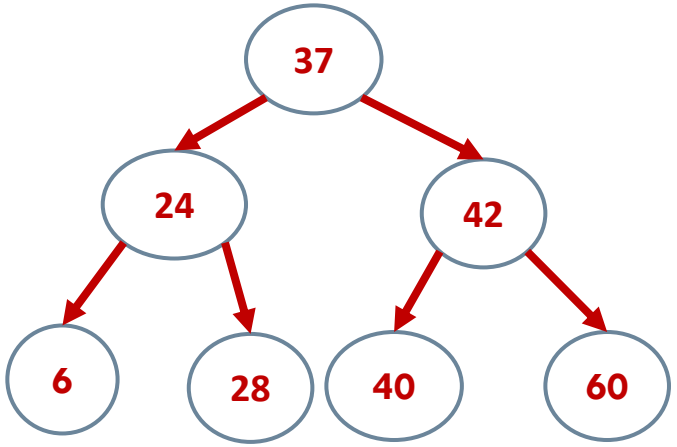
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60



ΠΟΤΕ ΘΑ ΓΙΝΕΙ ΤΕΛΕΙΟ;

Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60



ΜΕ ΤΗΝ ΕΙΣΑΓΩΓΗ ΕΝΟΣ ΚΟΜΒΟΥ ΜΕ ΤΙΜΗ X όπου:

$$24 \leq X < 37$$

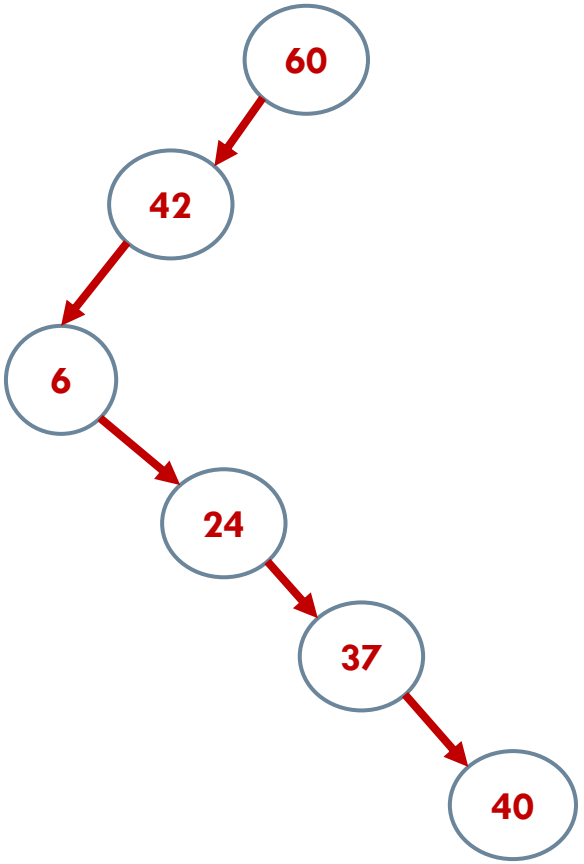
Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40

ΣΧΕΔΙΑΣΤΕ ΈΝΑ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΕΙΣΑΓΩΝΤΑΣ ΜΕ ΤΗΝ ΣΕΙΡΑ ΤΑ ΠΑΡΑΔΙΠΛΑ ΣΤΟΙΧΕΙΑ
(ΤΑ ΙΔΙΑ ΣΤΟΙΧΕΙΑ ΜΕ ΠΡΙΝ ΕΙΣΑΓΟΝΤΑΙ ΜΕ ΔΙΑΦΟΡΕΤΙΚΗ ΣΕΙΡΑ)

Δυαδικά δένδρα - Υλοποίηση

- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 60
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 42
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 6
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 24
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 37
- ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ 40



ΤΙ ΣΥΜΠΕΡΑΣΜΑ ΒΓΑΖΟΥΜΕ;

Σενάρια

- **Ποιο είναι το χειρότερο σενάριο?**

Οι τιμές εισόδου φθάνουν με την ακόλουθη σειρά

60,42,40,37,24,6 (φθίνουσα σειρά)

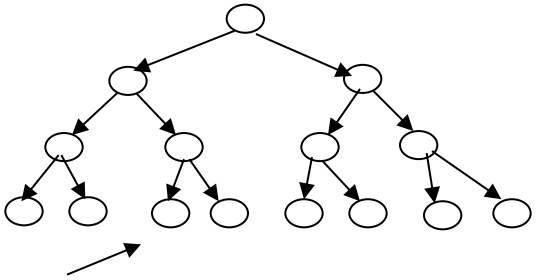
ή

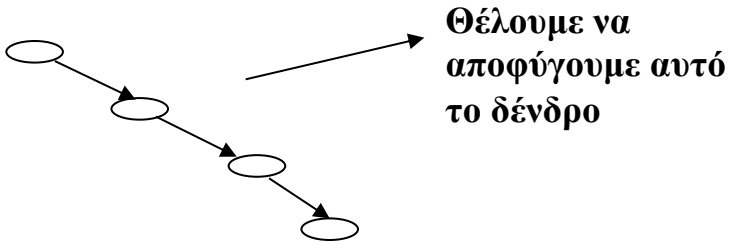
6,24,37,40,42,60 (αύξουσα σειρά)

Αυτές οι δυο τιμές εισόδου παράγουν μια συνδεμένη λίστα

Ιδέα Λύσης

- Βασικά θέλουμε να περιορίσουμε το ύψος του δένδρου όσο το δυνατό περισσότερο. Αυτό μπορεί να γίνει με:
  1. Σωστή αξιοποίηση όλων των παιδιών.





Το **τέλειο** δένδρο η είναι ιδανική περίπτωση!

  2. Να αυξήσουμε τον αριθμό των παιδιών σε κάθε κόμβο χωρίς να αυξηθεί πάρα πολύ. Γιατί;

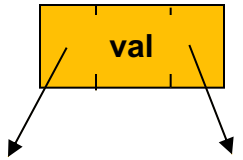
Διότι θα καταλήξουμε σε ένα πίνακα στο τέλος οπότε η αναζήτηση κάποιου στοιχείου θα πάρει t(n) εξέταση στοιχείων

Υλοποίηση Δυαδικών Δένδρων

- Αφού κάθε κόμβος σε ένα δυαδικό δένδρο έχει το πολύ δύο παιδιά, μπορούμε να κρατούμε δείκτες στο καθένα από αυτά.
- Δηλαδή, ένας κόμβος μπορεί να υλοποιηθεί ως μια εγγραφή NODE με τρία πεδία (παρόμοια με κόμβο διπλά συνδεδεμένης λίστας).
  1. **val**, όπου αποθηκεύουμε την πληροφορία του κόμβου,
  2. **left**, τύπου pointer, ο οποίος δείχνει το αριστερό, υπόδενδρο που ριζώνει στον συγκεκριμένο κόμβο, και
  3. **right**, τύπου pointer, ο οποίος δείχνει το δεξιό υπόδενδρο που ριζώνει στον συγκεκριμένο κόμβο.

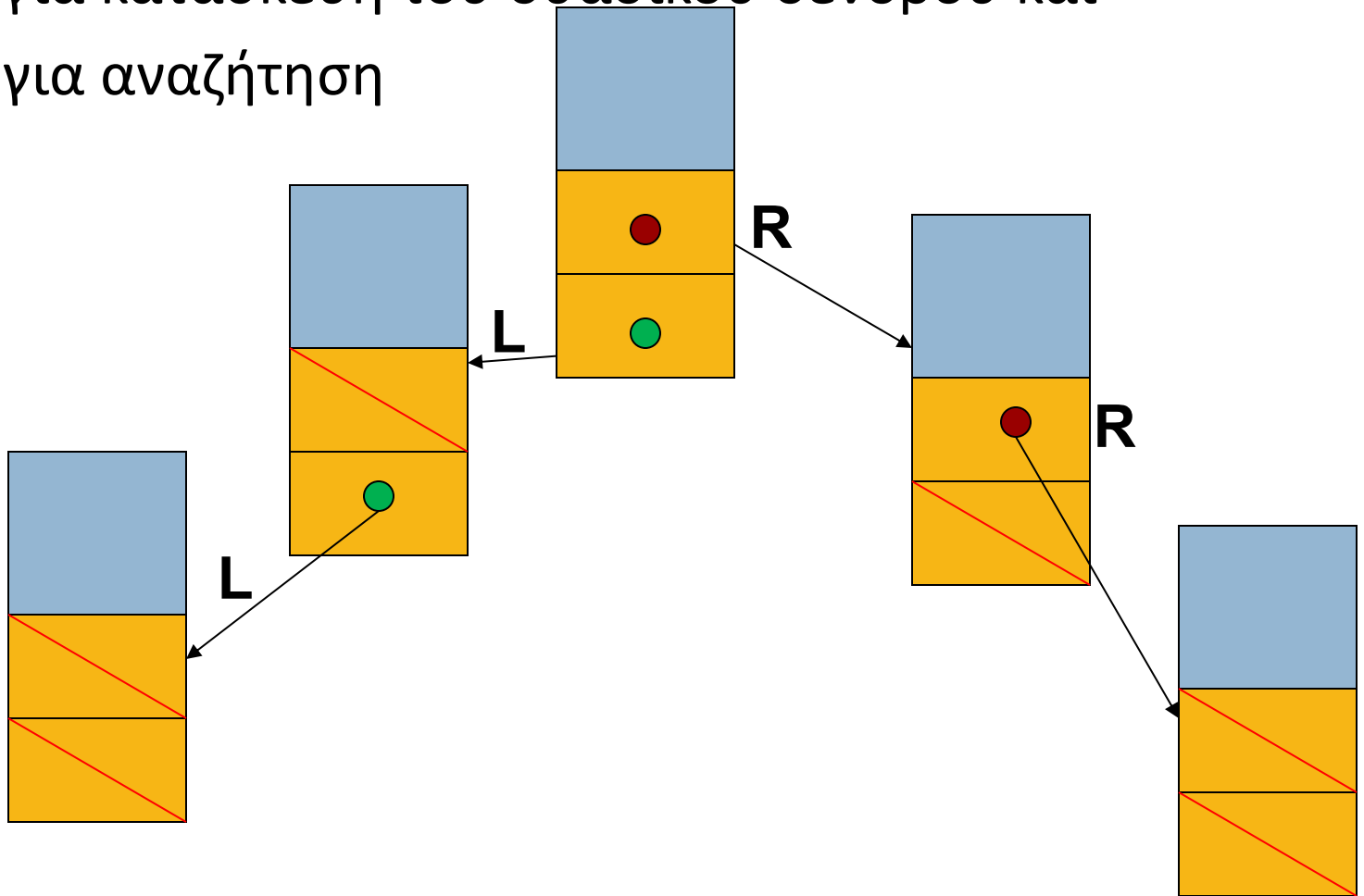
```
typedef struct node {
        int val;
        struct node *left;
        struct node *right;
} NODE;
```

NODE *root;



Δυαδικά Δένδρα

- Πρακτικές αναδρομικές συναρτήσεις
  - για κατασκευή του δυαδικού δένδρου και
  - για αναζήτηση

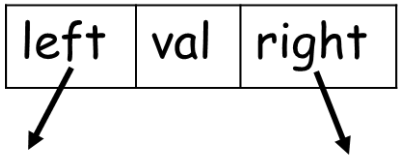


Δυαδικά δένδρα - Υλοποίηση

- Αφού κάθε κόμβος σε ένα δυαδικό δένδρο **έχει το πολύ δύο παιδιά**, μπορούμε να κρατούμε **δείκτες** στο καθένα από αυτά.

- Δηλαδή, ένας κόμβος μπορεί να υλοποιηθεί ως μια εγγραφή NODE με τρία πεδία (παρόμοια με κόμβο διπλά συνδεδεμένης λίστας). Τα πεδία αυτά είναι:
  1. **val**, όπου αποθηκεύουμε την πληροφορία του κόμβου,
  2. **left**, τύπου pointer, ο οποίος δείχνει το **αριστερό, υπόδενδρο** που ριζώνει στον συγκεκριμένο κόμβο, και
  3. **right**, τύπου pointer, ο οποίος δείχνει το **δεξιό υπόδενδρο** που ριζώνει στον συγκεκριμένο κόμβο.

```
typedef struct node {
          int val;
          struct node *left;
          struct node *right;
} NODE;
```



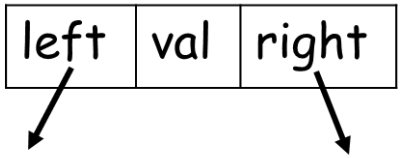
- Ξέροντας τα πιο πάνω πώς θα ορίζατε ένα καινούριο δέντρο;

Δυαδικά δένδρα - Υλοποίηση

- Αφού κάθε κόμβος σε ένα δυαδικό δένδρο **έχει το πολύ δύο παιδιά**, μπορούμε να κρατούμε **δείκτες** στο καθένα από αυτά.

- Δηλαδή, ένας κόμβος μπορεί να υλοποιηθεί ως μια εγγραφή NODE με τρία πεδία (παρόμοια με κόμβο διπλά συνδεδεμένης λίστας). Τα πεδία αυτά είναι:
  1. **val**, όπου αποθηκεύουμε την πληροφορία του κόμβου,
  2. **left**, τύπου pointer, ο οποίος δείχνει το **αριστερό, υπόδενδρο** που ριζώνει στον συγκεκριμένο κόμβο, και
  3. **right**, τύπου pointer, ο οποίος δείχνει το **δεξιό υπόδενδρο** που ριζώνει στον συγκεκριμένο κόμβο.

```
typedef struct node {
          int val;
          struct node *left;
          struct node *right;
} NODE;
```



- Έτσι, ένα δυαδικό δένδρο υλοποιείται **ως ένας δείκτης στη ρίζα του δένδρου**, δηλαδή δείκτης σε εγγραφή τύπου NODE → `NODE *root;`

Δυαδικά δένδρα – Υπολογισμός Ύψους Δένδρου

- **Αναδρομική Υλοποίηση**

```
int size(NODE *root)
{
  // τερματίζουμε αν βρήκαμε  κόμβο φύλλο
  if (root == NULL) {
       return 0;
  }
  return (size(root->left) + 1 + size(root->right));
}
```

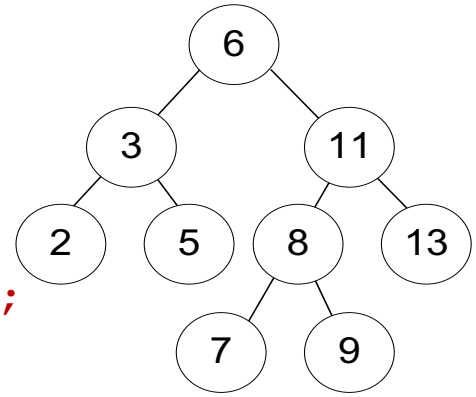
Σημειώστε ότι η μη-αναδρομική υλοποίηση χρειάζεται την χρήση μιας βοηθητικής δομής (π.χ. στοίβας)

Όμως η αναδρομική υλοποίηση είναι πιο εύκολα κατανοητή

Δυαδικά δένδρα – Εύρεση Ενός Κόμβου

- **Αναδρομική Υλοποίηση:**

```
NODE *FindNode(NODE *root, int val) {
  if (root == NULL)
       return NULL;  // το στοιχείο δεν βρέθηκε
  else if (root->val == val) // το στοιχείο βρέθηκε!
       return root;
  else {
       if (val < root->val)
              return FindNode(root->left, val);
       else
              return FindNode(root->right, val);
  }
}
```

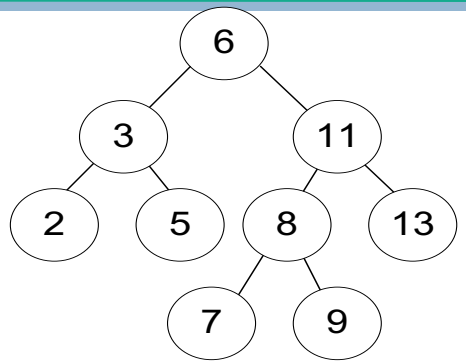


Δυαδικά δένδρα – Εισαγωγή Ενός Κόμβου

```
NODE *InsertNode(NODE *root, int val){
   if (root == NULL) {                          /* εισαγωγή κόμβου*/
         root = (NODE *) malloc(sizeof(NODE));
         if (root==NULL) {
                  printf("Out of Memory!");
                  exit(1);
         }
         root->val = val;
         root->left = NULL;
         root->right = NULL;
   }
   else {
         if (val < root->val)                   /* εισαγωγή αριστερά*/
                  root->left  = InsertNode(root->left, val);
         else                                   /*εισαγωγή δεξιά*/
                  root->right = InsertNode(root->right, val);
   }

   return root;
}
```

Δυαδικά δένδρα – Εισαγωγή Ενός Κόμβου

```
NODE *InsertNode(NODE *root, int val){
   if (root == NULL) {              /* εισαγωγή κόμβου*/
         root = (NODE *) malloc(sizeof(NODE));
         if (root==NULL) {
                  printf("Out of Memory!");
                  exit(1);
         }
         root->val = val;
         root->left = NULL;
         root->right = NULL;
   }
   else {
         if (val < root->val)              /* εισαγωγή αριστερά*/
                  root->left  = InsertNode(root->left, val);
         else                              /*εισαγωγή δεξιά*/
                  root->right = InsertNode(root->right, val);
   }

   return root;
}
```

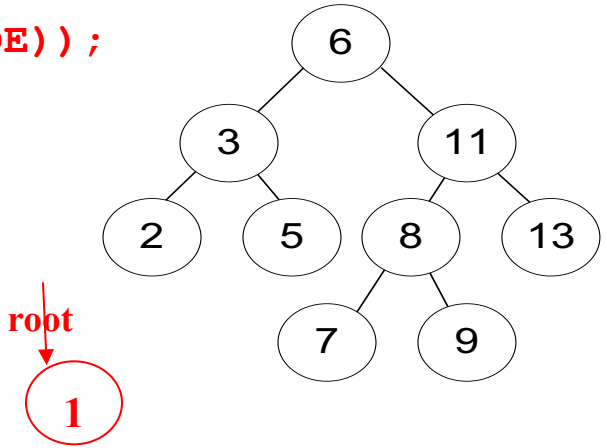


Εκτελέστε τα βήματα της πιο κάτω εντολής:

```
InsertNode(root, 1);
```

Διαδικασία Εισαγωγής Κόμβου – **InsertNode()**

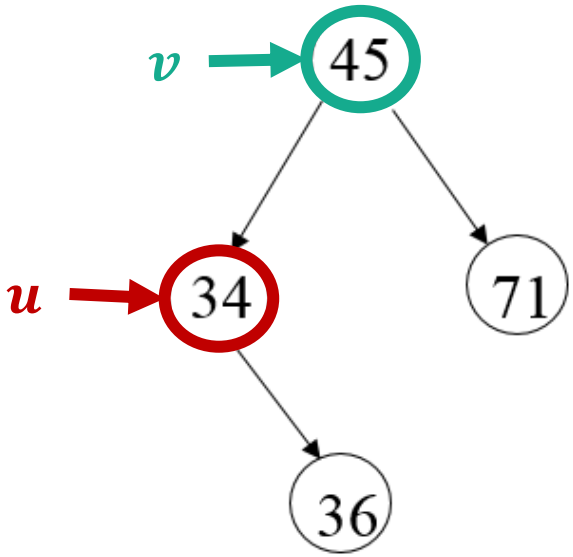
```
NODE *InsertNode(NODE *root, int val){
    if (root == NULL) { // εισαγωγή κόμβου
        root = (NODE *) malloc(sizeof(NODE));
        if (root==NULL) {
            printf("Out of Memory!");
            exit(1);
        }
        root->val = val;
        root->left = NULL;
        root->right = NULL;
    }
    else {
        if (val < root->val)          // εισαγωγή αριστερά
            root->left  = InsertNode(root->left, val);
        else                          // εισαγωγή δεξιά
            root->right = InsertNode(root->right, val);
    }

    return root;
}
```



Δυαδικά δένδρα – Διαγραφή Μικρότερου Κόμβου

1. **Ακολουθούμε τους αριστερούς δείκτες όσο μπορούμε, φθάνοντας στον κόμβο με το μικρότερο στοιχείο, u.**

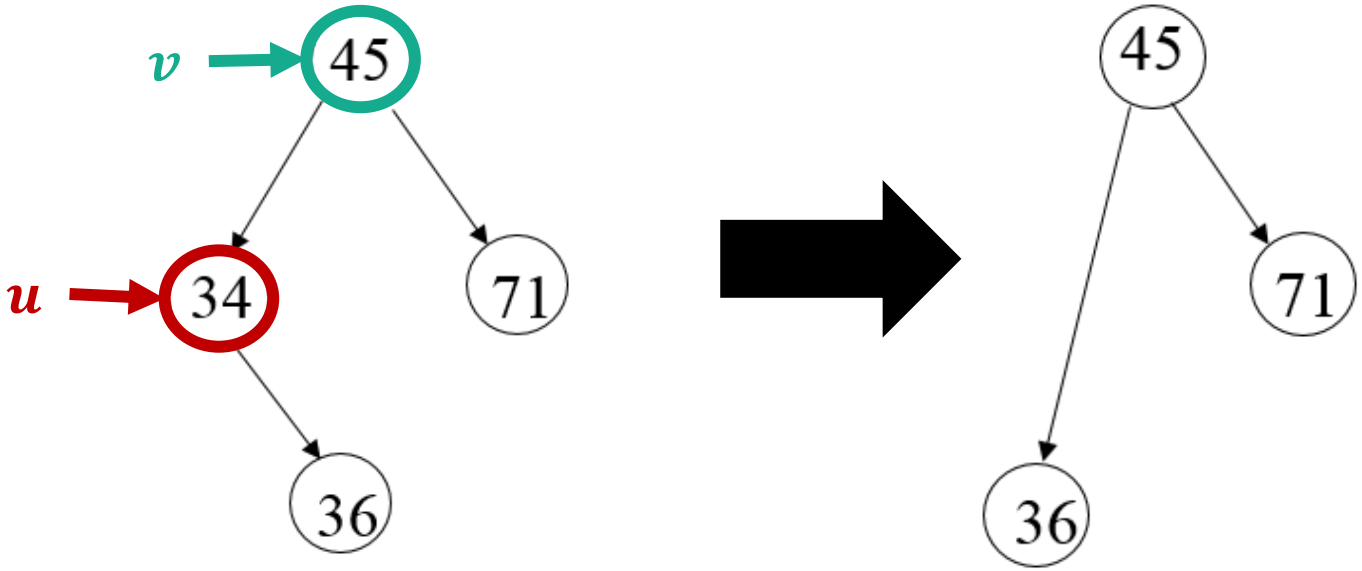
2. **Βρίσκουμε τον πατέρα v του u και αλλάζουμε τον αριστερό δείκτη του v ώστε να δείχνει στο δεξιό παιδί του u.**



Δυαδικά δένδρα – Διαγραφή Μικρότερου Κόμβου

1. **Ακολουθούμε τους αριστερούς δείκτες όσο μπορούμε, φθάνοντας στον κόμβο με το μικρότερο στοιχείο, u.**

2. **Βρίσκουμε τον πατέρα v του u και αλλάζουμε τον αριστερό δείκτη του v ώστε να δείχνει στο δεξιό παιδί του u.**

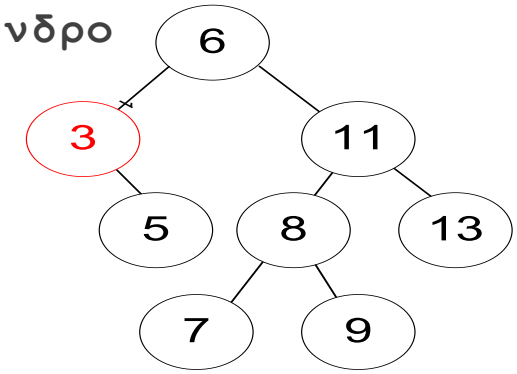


Δυαδικά δένδρα – Διαγραφή Μικρότερου Κόμβου

```
NODE *deleteMin(NODE *root) {
  NODE *p;
  if (root == NULL) {
    /* μικρότερο στοιχείο δεν βρέθηκε */
    return NULL;        /* πχ αν είναι κενό το δένδρο */
  }
  else if (root->left == NULL) {
       /*φτάσαμε στο μικρότερο (ΑΦΟΎ ΔΕΝ ΕΧΕΙ ΑΡΙΣΤΕΡΟ ΚΟΜΒΟ)*/
       p = root;
       root = root->right;
       free(p);
  }
  else {/* αναδρομική προχώρησε προς τα αριστερά*/
       root->left = deleteMin(root->left);
  }
  return root;
}
```

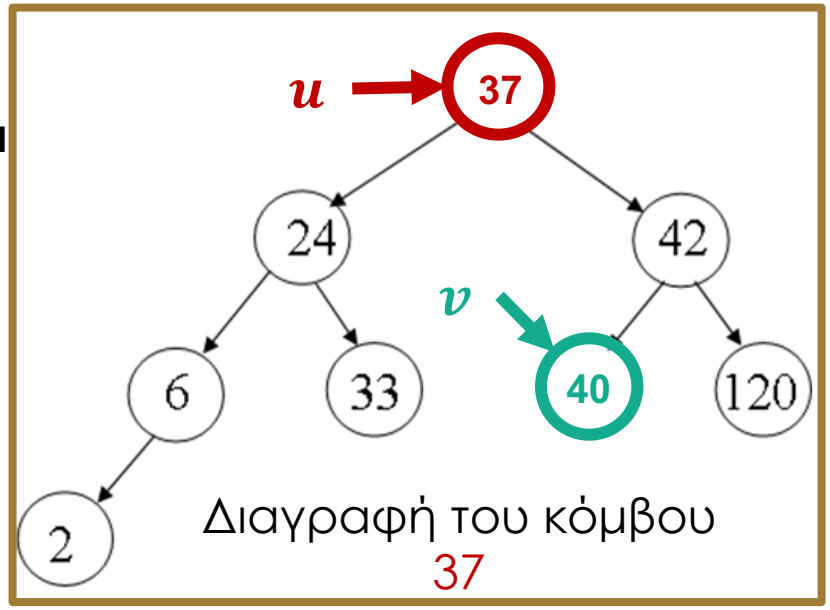
Δυαδικά δένδρα – Διαγραφή Μικρότερου Κόμβου

```
NODE *deleteMin(NODE *root) {
  NODE *p;
  if (root == NULL) {
    // μικρότερο στοιχείο δεν βρέθηκε
    return NULL;        // πχ αν είναι κενό το δένδρο
  }
  else if (root->left == NULL) {
       // φτάσαμε στο μικρότερο
       p = root;
       root = root->right;
       free(p);
  }
  else {// αναδρομική προχώρησε προς τα αριστερά
       root->left = deleteMin(root->left);
  }
  return root;
}
```



Δυαδικά δένδρα – Διαγραφή Κόμβου

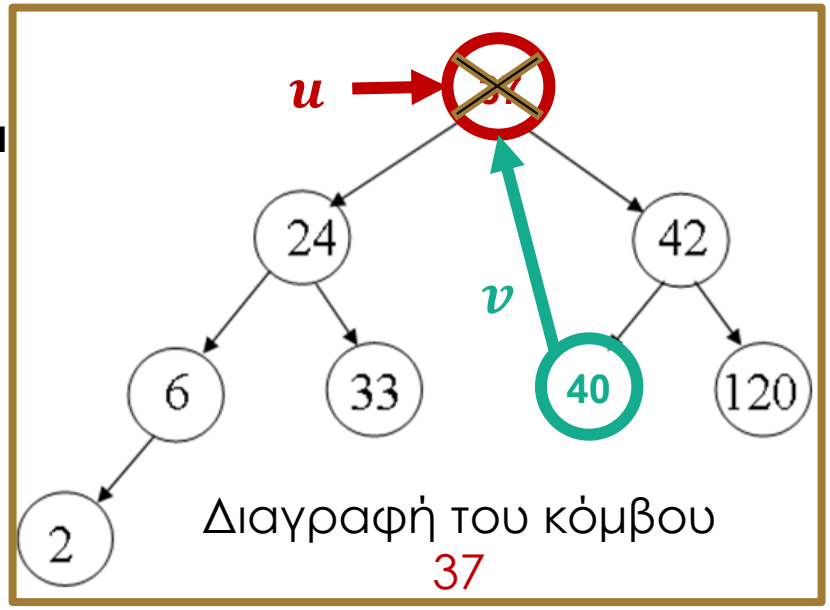
- **Για να αφαιρέσουμε έναν κόμβο Χ από ένα ΔΔΑ:**
  1. **Βρίσκουμε τον κόμβο u που περιέχει το Χ.**
  2. **Πριν διαγράψουμε τον u βρίσκουμε τον μικρότερο (από δεξιά) απόγονο του (τον οποίο ονομάζουμε v).**
  3. **Η ιδέα είναι ότι ο v πρέπει να αντικαταστήσει τον u!**



- Η υλοποίηση γίνεται πιο απλή χρησιμοποιώντας επαναληπτικούς βρόγχους παρά αναδρομή.

Δυαδικά δένδρα – Διαγραφή Κόμβου

- **Για να αφαιρέσουμε ένα κλειδί Χ από ένα ΔΔΑ:**
  1. **Βρίσκουμε τον κόμβο u που περιέχει το Χ.**
  2. **Πριν διαγράψουμε τον u βρίσκουμε τον μικρότερο (από δεξιά) απόγονο του (τον οποίο ονομάζουμε v).**
  3. **Η ιδέα είναι ότι ο v πρέπει να αντικαταστήσει τον u!**

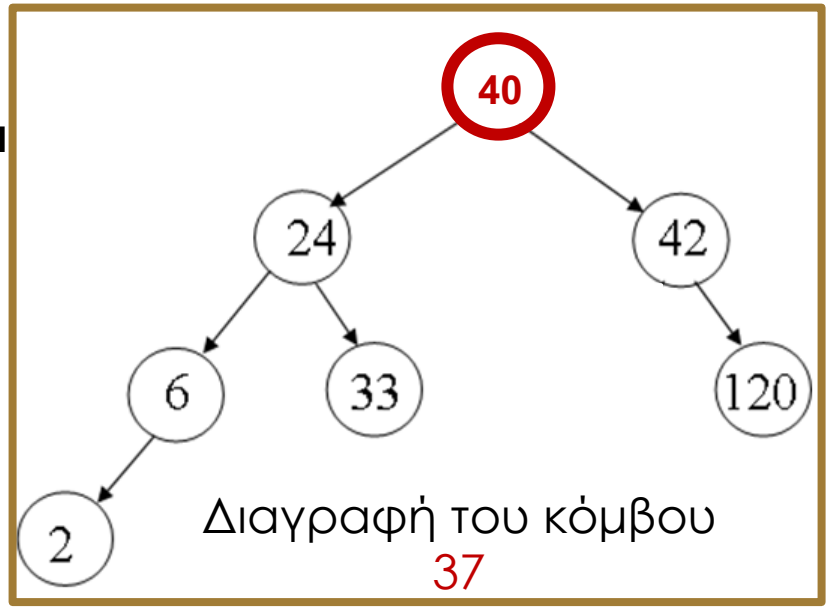


Διαγραφή του κόμβου

- Η υλοποίηση γίνεται πιο απλή χρησιμοποιώντας επαναληπτικούς βρόγχους παρά αναδρομή.

Δυαδικά δένδρα – Διαγραφή Κόμβου

- **Για να αφαιρέσουμε ένα κλειδί Χ από ένα ΔΔΑ:**
  1. **Βρίσκουμε τον κόμβο u που περιέχει το Χ.**
  2. **Πριν διαγράψουμε τον u βρίσκουμε τον μικρότερο (από δεξιά) απόγονο του (τον οποίο ονομάζουμε v).**
  3. **Η ιδέα είναι ότι ο v πρέπει να αντικαταστήσει τον u!**

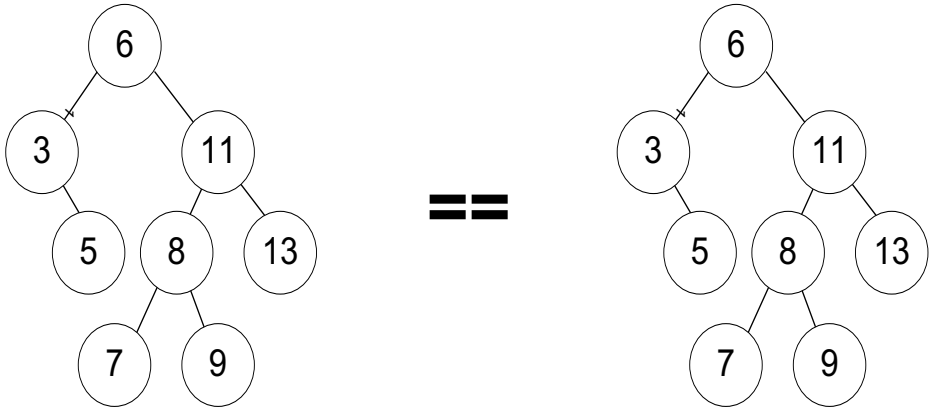


Διαγραφή του κόμβου 37

- Η υλοποίηση γίνεται πιο απλή χρησιμοποιώντας επαναληπτικούς βρόγχους παρά αναδρομή.

Δυαδικά δένδρα – Σύγκριση Δένδρων

Να γραφεί μια συνάρτηση η οποία να επιστρέφει “1” αν δυο δυαδικά δένδρα είναι τα ίδια και “0” στην αντίθετη περίπτωση.



Δυαδικά δένδρα – Σύγκριση Δένδρων

```
int sameTree(NODE *a, NODE *b) {
  // και τα δυο δένδρα είναι κενά => άρα επιστρέφουμε TRUE
  if (a==NULL && b==NULL)
       return 1;
  // και τα δυο δένδρα δεν είναι κενά – συγκρίνουμε τις ρίζες τους
  else if (a!=NULL && b!=NULL) {
    return(
      (a->data == b->data)
      &&
      sameTree(a->left, b->left)
      &&
      sameTree(a->right, b->right)
    );
  }
  // το ένα εκ των δυο υπό-δένδρων είναι κενό => επιστρέφουμε FALSE
  else return 0;
}
```

Άσκηση

Να γράψετε διαδικασία σε γλώσσα C, η οποία με δεδομένα εισόδου, ένα δείκτη στην ρίζα ενός δυαδικού δένδρου και ένα ακέραιο k, τυπώνει **όλα τα στοιχεία που βρίσκονται σε βάθος k του δένδρου**. Η διαδικασία μπορεί να είναι αναδρομική.

Πρότυπο συνάρτησης: **void printKLevel(NODE *root, int k)**

Ενδεικτική Λύση

```
void printKLevel(NODE *root, int k) {
   if ((root == NULL) || (k <1))
      return;
   if (k == 1) {
      printf("%d", root->val);
      return;
   }
   printKLevel(root->left, k-1);
   printKLevel(root->right, k-1);
}
```

Ευχαριστώ για την προσοχή σας

Επικοινωνία

- Skype: fidas.christos
- Email: fidas@upatras.gr
- Phone: 2610 – 996491
- Web: http://cfidas.info

Άμεση Επικοινωνία μέσω Skype

SkypeID:
fidas.christos

- Ώρες γραφείου: Τετάρτη & Παρασκεύη 11:00-13:00

Join Zoom Meeting
https://upatras-gr.zoom.us/j/95080297961?pwd=MzRta0JRd3ZwVEVrREZNc09qbG1Zdz09

Το υλικό της διάλεξης είναι διαθέσιμο στο eclass

- https://eclass.upatras.gr/

ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

11η Εβδομάδα: Στοίβες, Ουρές και Αναδρομή

Αναφορές

Οι διαφάνειες της διάλεξης στηρίζονται, εν μέρει, σε υλικό παραδόσεων παλαιότερων ετών του **Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογία Υπολογιστών του Πανεπιστημίου Πατρών** καθώς και του **Τμήματος Πληροφορικής του Πανεπιστήμιο Κύπρου.**

Ενώσεις (Unions)

Ο τύπος **union** χρησιμοποιείται στη C για τη δήλωση μεταβλητών που μπορούν να αποθηκεύσουν σε διαφορετικές στιγμές δεδομένα διαφορετικών τύπων και μεγεθών στην ίδια περιοχή μνήμης.

Θεωρήστε την περίπτωση που θέλουμε μια σταθερά σε ένα πρόγραμμα να έχει ως τιμή είτε μια ακέραια τιμή (int) είτε μια πραγματική τιμή (float). Θα μπορούσαμε να γράψουμε:

```
struct {
        int ival;
        float fval;
} constant;
```

και να τοποθετούμε τη σταθερά μας σε ένα από τα δύο πεδία της δομής σύμφωνα με την τιμή της.

- Μειονέκτημα αυτής της λύσης είναι ότι σπαταλά άσκοπα μνήμη: δεσμεύει μνήμη για δύο πεδία ενώ θα χρησιμοποιηθεί μόνο ένα.

Ενώσεις (Unions)

Χρησιμοποιώντας ενώσεις η δήλωση της μεταβλητής μας γίνεται ως εξής:

```
union {
        int ival;
        float fval;
} constant;
```

• Η μνήμη που δεσμεύεται σε αυτή την περίπτωση είναι η μέγιστη που απαιτείται για αποθήκευση ακριβώς μιας μεταβλητής από οποιοδήποτε από τους τύπους των πεδίων (στην πιο πάνω περίπτωση η μνήμη που απαιτείται για αποθήκευση είναι 4 bytes).

• Η τιμή που ανακτάται κάθε φορά είναι αυτή που αποθηκεύτηκε πιο πρόσφατα κατά την εκτέλεση. Είναι ευθύνη του προγραμματιστή να παρακολουθεί και να γνωρίζει ποιος τύπος είναι αποθηκευμένος σε μια ένωση.

Παράδειγμα - UNION


```
#include <stdio.h>
#define CHARACTER 'C'
#define INTEGER 'I'
#define FLOAT 'F'
struct generic_tag {
        char type;
        union shared_tag {
                char c;
                int i;
                double f;
        } shared;
};


void print_function(struct generic_tag generic);
```

Παράδειγμα - UNION


```
#include <stdio.h>
#define CHARACTER 'C'
#define INTEGER 'I'
#define FLOAT 'F'
struct generic_tag {
        char type;
        union shared_tag {
                char c;
                int i;
                double f;
        } shared;
} MYVAR;


void print_function(struct generic_tag generic);
```

Παράδειγμα - UNION


```
main()
{
   MYVAR var;
   var.type = CHARACTER;
   var.shared.c = '$';
   print_function(var);
   var.type = FLOAT;
   var.shared.f = 12.6;
   print_function(var);
   var.type = INTEGER;
   var.shared.i = 111;
   print_function(var);
return 0;
}
```

Παράδειγμα - UNION


```
void print_function(MYVAR generic) {
        switch(generic.type) {
                case CHARACTER:
                     printf("%c", generic.shared.c);
                break;

                case INTEGER:
                    printf("%d", generic.shared.i);
                break;

                case FLOAT:
                    printf("%.2f", generic.shared.f);
                break;

                default:
                  printf("an unknown type: %c\n", generic.type);
                break;
        }

}
```

Ευχαριστώ για την προσοχή σας

Επικοινωνία

- Skype: fidas.christos
- Email: fidas@upatras.gr
- Phone: 2610 – 996491
- Web: http://cfidas.info

Άμεση Επικοινωνία μέσω Skype

SkypeID:
fidas.christos

- Ώρες γραφείου: Τετάρτη & Παρασκεύη 11:00-13:00

Join Zoom Meeting
https://upatras-gr.zoom.us/j/95080297961?pwd=MzRta0JRd3ZwVEVrREZNc09qbG1Zdz09

Το υλικό της διάλεξης είναι διαθέσιμο στο eclass

- https://eclass.upatras.gr/