ΑΛΓΟΡΙΘΜΟΙ
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ
ΠΑΡΑΛΛΗΛΙΑ
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΑΛΓΟΡΙΘΜΟΙ- ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

- Η έννοια του Αλγορίθμου
  - Ιδιότητες
- Σχεδίαση Αλγορίθμων
  - Κατά-βήματα-ανάλυση
- Προγραμματισμός
  - Δομικοί λίθοι προγραμματισμού
- Δομημένος Προγραμματισμός
  - Μεθοδολογία
- Δομικά διαγράμματα
  - Δομημένου προγραμματισμού

Η Έννοια του ΑΛΓΟΡΙΘΜΟΥ

Τυπικές Διαδικασίες	Αλγόριθμος	Βήματα π.χ.
πλέξιμο πουλόβερ	σχέδιο πλεξίματος	κόμποι
παιδικό μοντέλο αεροπλάνου	οδηγίες συναρμολόγησης	κόλλα Α στο Β
ψήσιμο κέικ	συνταγή	χτύπα 3 αυγά
ράψιμο φορέματος	πατρόν	ράψε το μανίκι
παίξιμο σονάτας	παρτιτούρα	νότες

Αλγόριθμος
↓
Προγραμματισμός
↓
Πρόγραμμα σε γλώσσα υψηλού επιπέδου
↓
Μετάφραση
↓
Γλώσσα Μηχανής
↓
Εκτέλεση

Προέλευση του όρου Αλγόριθμος

- Προέρχεται από τον Πέρση (Khiva, 780 μ.Χ) Mukhammad ihn Musa abu Djafar al-Khorezmi
  - έγραψε δύο έργα (κιτάπια), ένα για Αλγορίθμους και ένα για Άλγεβρα
    - **Kitab hisab al-adad al-hindi→ algorithmic de numero indorum→ liber algorithmi**
    - **Kitab al-jabr-wal-muqabala**

- ΑΛΓΟΡΙΣΤΕΣ←→ ΑΒΑΚΙΣΤΕΣ
- ΑΛΓΟΡΙΘΜΟΣ : ΓΕΝΙΚΗ ΜΕΘΟΔΟΣ/ΔΙΑΔΙΚΑΣΙΑ
  ΣΥΝΤΑΓΗ/ΤΕΧΝΙΚΗ/ΡΟΥΤΙΝΑ

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΑΛΓΟΡΙΘΜΩΝ

- ΒΑΣΙΚΕΣ ΙΔΙΟΤΗΤΕΣ
  - ΠΕΠΕΡΑΣΜΕΝΗ ΠΕΡΙΓΡΑΦΗ
  - ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ
- ΠΕΠΕΡΑΣΜΕΝΗ ΠΕΡΙΓΡΑΦΗ =ΚΕΙΜΕΝΟ ΜΕ ΠΕΠΕΡΑΣΜΕΝΟ ΜΗΚΟΣ (ΠΡΟΓΡΑΜΜΑ) ΑΠΟΤΕΛΟΥΜΕΝΟ ΑΠΟ ΒΗΜΑΤΑ
- Π.χ., ΑΛΓΟΡΙΘΜΟΣ ΕΥΚΛΕΙΔΗ : μ.κ.δ.(x,y) x,y>0

ΒΗΜΑ 1: [ΕΥΡΕΣΗ ΥΠΟΛΟΙΠΟΥ]
ΔΙΑΙΡΕΣΕ ΤΟ x με το y και ΕΣΤΩ r ΤΟ ΥΠΟΛΟΙΠΟ.
ΒΗΜΑ 2: [ΥΠΟΛΟΙΠΟ ΜΗΔΕΝ;]
ΕΑΝ r=0 ΤΟΤΕ Ο ΑΛΓΟΡΙΘΜΟΣ ΤΕΡΜΑΤΙΖΕΙ ΚΑΙ Ο μ.κ.δ = y.
ΒΗΜΑ 3 :[ΑΝΤΑΛΛΑΓΗ]
ΒΑΛΕ ΣΤΗ ΘΕΣΗ ΤΟΥ x ΤΟ y (x ← y)
ΚΑΙ ΣΤΗ ΘΕΣΗ ΤΟΥ y ΤΟ r (y ← r).
ΠΗΓΑΙΝΕ ΣΤΟ ΒΗΜΑ 1.

μκδ(x,y) = μκδ(y, υπόλοιπο του x/y) , y>0
μκδ(x,y) = x , εάν y=0

ΑΠΟΤΕΛΕΣΜΑΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ

ΘΕΣΗ CHURCH: Κάθε υπολογισμός για τον οποίο υπάρχει αποτελεσματική διαδικασία μπορεί να πραγματοποιηθεί από μια μηχανή Turing

ΘΕΣΗ TURING: Αποτελεσματική διαδικασία είναι αυτή που μπορεί να διεκπεραιωθεί από μια μηχανή Turing

ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ

ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ = ΚΑΘΕ ΒΗΜΑ ΝΑ ΜΠΟΡΕΙ ΝΑ ΕΚΤΕΛΕΣΤΕΙ ΚΑΤΑ ΜΗΧΑΝΙΚΟ ΤΡΟΠΟ

ΒΗΜΑ: ΣΤΟΙΧΕΙΩΔΕΣ, ΜΕ ΑΚΡΙΒΕΙΑ ΟΡΙΣΜΕΝΟ, ΚΑΙ ΠΕΠΕΡΑΣΜΕΝΟ ΣΕ ΧΡΟΝΟ.

Π.χ.: Είναι αποτελεσματικό το παρακάτω Βήμα ;

ΒΗΜΑ i = Εάν 2 είναι ο μεγαλύτερος ακέραιος n για τον οποίο υπάρχουν θετικοί ακέραιοι x,y,z με $x^n + y^n = z^n$ τότε πήγαινε στο ΒΗΜΑ j

ΑΛΛΕΣ ΙΔΙΟΤΗΤΕΣ ΑΛΓΟΡΙΘΜΩΝ ΤΕΡΜΑΤΙΣΜΟΣ

- ΤΕΡΜΑΤΙΣΜΟΣ
  - **ΤΕΡΜΑΤΙΖΩΝ ΑΛΓΟΡΙΘΜΟΣ (π.χ., του Ευκλείδη)**
  - **ΜΗ-ΤΕΡΜΑΤΙΖΩΝ ΑΛΓΟΡΙΘΜΟΣ = ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΘΟΔΟΣ**

Π.χ.1, υπολογισμός του e (η βάση των φυσικών λογαρίθμων)

$$A_0 = 1,\ A_1 = 2,\ B_0 = 0,\ B_1 = 1$$
$$A_{i+1} = (4i+2)A_i + A_{i-1}$$
$$B_{i+1} = (4i+2)B_i + B_{i-1}$$
$$\text{ρητός } e = \frac{A_i + B_i}{A_i - B_i}$$
$$e = \frac{3}{1}, \frac{19}{7}, \frac{193}{71}, \frac{27}{1001}, \frac{49171}{18089}, \ldots$$

Π.χ.2: εύρεση των πρώτων αριθμών

ΑΛΛΕΣ ΙΔΙΟΤΗΤΕΣ ΑΛΓΟΡΙΘΜΩΝ ΑΙΤΙΟΚΡΑΤΙΑ

- ΑΙΤΙΟΚΡΑΤΙΑ= ΜΟΝΟΣΗΜΑΝΤΑ ΚΑΘΟΡΙΣΜΕΝΗ ΣΕΙΡΑ ΕΚΤΕΛΕΣΗΣ ΤΩΝ ΒΗΜΑΤΩΝ (ίδιο αποτέλεσμα)
  - **ΑΙΤΙΟΚΡΑΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ**
    Π.χ., ο αλγόριθμος του Ευκλείδη
  - **ΜΗ-ΑΙΤΙΟΚΡΑΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ**
    (όχι απαραίτητα ίδιο αποτέλεσμα)
    - **Π.χ.: εισαγωγή ενός στοιχείου x σε μια οποιαδήποτε θέση μιας ακολουθίας στοιχείων**

x

.....

ΑΠΟΔΟΤΙΚΟΤΗΤΑ

- ΑΠΟΔΟΤΙΚΟΤΗΤΑ= ΠΛΗΘΟΣ ΒΗΜΑΤΩΝ ΠΟΥ ΑΠΑΙΤΟΥΝΤΑΙ
  - **ΘΕΩΡΙΑ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ → ΧΡΟΝΟΣ, ΜΝΗΜΗ**
    - Κλάσεις πολυπλοκότητας: EXP, NP, P, Log
  - **ΠΡΑΚΤΙΚΟΤΗΤΑ ΑΛΓΟΡΙΘΜΟΥ**

Π.χ., ΠΡΟΒΛΗΜΑ ΣΚΑΚΙΟΥ :
ΜΠΟΡΟΥΝ ΝΑ ΠΑΙΞΟΥΝ ΤΑ ΛΕΥΚΑ ΚΑΙ ΝΑ ΝΙΚΗΣΟΥΝ ΑΚΟΜΑ ΚΑΙ ΑΝ ΤΑ ΜΑΥΡΑ ΠΑΙΞΟΥΝ ΚΑΤΑ ΑΡΙΣΤΟ ΤΡΟΠΟ;

Η ΣΧΕΔΙΑΣΗ ΑΛΓΟΡΙΘΜΩΝ

- Κατά-βήματα ανάλυση αλγορίθμων
- Δομικοί λίθοι – προγραμματιστικά μορφώματα
  - Ακολουθία Βημάτων - Sequence
  - Επιλογή Βημάτων - Selection if then else
  - Επανάληψη -Repeat until, while do
  - Αναδρομή -Recursion
  - Στοιχειοποίηση -Modules
  - Παραλληλία -Parallelism
  - Δομημένος προγραμματισμός

ΚΑΤΑ ΒΗΜΑΤΑ ΑΝΑΛΥΣΗ 1

{ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΠΑΡΑΣΚΕΥΗ ΚΑΦΕ}
(1)ΒΡΑΣΕ ΝΕΡΟ
(1.1)ΓΕΜΙΣΕ ΤΗ ΧΥΤΡΑ
(1.1.1)ΒΑΛΕ ΤΗ ΧΥΤΡΑ ΚΑΤΩ ΑΠΟ ΤΗ ΒΡΥΣΗ
(1.1.2)ΑΝΟΙΞΕ ΤΗ ΒΡΥΣΗ
(1.1.3)ΚΛΕΙΣΕ ΤΗ ΒΡΥΣΗ
(1.2)ΑΝΟΙΞΕ ΤΟ ΔΙΑΚΟΠΤΗ
(1.3)ΠΕΡΙΜΕΝΕ ΝΑ ΒΡΑΣΕΙ
(1.3.1)ΠΕΡΙΜΕΝΕ ΈΩΣ ΟΤΟΥ Η ΧΥΤΡΑ ΑΡΧΙΖΕΙ ΝΑ ΣΦΥΡΙΖΕΙ
(1.4)ΚΛΕΙΣΕ ΤΟΝ ΔΙΑΚΟΠΤΗ
(2)ΒΑΛΕ ΤΟΝ ΚΑΦΕ ΣΤΟ ΦΛΙΤΖΑΝΙ
(2.1)ΑΝΟΙΞΕ ΤΟ ΒΑΖΟ ΤΟΥ ΚΑΦΕ
(2.1.1)ΠΑΡΕ ΤΟ ΒΑΖΟ ΑΠΟ ΤΟ ΡΑΦΙ
(2.1.2)ΑΝΟΙΞΕ ΤΟ ΚΑΠΑΚΙ
(2.2)ΠΑΡΕ ΜΙΑ ΚΟΥΤΑΛΙΑ ΚΑΦΕ
(2.3)ΡΙΞΕ ΤΗ ΣΤΟ ΦΛΙΤΖΑΝΙ
(2.4)ΚΛΕΙΣΕ ΤΟ ΒΑΖΟ
(2.4.1)ΒΑΛΕ ΤΟ ΚΑΠΑΚΙ
(2.4.2)ΒΑΛΕ ΤΟ ΒΑΖΟ ΣΤΟ ΡΑΦΙ
(3)ΠΡΟΣΘΕΣΕ ΝΕΡΟ ΣΤΟ ΦΛΙΤΖΑΝΙ
(3.1)ΡΙΞΕ ΝΕΡΟ ΑΠΟ ΤΗ ΧΥΤΡΑ ΈΩΣ ΟΤΟΥ ΓΕΜΙΣΕΙ ΤΟ ΦΛΥΤΖΑΝΙ

ΚΑΤΑ ΒΗΜΑΤΑ_ΑΝΑΛΥΣΗ 2/1

{ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΕΚΤΥΠΩΣΗ ΤΡΙΓΩΝΟΥ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ}

```
1
2       4
3       6       9
4       8       12      16
5       10      15      20      25
6       12      18      24      30      36
```

```
(1) ΤΥΠΩΣΕ ΤΟΝ ΠΙΝΑΚΑ ΠΟΛΛΑΠΛΑΣΙΑΣΜΩΝ (N)
    (1.1) ΑΡΙΘΜΟΣ ΓΡΑΜΜΗΣ = 1;
              WHILE     ΑΡΙΘΜΟΣ_ΓΡΑΜΜΗΣ =< N DO
                  BEGIN
                        ΓΡΑΨΕ ΜΙΑ ΓΡΑΜΜΗ ΤΟΥ ΠΙΝΑΚΑ;
                        WRITELN; {ΑΛΛΑΓΗ ΓΡΑΜΜΗΣ}
                        ΑΡΙΘΜΟΣ_ΓΡΑΜΜΗΣ=ΑΡΙΘΜΟΣ_ΓΡΑΜΜΗΣ+1
                  END
```


ΚΑΤΑ ΒΗΜΑΤΑ ΑΝΑΛΥΣΗ 2/2

```
(1.1.4)     {ΓΡΑΦΕ ΜΙΑ ΓΡΑΜΜΗ ΤΟΥ ΠΙΝΑΚΑ}
                ΑΡΙΘΜΟΣ_ΣΤΗΛΗΣ=1;
                WHILE ΑΡΙΘΜΟΣ_ΣΤΗΛΗΣ=<ΑΡΙΘΜΟΣ_ΓΡΑΜΜΗΣ DO
                        BEGIN
                        ΓΡΑΨΕ ΕΝΑ ΣΤΟΙΧΕΙΟ ΤΟΥ ΠΙΝΑΚΑ
                        ΑΡΙΘΜΟΣ_ΣΤΗΛΗΣ=ΑΡΙΘΜΟΣ_ΣΤΗΛΗΣ+1
                        END
        (1.1.4.1){ΓΡΑΨΕ ΕΝΑ ΣΤΟΙΧΕΙΟ ΤΟΥ ΠΙΝΑΚΑ}
                WRITE (ΑΡΙΘΜΟΣ_ΓΡΑΜΜΗΣ * ΑΡΙΘΜΟΣ_ΣΤΗΛΗΣ)
```

Συνολικό Πρόγραμμα

```
ARGRAMIS=1;
WHILE ARGRAMIS<=N DO
    BEGIN
      ARSTILIS=1;
      WHILE ARSTILIS<=ARGRAMIS DO
        BEGIN
          WRITE(ARGRAMIS*ARSTILIS)
          ARSTILIS=ARSTILIS+1
        END;
      WRITELN;
      ARGRAMIS=ARGRAMIS+1
    END
```


Πρόγραμμα εκτύπωσης του πίνακα πολλαπλασιασμών σε C

```
/*Arxeio stoixeiwn*/
/* Dhlwseis ypoprogrammatwv */
void grapse_ena_stoixeio_toy_pinaka(int arithmos_grammhs, int arithmos_sthlhs);
void grapse_mia_grammh_toy_pinaka(int arithmos_grammhs);
void typose_ton_pinaka_pollaplasiasmwn(int n);
/* Kyriws programma. Diabazei apo to pliktrologio tov arithmo grammwn
 * toy pinaka */
int main()
{int n;
 /* Diabase tov arithmo grammwv apo to pliktrologio */
 printf("Dwse arithmo grammwv: ");
 scanf("%d", &n);
 typose_ton_pinaka_pollaplasiasmwn(n);
 return 0;}

/* Typwnei enan pinaka n grammwn */
void typose_ton_pinaka_pollaplasiasmwn(int n)
{int arithmos_grammhs = 1;
 while (arithmos_grammhs <= n)
  {grapse_mia_grammh_toy_pinaka(arithmos_grammhs);
   printf("\n");
   arithmos_grammhs++;}}

/* Typwvei tn grammh yp' arithmo arline toy pinaka */
void grapse_mia_grammh_toy_pinaka(int arithmos_grammhs)
{int arithmos_sthlhs = 1;
 while (arithmos_sthlhs <= arithmos_grammhs) {
   grapse_ena_stoixeio_toy_pinaka(arithmos_grammhs, arithmos_sthlhs);
   arithmos_sthlhs++;}}

/* Typwnei to stoixeio toy pinaka gia sygkekrimenh grammh kai sthlh */
void grapse_ena_stoixeio_toy_pinaka(int arithmos_grammhs, int arithmos_sthlhs)
{printf("%d ", arithmos_grammhs * arithmos_sthlhs);}
```


ΒΑΣΙΚΑ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΑ ΜΟΡΦΩΜΑΤΑ

ΒΑΣΙΚΕΣ ΛΕΙΤΟΥΡΓΙΕΣ:

- ΑΝΑΘΕΣΗ ΤΙΜΩΝ – ΕΝΤΟΛΗ ΑΝΤΙΚΑΤΑΣΤΑΣΕΩΣ

```
ΜΕΤΑΒΛΗΤΗ = ΕΚΦΡΑΣΗ
I = 7
PERIFERIAKYKLOY=3.14159+DIAMETROS
ATHRISMA=ATHRISMA+1
```

- INPUT – OUTPUT ΤΙΜΩΝ

```
READ(V1,V2,……)
WRITE(V1,V2,……)
READLN(V1,V2,……)
READLN
WRITELN(V1,V2,……)
WRITELN

EOLN    ΤΕΛΟΣ ΓΡΑΜΜΗ
EOF     ΤΕΛΟΣ ΑΡΧΕΙΟΥ
```


ΒΑΣΙΚΕΣ ΛΕΙΤΟΥΡΓΙΕΣ σε C

```
/*anathesi timwn*/

metavliti = ekfrasi;

I = 7;

periferia_kyklou = 3.14159 *diametros;

athroisma = athroisma +1;

/*isodynama grafoume*/
 athroisma++;


/*Input – Output timwn*/

int a;
int b;
float c;

scanf("%d %d %f", &a, &b, &c); /* anagnwsi apo pliktrologio */

printf("%d  %d %f", a, b, c);  /* ektypwsi sthn othoni */

\n = newline
```


ΑΚΟΛΟΥΘΙΑ ΒΗΜΑΤΩΝ

ΒΗΜΑΤΑ A, B, C,…..ΕΝΟΣ ΑΛΓΟΡΙΘΜΟΥ

BEGIN A, B, C, D END

1. ΤΑ ΒΗΜΑΤΑ ΕΚΤΕΛΟΥΝΤΑΙ ΕΝΑ ΚΑΘΕ ΦΟΡΑ
2. ΚΑΘΕ ΒΗΜΑ ΕΚΤΕΛΕΙΤΑΙ ΜΙΑ ΜΟΝΟ ΦΟΡΑ, ΚΑΝΕΝΑ ΔΕΝ ΠΑΡΑΛΕΙΠΕΤΑΙ
3. Η ΣΕΙΡΑ ΕΚΤΕΛΕΣΗΣ ΕΊΝΑΙ ΑΥΤΉ ΜΕ ΤΗΝ ΟΠΟΙΑ ΕΊΝΑΙ ΓΡΑΜΜΕΝΑ
4. Η ΠΕΡΑΤΩΣΗ ΤΟΥ ΤΕΛΕΥΤΑΙΟΥ ΒΗΜΑΤΟΣ ΣΗΜΑΙΝΕΙ ΠΕΡΑΤΩΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

```
BEGIN
    ΒΡΑΣΕ ΝΕΡΟ,
    ΒΑΛΕ ΚΑΦΕ ΣΤΟ ΦΛΙΤΖΑΝΙ,
    ΠΡΟΣΘΕΣΕ ΝΕΡΟ ΣΤΟ ΦΛΙΤΖΑΝΙ
END
```

ΑΚΟΛΟΥΘΙΑ ΒΗΜΑΤΩΝ στη C

```
/* H akolouthia entolwn A; B; C; ... grafetai ws eksis */

{
  A;
  B;
  C;
}

/* fiaxno kafe */

{
   vrase_nero();
   vale_kafe_sto_flytzani();
    prosthese_nero_sto_flytzani();
}
```


ΕΠΙΛΟΓΗ ΒΗΜΑΤΟΣ-1

- IF Q THEN A ΕΑΝ Q ΤΟΤΕ Α
  ΠΑΡΕ ΤΟ ΒΑΖΟ ΑΠΟ ΤΟ ΡΑΦΙ
  ΕΑΝ ΤΟ ΒΑΖΟ ΕΙΝΑΙ ΑΔΕΙΟ
  ΤΟΤΕ ΠΑΡΕ ΝΕΟ ΒΑΖΟ ΑΠ' ΤΟ ΝΤΟΥΛΑΠΙ
  ΑΝΟΙΞΕ ΤΟ ΚΑΠΑΚΙ

- IF Q THEN A ELSE B ΕΑΝ Q ΤΟΤΕ Α ΑΛΛΙΩΣ Β
  ΕΑΝ ΤΟ ΦΑΝΑΡΙ ΕΙΝΑΙ ΚΟΚΚΙΝΟ ή ΚΙΤΡΙΝΟ
  ΤΟΤΕ ΣΤΑΜΑΤΑ
  ΑΛΛΙΩΣ ΠΕΡΑΣΕ

- IF Q THEN A ELSEIF ... ΕΑΝ Q ΤΟΤΕ Α ΑΛΛΙΩΣ_ΕΑΝ...
  ΕΑΝ ΔΕΝ ΛΕΙΤΟΥΡΓΕΙ ΤΟ ΦΑΝΑΡΙ
  ΤΟΤΕ ΠΡΟΧΩΡΑ ΜΕ ΠΡΟΣΟΧΗ
  ΑΛΛΙΩΣ_ΕΑΝ ΤΟ ΦΑΝΑΡΙ ΕΙΝΑΙ ΚΟΚΚΙΝΟ ή ΚΙΤΡΙΝΟ
  ΤΟΤΕ ΣΤΑΜΑΤΑ
  ΑΛΛΙΩΣ ΠΕΡΑΣΕ

ΕΠΙΛΟΓΗ ΒΗΜΑΤΟΣ-1 στη C

(if then – if then else – if then elseif)

```
/*if Q then A */
If ( Q )
  A;
pare_to_vaso_apo_to_rafi();
if ( to_vaso_einai_adeio() )
    pare_neo_vaso_apo_to_ntoulapi();
anoikse_to_kapaki();

/* if Q then A else B*/
if ( Q )
   A;
else B;

if ( to_fanari_einai_kokkino_i_kitrino() )
  stamata();
else
  perase();
```

```
/* if Q1 then A elseif Q2 then B elseif ...else Z */
if ( Q1 )
   A;
else if ( Q2 )
   B;
 else if (....



if (den_leitourgei_to_fanari())
   proxvra_me_prosohi();
else if (to_fanari_einai_kokkino_i_kitrino())
   stamata();
else
   perase();
```


ΕΠΙΛΟΓΗ ΒΗΜΑΤΟΣ-2

- CASE Q OF ΠΕΡΙΠΤΩΣΗ Q ΑΠΟ
  a: A; a: A;
  b: B; b: B;
  c: C; c: C;
  otherwise: D; αλλιώς: D;
  END ΤΕΛΟΣ

- Π.χ., ΠΕΡΙΠΤΩΣΗ ΗΜΕΡΑ ΑΠΟ
  ΔΕΥΤΕΡΑ : ΒΑΛΕ ΠΛΗΝΤΗΡΙΟ
  ΤΡΙΤΗ : ΒΑΛΕ ΣΙΔΕΡΟ
  ΤΕΤΑΡΤΗ : ΜΠΑΛΟΣΕ & ΡΑΨΕ
  ΠΕΜΠΤΗ : ΠΗΓΑΙΝΕ ΓΙΑ ΨΩΝΙΑ
  ΠΑΡΑΣΚΕΥΗ : ΚΑΘΑΡΙΣΕ ΤΟ ΣΠΙΤΙ
  ΣΑΒΒΑΤΟ : ΦΤΙΑΞΕ ΓΛΥΚΑ
  ΚΥΡΙΑΚΗ : ΞΕΚΟΥΡΑΣΟΥ
  ΤΕΛΟΣ

  ΕΑΝ ΗΜΕΡΑ=ΔΕΥΤΕΡΑ
  ΤΟΤΕ ΒΑΛΕ ΠΛΥΝΤΗΡΙΟ
  ΑΛΛΙΩΣ ΕΑΝ ΗΜΕΡΑ = ΤΡΙΤΗ
  ΤΟΤΕ ΒΑΛΕ ΣΙΔΕΡΟ
  ΑΛΛΙΩΣ ΕΑΝ ΗΜΕΡΑ =.....
  .....

- CASE ≈ IF ..THEN....ELSEIF.....

ΕΠΙΛΟΓΗ ΒΗΜΑΤΟΣ-2 σε C

```
/* case sti glwssa C */

switch ( Q ) {
    case a: A; break;
    case b: B; break;
    case c: C; break;
     .....
    default: default_command;
}


switch (imera) {
      case Deutera: bale_plyntirio(); break;
      case Triti: ..........
      ......
      case Kyriaki: ksekourasou(); break;
}

/*anti tis xrisis if then elseif... */

if (imera == Deutera) bale_plyntirio();
else if (imera == Triti) bale_sidero();
else if ....
......
else ksekourasou();
```


ΕΠΑΝΑΛΗΨΗ Repeat-Until

(αόριστη διάρκεια)

- REPEAT A UNTIL Q ΕΠΑΝΕΛΑΒΕ Α ΕΩΣ ΟΤΟΥ Q

- Π.χ., εύρεση δοθέντος ονόματος σε μια λίστα
  ΠΑΡΕ ΤΟ ΠΡΩΤΟ ΟΝΟΜΑ ΑΠΟ ΤΗ ΛΙΣΤΑ
  ΕΠΑΝΕΛΑΒΕ
  ΕΑΝ ΤΟ ΟΝΟΜΑ ΕΙΝΑΙ ΤΟ ΔΟΘΕΝ
  ΤΟΤΕ ΠΑΡΕ ΤΗ ΔΙΕΥΘΥΝΣΗ ΤΟΥ
  ΑΛΛΙΩΣ ΠΑΡΕ ΤΟ ΕΠΟΜΕΝΟ ΟΝΟΜΑ ΣΤΗ ΛΙΣΤΑ
  ΕΩΣ ΟΤΟΥ ΒΡΕΘΕΙ ΤΟ ΔΟΘΕΝ ΟΝΟΜΑ ή ΕΞΑΝΤΛΗΘΕΙ Η ΛΙΣΤΑ

- Π.χ., εύρεση του πρώτου αριθμού μετά ένα δοθέντα
  ΠΑΡΕ ΤΟΝ ΑΡΧΙΚΟ ΑΡΙΘΜΟ
  ΕΠΑΝΕΛΑΒΕ
  ΠΡΟΣΘΕΣΕ ΤΟΥ 1
  ΔΕΣ ΑΝ ΕΙΝΑΙ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ (ανάλυση)
  ΕΩΣ ΟΤΟΥ ΕΙΝΑΙ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ
  ΓΡΑΨΕ ΤΟΝ ΑΡΙΘΜΟ

  ΒΑΛΕ ΠΙΘΑΝΟ ΠΑΡΑΓΟΝΤΑ ΙΣΟ ΜΕ 2
  ΕΠΑΝΕΛΑΒΕ
  ΔΙΑΙΡΕΣΕ ΤΟΝ ΑΡΙΘΜΟ ΜΕ ΤΟΝ ΠΑΡΑΓΟΝΤΑ
  ΠΡΟΣΘΕΣΕ 1 ΣΤΟΝ ΠΑΡΑΓΟΝΤΑ
  ΕΩΣ ΟΤΟΥ Η ΔΙΑΙΡΕΣΗ ΕΙΝΑΙ ΑΚΡΙΒΗΣ ή Ο ΠΑΡΑΓΟΝΤΑΣ > √ ΑΡΙΘΜΟΥ
  ΕΑΝ Η ΔΙΑΙΡΕΣΗ ΔΕΝ ΕΙΝΑΙ ΑΚΡΙΒΗΣ
  ΤΟΤΕ Ο ΑΡΙΘΜΟΣ ΕΙΝΑΙ ΠΡΩΤΟΣ
  ΑΛΛΙΩΣ ΔΕΝ ΕΙΝΑΙ ΠΡΩΤΟΣ

ΕΠΑΝΑΛΗΨΗ Repeat-Until στη C

(αόριστη διάρκεια)

Do A; while (Q);

```
#include <stdio.h>

/* do command while q einai TRUE */
do
  command;
while (q);
/* PROSOXH: antistoixei sthn REPEAT command until q ginei FALSE */


/* index onomatos se mia lista */
onoma = pare_to_prwto_onoma_apo_th_lista();
do
  if (onoma == to_dothen_onoma)
    take_the_address_of(onoma);
  else
    onoma = epomeno_onoma_sth_lista(name);
while (onoma != to_dothen_onoma && !telos_listas());
//while (!(onoma == to_dothen_onoma) || telos_listas()));
```


ΕΠΑΝΑΛΗΨΗ Repeat-Until στη C

(αόριστη διάρκεια) **Do A; while (Q);**

```
/* prwtos arithmos amesws meta apo dothenta arxiko arithmo */
int main()
{
  int arithmos;
  printf("Dwse arithmo: ");
  scanf("%d", &arithmos);
  do
    arithmos++;
  while (!einai_prwtos_arithmos(arithmos));

  printf("O epomevos prwtos einai o %d\n", arithmos);
  return 0;
}

/* elegxei ean enas arithmos einai prwtos */
int einai_prwtos_arithmos(int arithmos)
{
  int pithanos_paragontas = 2;
  int modulo;
  if (arithmos == 1)
    return 0; /* 0 shmainei oxi prwtos */
  if (arithmos == 2)
    return 1; /* 1 shmainei prwtos */
  do
    modulo = arithmos % pithanos_paragontas++;
  while (modulo && pithanos_paragontas < arithmos);
  //while (!(modulo || pithanos_paragontas >= arithmos));
  return modulo;
}
```


ΕΠΑΝΑΛΗΨΗ While-Do

(αόριστη διάρκεια)

- WHILE Q DO A ΟΣΟ Q ΚΑΝΕ Α
- Π.χ.1, βρές τον μεγαλύτερο αριθμό μιας λίστας

```
ΒΑΛΕ ΣΑΝ ΜΕΓΑΛΥΤΕΡΟ ΤΟΝ ΠΡΩΤΟ ΑΡΙΘΜΟ ΤΗΣ ΛΙΣΤΑΣ
ΕΠΑΝΕΛΑΒΕ
        ΠΑΡΕ ΤΟΝ ΕΠΟΜΕΝΟ ΑΡΙΘΜΟ ΤΗΣ ΛΙΣΤΑΣ
        ΕΑΝ ΑΥΤΟΣ ΕΙΝΑΙ > ΜΕΓΑΛΥΤΕΡΟΥ
                ΤΟΤΕ ΒΑΛΕ ΑΥΤΟΝ ΣΑΝ ΜΕΓΑΛΥΤΕΡΟ
ΕΩΣ ΟΤΟΥ Η ΛΙΣΤΑ ΕΞΑΝΤΛΗΘΕΙ
ΓΡΑΨΕ ΤΟΝ ΜΕΓΑΛΥΤΕΡΟ
```

- Τι εάν η λίστα έχει μόνο ένα όνομα;;

```
ΒΑΛΕ ΣΑΝ ΜΕΓΑΛΥΤΕΡΟ ΤΟΝ ΠΡΩΤΟ ΑΡΙΘΜΟ ΤΗΣ ΛΙΣΤΑΣ
ΟΣΟ Η ΛΙΣΤΑ ΔΕΝ ΕΞΑΝΤΛΗΘΗΚΕ ΚΑΝΕ
        ΠΑΡΕ ΤΟΝ ΕΠΟΜΕΝΟ ΑΡΙΘΜΟ ΑΠΟ ΤΗ ΛΙΣΤΑ
        ΕΑΝ ΑΥΤΟΣ > ΜΕΓΑΛΥΤΕΡΟ
                ΤΟΤΕ ΒΑΛΕ ΑΥΤΟΝ ΣΑΝ ΜΕΓΑΛΥΤΕΡΟ
ΓΡΑΨΕ ΤΟΝ ΜΕΓΑΛΥΤΕΡΟ
```

- Π.χ.1, βρές τον μέγιστο κοινό διαιρέτη δύο αριθμών

```
{ΜΕΓΙΣΤΟΣ ΚΟΙΝΟΣ ΔΙΑΙΡΕΤΗΣ} Μ.Κ.Δ(Χ,Υ)
ΟΣΟ Υ <> 0 ΚΑΝΕ
        ΥΠΟΛΟΓΙΣΕ ΤΟ ΥΠΟΛΟΙΠΟ ΤΟΥ Χ/Υ
        ΑΝΤΙΚΑΤΑΣΤΗΣΕ ΤΟ Χ ΜΕ ΤΟ Υ
        ΑΝΤΙΚΑΤΑΣΤΗΣΕ ΤΟ Υ ΜΕ ΤΟ ΥΠΟΛΟΙΠΟ
ΓΡΑΨΕ ΤΟ Χ ΣΑΝ ΤΟΝ Μ.Κ.Δ
```


ΕΠΑΝΑΛΗΨΗ While-Do στη C

(αόριστη διάρκεια) **while (Q) {A,B,…};**

```
#include <stdio.h>
int lista[] = { 12, 17, 25, 9, 42, 19 };

int main()
{
  int megalytero = lista[0];
  int index = 1;
  do {
    int epomenos_arithmos = lista[index++];
    if (epomenos_arithmos > megalytero)
      megalytero = epomenos_arithmos;
  } while (index < sizeof (lista) / sizeof(int) );
  printf("To megalytero stoixeio einai to d\n", megalytero);
  return 0;
}
```

/* while command */ while (Q) command

```
#include <stdio.h>
int lista[] = { 12, 17, 25, 9, 42, 19 };

int main()
{
  int megalytero = lista[0];
  int index = 1;
  while (index < sizeof (lista) / sizeof(int) ) {
    int epomenos_arithmos = lista[index++];
    if (epomenos_arithmos > megalytero)
      megalytero = epomenos_arithmos;
  }
  printf("To megalytero stoixeio einai to %d\n", megalytero);
return 0;
}
```


ΕΠΑΝΑΛΗΨΗ While-Do στη C

(αόριστη διάρκεια) **while (Q) {A,B,…};**

```
#include <stdio.h>

int main()
{
  int x, y;
  printf("Dwse arithmoys x,y: ");
  scanf("%d %d", &x, &y);

  while (y != 0) {
    int modulo = x % y;
    x = y;
    y = modulo;
  }
  printf(") megistos koinos diaireths einai %d\n", x);
  return 0;
}
```


ΕΠΑΝΑΛΗΨΗ Repeat n times

ορισμένη διάρκεια)

- REPEAT N TIMES Α ΕΠΑΝΕΛΑΒΕ Ν ΦΟΡΕΣ ΤΟ Α
- Π.χ., Ύψωση σε δύναμη

```
{X^N}: ΔΙΑΒΑΣΕ ΤΟ Χ ΚΑΙ ΤΟ Ν
       ΔΥΝΑΜΗ =1
       ΕΠΑΝΕΛΑΒΕ Ν ΦΟΡΕΣ
                ΔΥΝΑΜΗ = ΔΥΝΑΜΗ *Χ
       ΓΡΑΨΕ ΤΗ ΔΥΝΑΜΗ
```

- ισοδύναμα γράφεται ως αόριστη επανάληψη
-

```
ΑΡΙΘΜΟΣ ΕΠΑΝΑΛΗΨΕΩΝ=0
ΟΣΟ ΑΡΙΘΜΟΣΕΠΑΝΑΛΗΨΕΩΝ < Ν ΚΑΝΕ
        {ΣΩΜΑ ΕΠΑΝΑΛΗΨΗΣ}
        ΑΥΞΗΣΕ ΚΑΤΑ 1 ΤΟΝ ΑΡΙΘΜΟΕΠΑΝΑΛΗΨΕΩΝ
```

- Αιώνια επανάληψη

```
REPEAT
    σώμα επανάληψης
FOREVER
```

- Επανάληψη Ορισμένης διάρκειας
  - παίξε το δύο φορές
  - περπάτα για 2 km
  - περίμενε 2 min
- αόριστης διάρκειας
  - παίξε το ώσπου να το μάθεις
  - περπάτα ως το τέλος του δρόμου
  - περίμενε μέχρι να γυρίσω

ΕΠΑΝΑΛΗΨΗ Repeat n times στη C

ορισμένη διάρκεια) For (rep) A;

```
#include <stdio.h>

int main()
{
  int x, n;
  int rep;
  int dynamh = 1;
  printf("Dwse arithmoys x,n: ");
  scanf("%d %d", &x, &n);

  for (rep = 1; rep <= n; rep++)
    dynamh *= x;

  printf("%d ^ %d = %d\n", x, n, dynamh);

  return 0;
}
```

```
#include <stdio.h>

int main()
{
  int x, n;
  int rep;
  int dynamh = 1;
  printf("Dwse arithmoys x,n: ");
  scanf("%d %d", &x, &n);

  rep = 0;

  while (rep < n) {
    dynamh *= x;
    rep++;
  }

  printf("%d ^ %d = %d\n", x, n, dynamh);

  return 0;
}
```


ΦΥΣΣΑΛΟΕΙΔΗΣ ΤΑΞΙΝΟΜΗΣΗ
Bubble Sort

- Ταξινόμησε τα ονόματα μιας λίστας αλφαβητικά
  - **όσο η λίστα δεν είναι ταξινομημένη κάνε**
    - **ξεκίνα από την αρχή της λίστας**
    - **επανέλαβε**
      - **εάν το όνομα έπεται αλφαβητικά του επόμενου στη λίστα**
        - **τότε αντάλλαξέ το με το επόμενο**
      - **θεώρησε το επόμενο όνομα στη λίστα**
    - **έως ότου φθάσεις στο τέλος της λίστας**
- Εάν είναι γνωστό το μήκος της λίστας τότε
  - επανέλαβε
    - ξεκίνα από την αρχή της λίστας
    - επανέλαβε N-1 φορές
      - εάν το όνομα έπεται αλφαβητικά του επόμενου
        - τότε αντάλλαξέ το και σημείωσε ότι έκανες ανταλλαγή
      - θεώρησε το επόμενο όνομα στη λίστα
    - έως ότου δεν γίνει ανταλλαγή


```
/* bubble sort in C ελαφριά φυσαλίδα ανεβαίνει*/
#include <stdio.h>
#include <string.h>

char* lista[] = { "Socrates Antikas", "Fjodor Dostoyevsky",
        "Tom Waits", "William Clinton", "Isaac Newton",
    "Friedrich Nietsche", "Franz Kafka"}
int main()
{
    int i,j;
    int megethos_listas;

    megethos_listas = sizeof(lista) / sizeof(char*);

    for (i = 1; i < megethos_listas; i++) {
         for (j = megethos_listas - 1; j >= i; j--) {
                if (strcmp(lista[j -1], lista[j]) > 0) {
                        char* tem;    /* tem δηλώνεται char* */
                        tem = lista[j-1];
                        lista[j-1] = lista[j];
                        lista[j] = tem;
                } } }
    for (i = 0; i < megethos_listas; i++) printf("%s\n", lista[i]);
    return 0;
}
```


ΦΥΣΣΑΛΟΕΙΔΗΣ ΤΑΞΙΝΟΜΗΣΗ
ελαφριά "φυσαλίδα"

	0	1	2	3	4	5	6	7
0	JC	AE						
1	AE	JC	FD					
2	FD	FD	JC	FK				
3	TW	FK	FK	JC	FN			
4	WC	TW	FN	FN	JC	IN		
5	IN	WC	TW	IN	IN	JC	JC	
6	FN	IN	WC	TW	TW	TW	TW	TW
7	FK	FN	IN	WC	WC	WC	WC	WC


```
/* bubble sort in C, βαριά φυσαλίδα */
#include <stdio.h>
#include <string.h>

char* lista[] = { "Socrates Antikas", "Fjodor Dostoyevsky", "Tom
   Waits","William Clinton", "Isaac Newton", "Friedrich Nietsche", "Franz Kafka"}
int main()
{
    int i,j;
    int megethos_listas;
    int antallagh;
    megethos_listas = sizeof(lista) / sizeof(char*);
    do {
         antallagh = 0;
         for (i = 0; i < megethos_listas - 1; i++)  {
                if (strcmp(lista[i], lista[i+1]) > 0) {
                        char* tem;
                        tem = lista[i];
                        lista[i] = lista[i+1];
                        lista[i+1] = tem;
                        antallagh = 1;
      } }
    } while (antallagh);
    for (i = 0; i < megethos_listas; i++)
         printf("%s\n", lista[i]);
    return 0;
}
```


ΦΥΣΣΑΛΟΕΙΔΗΣ ΤΑΞΙΝΟΜΗΣΗ
Βαριά "φυσαλίδα"

	0	1	2	3	4	5	6
0	JC	AE	AE	AE	AE	AE	AE
1	AE	FD	FD	FD	FD	FD	FD
2	FD	JC	JC	IN	FN	FK	FK
3	TW	TW	IN	FN	FK	FN	FN
4	WC	IN	FN	FK	IN	IN	IN
5	IN	FN	FK	JC	JC	JC	JC
6	FN	FK	TW	TW	TW	TW	TW
7	FK	WC	WC	WC	WC	WC	WC

ΑΝΑΔΡΟΜΗ
(recursion)

- Είναι ο ορισμός ενός υπολογισμού Υ με τη βοήθεια του εαυτού του!
- Π.χ., N! = N*(N-1)!

{ΔΙΑΔΙΚΑΣΙΑ ΠΑΡΑΓΟΝΤΙΚΟ (N)}
ΕΑΝ N =1
ΤΟΤΕ ΑΠΑΝΤΗΣΗ 1
ΑΛΛΙΩΣ
N * ΠΑΡΑΓΟΝΤΙΚΟ (N-1)

ΠΑΡΑΓΟΝΤΙΚΟ (3)
ΕΑΝ 3 = 1
ΤΟΤΕ ...
ΑΛΛΙΩΣ 3 * ΠΑΡΑΓΟΝΤΙΚΟ (2)
ΕΑΝ 2 = 1
ΤΟΤΕ ...
ΑΛΛΙΩΣ 2 * ΠΑΡΑΓΟΝΤΙΚΟ (1)
ΕΑΝ 1 = 1 ΤΟΤΕ ΑΠΑΝΤΗΣΗ 1
ΑΛΛΙΩΣ ...

Παραγοντικό σε C με αναδρομή

```
#include <stdio.h>

int factorial (int n)
{ if (n==1)   return 1;
  else return n * factorial(n-1); }

int main()
{ int n;
  printf("Dwse arithmo: ");
  scanf("%d", &n);

  printf("Factorial of %d is %d\n", n, factorial(n));
  return 0; }
```


ΑΝΑΔΡΟΜΗ 2/2

- ΙΣΟΔΥΝΑΜΙΑ ΜΕ REPEAT; WHILE;

ΑΠΑΝΤΗΣΗ = 1
WHILE N ≥ 1 DO
ΑΠΑΝΤΗΣΗ = ΑΠΑΝΤΗΣΗ * N
N = N − 1
ΓΡΑΨΕ ΑΠΑΝΤΗΣΗ

- Κάθε αναδρομικός υπολογισμός μπορεί να γίνει και με επανάληψη
- Όμως σε μερικές περιπτώσεις αυτό είναι πολύ δύσκολο
- Π.χ., οι Πύργοι του Ανόϊ

Παραγοντικό σε C με επανάληψη

```
#include <stdio.h>
int main()
{  int n, n_, factorial;
   printf("Dwse arithmo: ");
   scanf("%d", &n_);
   n = n_;
   factorial = 1;
   while (n >= 1)
       factorial *= n--;
   printf("Factorial of %d is %d\n", n_, factorial);
   return 0;
}
```


Άλλα παραδείγματα

ΑΝΤΙΣΤΡΟΦΗ "ΛΕΞΗΣ"
ΑΦΑΙΡΕΣΕ ΤΟ ΠΡΩΤΟ ΓΡΑΜΜΑ
ΑΝΤΙΣΤΡΟΦΗ "ΥΠΟΛΟΙΠΗΣ ΛΕΞΗΣ"
ΠΡΟΣΑΡΤΗΣΕ ΤΟ ΓΡΑΜΜΑ ΠΟΥ ΑΦΑΙΡΕΣΕΣ

ΔΙΑΔΙΚΑΣΙΑ ΑΝΤΙΣΤΡΟΦΗ (ΑΚΟΛΟΥΘΙΑΣ)
ΕΑΝ ΑΚΟΛΟΥΘΙΑ ΕΧΕΙ ΕΝΑ ΜΟΝΟ ΓΡΑΜΜΑ
ΤΟΤΕ ΓΡΑΨΤΟ
ΑΛΛΙΩΣ ΑΦΑΙΡΕΣΕ ΤΟ ΠΡΩΤΟ ΓΡΑΜΜΑ
ΑΝΤΙΣΤΡΟΦΗ (ΥΠΟΛΟΙΠΗΣ ΑΚΟΛΟΥΘΙΑΣ)
ΠΡΟΣΑΡΤΗΣΕ ΤΟ ΑΦΑΙΡΕΘΕΝ ΓΡΑΜΜΑ

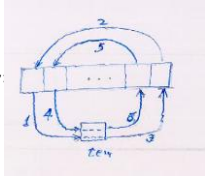
Εφαρμογή: ΑΝΤΙΣΤΡΟΦΗ (ΑΣΤΕΡΑ)

ΔΙΑΔΙΚΑΣΙΑ Μ.Κ.Δ. (Χ, Υ)
ΕΑΝ Υ=0
ΤΟΤΕ ΑΠΑΝΤΗΣΗ ΕΙΝΑΙ ΤΟ Χ
ΑΛΛΙΩΣ ΥΠΟΛΟΓΙΣΕ ΤΟ Μ.Κ.Δ. (Χ, ΥΠΟΛΟΙΠΟ Χ/Υ)

ΙΣΟΔΥΝΑΜΙΑ ΜΕ REPEAT; WHILE;

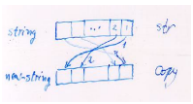
Αντιστροφή string σε C
(με επανάληψη)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXIMUM_SIZE 256
void reverse_string(char* str)
{   int i, j;
    i = 0;
    j = strlen(str) - 1;        /*strlen = μήκος string*/
    while (i < j) {
          char tem = str[i];
          str[i++] = str[j];
          str[j--] = tem;   } }

int main()
{   char buffer[MAXIMUM_SIZE];
    printf("Dwste eva alfariθmntiko gia avtistrofn:"
    fgets(buffer, MAXIMUM_SIZE, stdin);
    buffer[strlen(buffer) - 1] = '\0';
    reverse_string(buffer);
    printf("Reversed string:%s\n", buffer);
    return 0; }
```



Αντιστροφή string σε C
(με αναδρομή)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int length;
char* string = "This is a string";
char new_string[256];
void recursively_reverse_string(char* str, char* copy, int len)
   { if (len == length - 1)
                  copy[0] = str[len];
     else { recursively_reverse_string(str, copy, len + 1);
                  copy[length - len - 1] = str[len]; } }

int main() {
    length = strlen(string);
    new_string[length] = '\0';      /*end of string*/
    printf("Original string: %s\n", string);
    recursively_reverse_string(string, new_string, 0);
    printf("Reversed string: %s\n", new_string);
    return 0;
 }
```



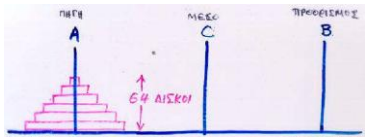
Μέγιστος Κοινός Διαιρέτης σε C
(με αναδρομή)

```
#include <stdio.h>
int n1, n2;
int recursive_gcd(int x, int y) {
  if (x == y)
    return x;
  else
    if (x < y)
      return recursive_gcd(y - x, x);
    else
      if (x > y)
        return recursive_gcd(x - y, y);
}

int main() {
  int g;
  printf("Dwste ari0mo n1 -> "); scanf("%d", &n1);
  printf("Dwste ari0mo n2 -> "); scanf("%d", &n2);
  g = recursive_gcd(n1, n2);
  printf("Greatest Common Divisor of %d and %d is %d\n", n1, n2, g);
  return 0;
}
```


Μέγιστος Κοινός Διαιρέτης σε C
(με επανάληψη)

```
#include <stdio.h>
int n1, n2;
int gcd(int x, int y) {
  int temp;
  while(x != y)
    if (x > y)
      x = x - y;
    else
      y = y - x;
  return x; }

int main() {
  int g;
  printf("Dwste ari0mo n1 -> "); scanf("%d", &n1);
  printf("Dwste ari0mo n2 -> "); scanf("%d", &n2);
  g = gcd(n1, n2);
  printf("Greatest Common Divisor of %d and %d is %d\n", n1, n2, g);
  return 0;
 }
```

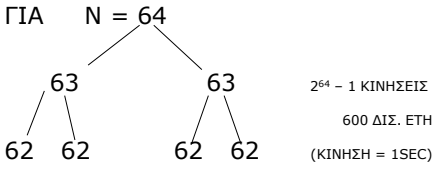

ΠΥΡΓΟΙ ΤΟΥ ΑΝΟΪ (1/2)



ΜΕΤΑΦΟΡΑ 64 ΔΙΣΚΩΝ ΑΠΟ Α → Β
- ΜΕΤΕΦΕΡΕ ΤΟΥΣ 63 ΠΑΝΩ ΔΙΣΚΟΥΣ ΑΠΟ Α→ C
- ΚΙΝΗΣΕ ΤΟΝ ΚΑΤΩ ΔΙΣΚΟ ΑΠΟ Α → Β
- ΜΕΤΕΦΕΡΕ ΤΟΥΣ 63 ΔΙΣΚΟΥΣ ΑΠΟ C → Β

ΔΙΑΔΙΚΑΣΙΑ ΜΕΤΑΦΟΡΑ (N, ΠΗΓΗ, ΠΡΟΟΡΙΣΜΟΣ, ΜΕΣΟ)
- ΕΑΝ N =1
  - ΤΟΤΕ ΚΙΝΗΣΕ ΔΙΣΚΟ ΠΗΓΗ → ΠΡΟΟΡΙΣΜΟ
  - ΑΛΛΙΩΣ ΜΕΤΑΦΟΡΑ (N-1, ΠΗΓΗ, ΜΕΣΟ, ΠΡΟΟΡΙΣΜΟΣ)
    - ΚΙΝΗΣΕ ΕΝΑ ΔΙΣΚΟ ΑΠΟ ΠΗΓΗ → ΠΡΟΟΡΙΣΜΟΣ
    - ΜΕΤΑΦΟΡΑ (N-1, ΜΕΣΟ, ΠΡΟΟΡΙΣΜΟΣ, ΠΗΓΗ)

ΠΥΡΓΟΙ ΤΟΥ ΑΝΟΪ (2/2)

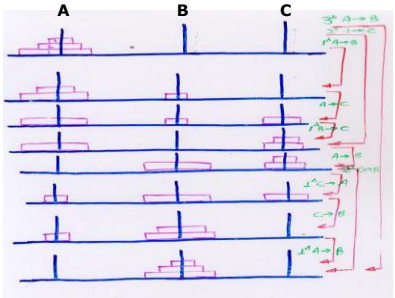


$2^{64} - 1$ ΚΙΝΗΣΕΙΣ

600 ΔΙΣ. ΕΤΗ

(ΚΙΝΗΣΗ = 1SEC)

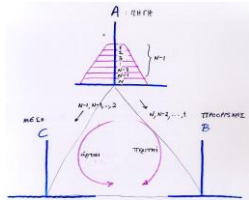
Κινήσεις των ΠΥΡΓΩΝ ΤΟΥ ΑΝΟΪ
(για N=3)



ΠΥΡΓΟΙ ΤΟΥ ΑΝΟΪ σε C

```
#include <stdio.h>
typedef enum  { LEFT, MIDDLE, RIGHT } poleType;
void print_pole(poleType pole)
{   switch (pole) {
    case LEFT: printf("left"); break;
    case MIDDLE: printf("middle"); break;
    case RIGHT: printf("right"); break; } }
void move_a_disk_from_source_to_destination(poleType source, poleType
    destination)
{   printf("Move a disk from "); print_pole(source);
    printf(" to "); print_pole(destination);
    printf("\n"); }
void move(int n, poleType source, poleType auxiliary, poleType destination)
{   if (n == 1) move_a_disk_from_source_to_destination(source, destination);
    else { move(n-1, source, destination, auxiliary);
    move_a_disk_from_source_to_destination(source, destination);
    move(n-1, auxiliary, source, destination); } }
int main()
{   int number_of_disks;
    printf("Dwse arithmo diskwv: ");
    scanf("%d", &number_of_disks);
    printf("For %d disks the required moves are:\n", number_of_disks);
      move(number_of_disks, LEFT, MIDDLE, RIGHT);
    return 0; }
```


ΠΥΡΓΟΙ ΤΟΥ ΑΝΟΪ
(Αρχή - με επανάληψη)



Εάν N περιττός τότε

N	
N-1	A→B
	A→C
	B→C
	A→B
	C→A
	C→B
	A→B

1	A→B
2	A→C
1	B→C
3	A→B
1	C→A
2	C→B
1	A→B
4	

Το μικρότερο πρόγραμμα ΑΝΟΙ (επαναληπτικό)

```
/* Start post is post 0. */
/* If # posts even, then final post is post 1, else is post 2 */

#include <stdio.h>
#include <stdlib.h>

void main(void)
{
 int n, x;
   printf( "How many disks? " );
   scanf( "%d", &n );
   puts( "\n\n" );

   for (x=1; x < (1 << n); x++)
     printf( "move from pole %i to pole %i.\n",  (x&x-1)%3,
       ((x|x-1)+1)%3 );
}
```

ΠΡΟΒΛΗΜΑ: Εξηγήστε το πως και γιατί αυτό το πρόγραμμα δουλεύει!

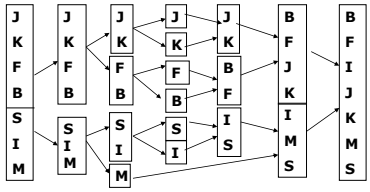
Το μικρότερο πρόγραμμα ΑΝΟΙ
(με εκτύπωση και του αριθμού του μετακινούμενου δίσκου)

```
/* Start post is post 0. */
/* If # posts even, then final post is post 1, else is post 2 */
#include <stdio.h>
#include <stdlib.h>
int  main(void)
{   int n, x, to, fr;
    unsigned int i, j;
   printf( "How many disks? " );
   scanf( "%d", &n );
   puts( "\n\n" );
   for (x=1;  x < (1 << n); x++) {
   i=x&x-1;  fr=(i+i/3)&3;
   i=(x|x-1)+1;  to=(i+i/3)&3;
   for(i=x, j=1;   ; i>>=1, j++) { if(i&1) break; }
   printf( "move disc %i from %i to %i\n", j, fr, to ); } }
```

ΠΡΟΒΛΗΜΑ: Εξηγήστε το πως και γιατί αυτό το πρόγραμμα δουλεύει!

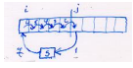
ΤΑΞΙΝΟΜΗΣΗ - ΣΥΓΧΩΝΕΥΣΗ

{module sort (list)}
εάν N>1
 τότε sort (1st half list)
      sort (2nd half list)
      merge (1st half, 2nd half)



Ταξινόμηση με Συγχώνευση σε C 1/2

```
#include <stdio.h>
   #include <string.h>
   #define MAXIMUM_SIZE 256
   void merge(char* lista[], int size1, int size2);
   void sort(char* lista[], int megethos_listas)
   {  if (megethos_listas > 1) {
       int half_size = megethos_listas / 2;
       sort(lista, half_size);
       sort(lista + half_size, megethos_listas - half_size);
       merge(lista, half_size, megethos_listas - half_size);  }}

   void merge(char* lista[], int size1, int size2)
   {  int i = 0;
     int j = size1;
     while ((i < j) && (j < size1 + size2)) {
       if (strcmp(lista[i], lista[j]) > 0) {
         char* s = lista[j];
         int k = j;
         while (k-- > i)
         lista[k + 1] = lista[k];
         lista[i] = s;
         j++;      }
       i++; } }
```



Ταξινόμηση με Συγχώνευση σε C 2/2

```
int main()
{  int i;
   char* lista[MAXIMUM_SIZE];
   int megethos_listas = 0;
   char buffer[MAXIMUM_SIZE];
   int tem;
   /* Eisagwgn dedomevwv apo to xrnstn av eivai epi8ymnto */
   printf("Ta3ivomnsn listas alfari8mntikwv\n");
   printf("--------------------------------\n");
   printf("Dwste ta alfari8mntika eva se ka8e grammn\n");
   printf("(dwste eva kevo gia va termatisete tnv eisagwgn)\n");
   fgets(buffer, MAXIMUM_SIZE, stdin);
   while (tem = strlen(buffer) - 1) {
     lista[megethos_listas] = (char*) malloc(sizeof(char) * (tem + 1));
     strncpy(lista[megethos_listas], buffer, tem);
     *(lista[megethos_listas] + tem) = '\0';
     megethos_listas++;
     fgets(buffer, MAXIMUM_SIZE, stdin); }
     if (!megethos_listas) {lista[0] = "Jesus Christ"; lista[1] = "Albert Einstein";
     lista[2] = "Fjodor Dostoyevsky"; lista[3] = "Tom Waits";lista[4] = "William Clinton";
     lista[5] = "Isaac Newton"; lista[6] = "Friedrich Nietsche";lista[7] = "Franz Kafka";
     megethos_listas = 8; }

   sort(lista, megethos_listas);
   for (i = 0; i < megethos_listas; i++)
     printf("%s\n", lista[i]);
   return 0;
}
```


Στοιχειοποίηση – Modularity

- module <module_name> (<τυπική παράμετρος>)
- module σχεδιασμός_τετραγώνου (μέγεθος)
  χαμήλωσε πέννα
  repeat 4 φορές
  move(μέγεθος)
  αριστερά(90°)
  σήκωσε πέννα
- module σχεδιασμός_πολυγώνου (μέγεθος, N)
  χαμήλωσε πέννα
  repeat N φορές
  move(μέγεθος)
  αριστερά(360/N)
  σήκωσε πέννα
- module σχεδιασμός_τετραγώνου (μέγεθος)
  module σχεδιασμός_πολυγώνου (μέγεθος, 4)
- module σχεδιασμός_τριγώνου (μέγεθος)
  module σχεδιασμός_πολυγώνου (μέγεθος, 3)

Στοιχειοποίηση στη C
Procedures - functions

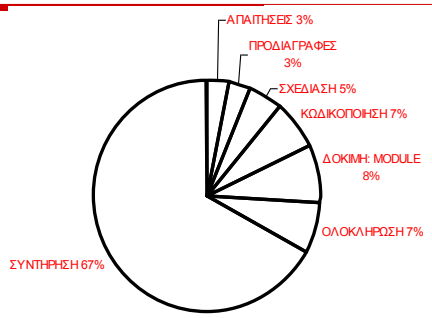
```
void p_name(<param1_type> param1,
      <param2_type> param2, … <paramn_type>, paramn)
{
  …
}

<function_return_type> f_name(<parameter_list>)
{
  …
  return <return_value>;
}
```


ΔΟΜΗΜΕΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ
STRUCTURED PROGRAMMING

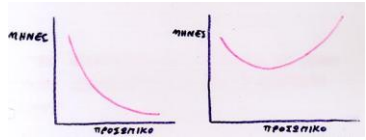
Ορισμοί:

1) Δομημένος προγραμματισμός είναι μια μεθοδολογία οργάνωσης και κωδικοποίησης προγραμμάτων έτσι ώστε να είναι εύκολη
   η κωδικοποίησή τους
   η κατανόησή τους (και από άλλους)
   η διόρθωσή τους (debugging)
   η συντήρησή τους (@ στο απώτερο μέλλον)
2) Δομημένος προγραμματισμός είναι η εφαρμογή μιας βασικής μεθόδου αποσύνθεσης ενός προβλήματος έτσι ώστε να καταλήξει κανείς σε μια ιεραρχική δομή που μπορεί να την χειριστεί κανείς εύκολα.
3) Η βασική έννοια του δομημένου προγραμματισμού είναι μια απόδειξη της ορθότητας του προγράμματος.
4) Δομημένος προγραμματισμός είναι ο σχηματισμός προγραμμάτων σαν ιεραρχικά φωλιασμένες δομές εντολών και υποπρογραμμάτων.

Πραγματική απαιτούμενη προσπάθεια σε όλη τη περίοδο «ζωής» ενός software πακέτου (1/3)



Πραγματική απαιτούμενη προσπάθεια σε όλη τη περίοδο «ζωής» ενός software πακέτου (2/3)



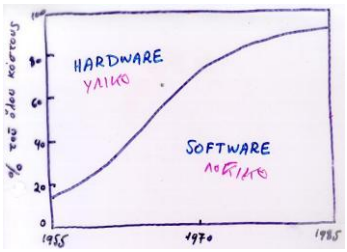
- ☐ ΔΟΥΛΕΙΑ ΜΕ ΜΙΚΡΗ ΑΛΛΗΛΟΕΞΑΡΤΗΣΗ ΤΩΝ ΕΠΙ ΜΕΡΟΥΣ ΤΜΗΜΑΤΩΝ
- ☐ ΔΟΥΛΕΙΑ ΜΕ ΜΕΓΑΛΗ ΑΛΛΗΛΟΕΞΑΡΤΗΣΗ ΤΩΝ ΕΠΙ ΜΕΡΟΥΣ ΤΜΗΜΑΤΩΝ

Πραγματική απαιτούμενη προσπάθεια σε όλη τη περίοδο «ζωής» ενός software πακέτου (3/3)

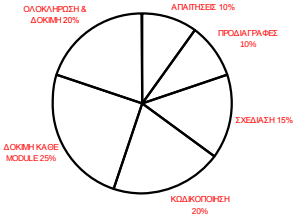
ΝΟΜΟΣ ΤΟΥ BROOKS

Η προσθήκη και άλλου προσωπικού σε ένα αργοπορημένο Software Project το κάνει να αργοπορήσει ακόμα περισσότερο !!!

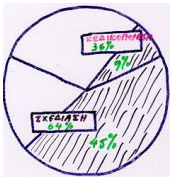
Τάσεις κατανομής του κόστους μεταξύ H/S



Κατανομή της προσπάθειας που χρειάζεται για την ΑΝΑΠΤΥΞΗ κάποιου Software



Πηγές λαθών Software



Λάθη που βρίσκονται κατά τη διάρκεια η μετά τη δοκιμή αποδοχής

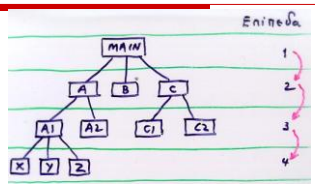
ΟΣΟ ΝΩΡΙΤΕΡΑ ΒΡΙΣΚΟΝΤΑΙ ΤΑ ΛΑΘΗ ΤΟΣΟ ΛΙΓΟΤΕΡΟ ΣΤΟΙΧΙΖΕΙ Η ΔΙΟΡΘΩΣΗ ΤΟΥΣ

ΣΧΕΔΙΑΣΗ ΑΠΟ ΠΑΝΩ ΠΡΟΣ ΤΑ ΚΑΤΩ

- Μέθοδος των Διαδοχικών Βημάτων Προσδιορισμού
  ΚΑΤΑ ΒΗΜΑΤΑ ΔΙΥΛΗΣΗ/ΑΝΑΛΥΣΗ
  Μια ακολουθία αποσυνθέσεων του προβλήματος και διαμέρισης των λειτουργικών προδιαγραφών
  Λειτουργική Δομή
- Αποφάσεις σχετικά με λεπτομέρειες και δομές δεδομένων *αναβάλλονται* όσο είναι δυνατόν για αργότερα
- Σχεδίαση κατά επίπεδα
  - **Κάθε επίπεδο δείχνει μια λήψη απόφασης για τη σχεδίαση και ένα επίπεδο κατανόησης του προγράμματος.**
  - **Ένα επίπεδο λογικά αποτελείται από ένα σύνολο εννοιών που σχετίζονται μεταξύ τους και αντιμετωπίζονται σε αυτό το στάδιο της διαδικασίας αποσύνθεσης.**
- Εγγυάται ορθότητα σε κάθε επίπεδο
- Αρχική ανεξαρτησία από γλώσσα προγραμματισμού
- Ακριβής ορισμός του προβλήματος

Σχεδίαση ΠΑΝΩ → ΚΑΤΩ

TOP-DOWN

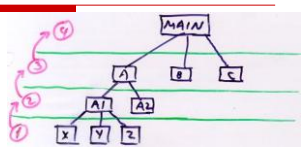


Πλεονεκτήματα
- Ολοκλήρωση γνωρίς (όχι προβλήματα διασύνδεσης)
- Παράλληλοι έλεγχοι
- Ορατότητα, παρακολούθηση
- Συντήρηση (αλλαγές σε υψηλό επίπεδο μεταφέρονται χαμηλότερα όπου χρειάζεται)

Μειονεκτήματα
- Κώδικας ΚΡΙΣΙΜΟΣ μπορεί να δημιουργήσει προβλήματα αργότερα
- Κοινά ή ξαναχρησιμοποιητέα modules (ίδια δουλειά, αλλά διαφορετική διασύνδεση)

ΣΧΕΔΙΑΣΗ ΚΑΤΩ → ΠΑΝΩ
BOTTOM - UP

- Ανάπτυξη ΚΡΙΣΙΜΟΥ ΚΩΔΙΚΑ πρώτα
- Έλεγχος αυτού
- Ανάπτυξη του υπόλοιπου συστήματος



Πλεονεκτήματα:
- Έλεγχος του ΚΡΙΣΙΜΟΥ ΚΩΔΙΚΑ
- Επαναχρησιμοποίηση MODULES

Μειονεκτήματα:
- Ολοκλήρωση, επαλήθευση, διασύνδεση, ορατότητα, παρακολούθηση
- Συντήρηση
- Δομή

ΣΤΟΙΧΕΙΟΠΟΙΗΣΗ Προγράμματος – MODULARIZATION

- Ένα ΣΤΟΙΧΕΙΟ πραγματώνει μία μόνο λογική συνάρτηση ή ένα μικρό αριθμό «συναρτήσεων» που σχετίζονται μεταξύ τους
- Ένα ΣΤΟΙΧΕΙΟ λειτουργεί ανεξάρτητα περιεχομένου
- Τα ΣΤΟΙΧΕΙΑ μπορούν να συνδεθούν μεταξύ τους χωρίς να ξέρει το ένα το άλλο το πώς δουλεύει εσωτερικά (τυποποίηση Διασύνδεσης)

Πλεονεκτήματα Στοιχειοποίησης

- Το ΣΤΟΙΧΕΙΟ είναι μια αρκετά μικρή μονάδα που είναι εύκολα κατανοητή, δίνεται έτσι μεγαλύτερη προσοχή στη λογική σχεδίαση της μονάδας με αποτέλεσμα να έχουμε λιγότερα λάθη
- Πιο εύκολα ελέγχεται – Αξιοπιστία.
- Βιβλιοθήκη ΣΤΟΙΧΕΙΩΝ
- Χρονικός προγραμματισμός του project πιο ακριβής
- Ευκολότερη συντήρηση

Κριτήρια Στοιχειοποίησης

- Μέγεθος ΣΤΟΙΧΕΙΟΥ
- Διαχωρισμός «Ελέγχου» και Υπολογισμών
- Ελαχιστοποίηση στον όγκο πληροφοριών που περνούν από τα σημεία Διασύνδεσης
- Επίτευξη Ιεραρχικής Δομής
- Απομόνωση αλλαγών

Πρακτική για Δομημένο Προγραμματισμό (1/3)

- Υπορουτίνες
  - εξωτερικές: δεν αλλάζουν το περιβάλλον του καλούντα
  - εσωτερικές
- Καταχωρητές Διασύνδεσης (τυποποίηση)
- Πέρασμα Παραμέτρων (Λίστα παραμέτρων)
- Είσοδος (σε εξωτερική υπορουτίνα)
  - αποκατάσταση όλων των καταχωρητών
- Έξοδος: μια μόνο έξοδος (στο τέλος του κώδικα)

Πρακτική για Δομημένο Προγραμματισμό (2/3)

- Εξωτερική τεκμηρίωση (αντανακλά τη Σχεδίαση του προγράμματος)
  - **Οδηγός χρήστη** — ΤΙ ΚΑΝΕΙ / ΔΙΑΓΝΩΣΤΙΚΑ
  - **Περιγραφή προγράμματος**
    - Περιγραφή σχεδίασης (δομή – διάγραμμα βάσης)
    - Προδιαγραφές Στοιχείων
      ΟΝΟΜΑ
      ΣΥΝΑΡΤΗΣΗ
      ΠΙΝΑΚΑ ΚΛΗΣΕΩΝ (CROSS-REFERENCE usr)
      ΠΑΡΑΜΕΤΡΟΙ
      ΧΡΗΣΗ ΜΝΗΜΗΣ
      I/O
      ΚΑΝΟΝΙΚΗ ΕΞΟΔΟΣ
      ΣΧΟΛΙΑ
      ΣΥΝΘΗΚΕΣ ΛΑΘΩΝ
      ΠΕΡΙΓΡΑΦΗ ΣΕ ΨΕΥΔΟΓΛΩΣΣΑ

Πρακτική για Δομημένο Προγραμματισμό (3/3)

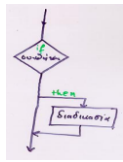
- Εσωτερική τεκμηρίωση (αντανακλά τη κωδικοποίηση του προγράμματος)
  - **Σχόλιο – Επικεφαλίδα**
  - **Σχόλια Τμημάτων**
  - **Σχόλια για τη χρήση καταχωρητών**
  - **Μορφή προγράμματος (ευθυγράμμιση, κενά κλπ.)**
  - **Σχόλια εντολών – ΨΕΥΔΟΓΛΩΣΣΑ**
  - **Ροή προγράμματος**

- Δομικοί λίθοι Δομημένου Προγραμματισμού για κατά βήματα διύλιση

ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

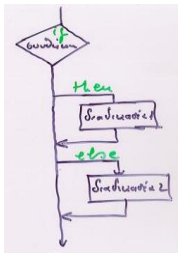
1) begin – end
```
begin
<διαδικασία>
end
```

2) if – then
```
if <συνθήκη> then
   <διαδικασία>
endif
```



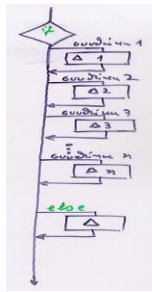
ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

3) if – then – else
```
if <συνθήκη> then
   <διαδικασία 1>
else
   <διαδικασία 2>
endif
```



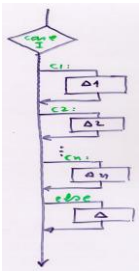
ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

4) if – then – elseif
```
if <συνθήκη 1> then
   <διαδικασία 1>
elseif <συνθήκη 2> then
   <διαδικασία 2>
 elseif <συνθήκη 3> then
   <διαδικασία 3>
...
 elseif <συνθήκη n> then
   <διαδικασία n>
else
   <διαδικασία>
endif
```



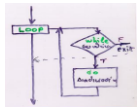
ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

5) case
```
case I
c1 : <διαδικασία 1>
c2 : <διαδικασία 2>

cn : <διαδικασία n>
else : <διαδικασία>
endcase
```

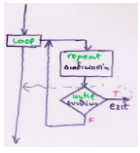


ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

6) while – do
```
while <συνθήκη> do
   <διαδικασία>
endwhile
```



7) repeat – until
```
repeat
   <διαδικασία>
until <συνθήκη>
```



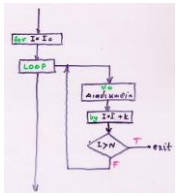
ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

8) for $I=I_0$ to N by k do

for $I=I_0$ to N by k do
<διαδικασία>
endfor

⇔

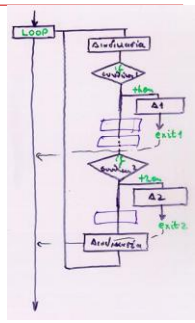
$I=I_0$
repeat
<διαδικασία>
I=I+k
until I>N



ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

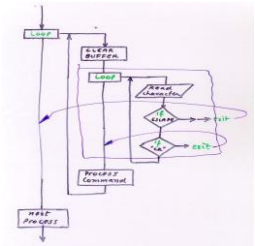
9) Γενικό LOOP – πολλαπλές EXITS

```
loop
   begin
      <διαδικασία>
   end
   if <συνθήκη 1> then
      <διαδικασία 1> exit1
   endif
   ...
   if <συνθήκη 2> then
      <διαδικασία 2> exit2
   endif
   begin
      <διαδικασία>
   end
endloop
```



ΔΟΜΙΚΟΙ ΛΙΘΟΙ Δομημένου Προγραμματισμού

10) Φωλιασμένες δομές – exits πολλών επιπέδων



Παραλληλία (Parallelism)

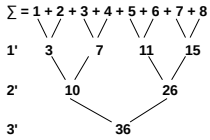
- Παραλληλία, ταυτοχρονισμός
- Παράλληλος αλγόριθμος - σειριακός αλγόριθμος
- Παράδειγμα 1: υπολογισμός κέρδους
  - **module σειριακό_Κέρδος(Λ_Α, Λ_Π)** **{ένας επεξεργαστής, Χρόνος: O(N) }**
    {υπολόγισε λίστα κέρδους από λίστες αγοράς & πώλησης }
    ξεκίνα από την κορυφή κάθε λίστας
    repear N φορές
    - **αφαίρεσε τιμή αγοράς από τιμή πώλησης**
    - **βάλε την απάντηση στη λίστα κέρδους**
    - **πάρε τις επόμενες τιμές από τις λίστες**
  - **module παράλληλο_Κέρδος(i)** **{ i-οστός επεξεργαστής , Χρόνος: O(1) }**
    {N επεξεργαστές εκτελούν παράλληλα το ίδιο πρόγραμμα}
    { υπολόγισε το i-οστό κέρδος }
    αφαίρεσε την i-οστή τιμή αγοράς από αυτή της πώλησης
    βάλε την απάντηση στην i-οστή θέση της λίστας κέρδους
- Speed-up = O(N)

Παραλληλία (Parallelism)

- Παράδειγμα 2: Άθροιση αριθμών
  - **module σειρακό_Αθροισμα(N)** **{ένας επεξεργαστής, χρόνος = O(N)**
    { πρόσθεσε μια σειρά N αριθμών }
    άθροισμα = 0
    πάρε τον πρώτο αριθμό στη σειρά
    repeat N φορές
    άθροισμα = άθροισμα + αριθμός
    πάρε τον επόμενο αριθμό στη σειρά
  - **module παράλληλο_Αθροισμα(i)**
    { αλγόριθμος πρόσθεσης i επεξεργαστή }
    repeat $\log_2 N$ φορές
    - **θέσε num(i) = num(2i-1)+num(2i)**
  - **Χρόνος = O($\log_2$N)**

num(i) 1 2 3 4 5 6 7 8

1' 3 7 11 15
2' 10 26
3' 36



  - **Speed-up = N / $\log_2$N**

Παραλληλία (Parallelism)

- Παράδειγμα 3: {ταξινόμηση λίστας N ονομάτων}

module σειριακή_ταξινόμιση(λίστα) (ένας επεξεργαστής, χρόνος = O(N^2)
repeat
πάρε το άνομα στη κορυφή της λίστας
repeat N-1 φορές
if όνομα ακολουθεί αλφαβητικά το επόμενο όνομα στη λίστα
τότε αντάλλαξε τα
θεώρησε το επόμενο όνομα στη λίστα
until δεν γίνει ανταλλαγή

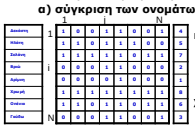
module αναδρομική_ταξινόμιση(λίστα) {N ονόματα} (ένας επεξεργαστής)
εάν N>1
τότε αναδρομική ταξινόμηση (1ο ήμισυ λίστας)
αναδρομική ταξινόμηση (2ο ήμισυ λίστας)
συγχώνευσε τα δύο ήμισυ

Χρόνος: T(N)= 2 T(N/2) + c N ⇒ T(N) = cNlogN + kN
⇒ O(Nlog$_2$N)

Παραλληλία (Parallelism)

- Παράλληλη ταξινόμηση

α) σύγκριση των ονομάτων ανά δυο χρησιμοποιώντας N^2 επεξεργαστές



```
module παραλληλη_σύγκριση(i,j)
  if όνομα(i) προηγείται του όνομα(j)
       then βάλε 0 στην θέση (i,j) του πίνακα
       else βάλε 1
```

χρόνος O(1)

β) module παράλληλο_Αθροισμα(i)
{πρόσθεσε παράλληλα τα '1' κάθε γραμμής χρησιμοποιώντας N/2 επεξεργαστές / γραμμή, δηλαδή συνολικά N N/2 επεξεργαστές}
χρόνος O($\log_2$N)

γ) μετακίνησε παράλληλα κάθε όνομα στην σωστή του θέση χρησιμοποιώντας N επεξεργαστές
χρόνος O(1)

➢ **Συνολικά χρόνος O($\log_2$N)**

Παραλληλία (Parallelism)

- Σύγκριση των αλγορίθμων ταξινόμησης

N Μέγεθος Λίστας	O(N^2) Σειριακή Ταξινόμηση	O(N $\log_2$N) Με αναδρομή	O($\log_2$N) Παράλληλη Ταξινόμηση
10	0,0001 sec	0,00003 sec	0,000003 sec
100	0,01 sec	0,0007 sec	0,000007 sec
1.000	1 sec	0,01 sec	0,00001 sec
10.000	1,7 min	0,13 sec	0,000013 sec
100.000	2,8 hours	1,7 sec	0,000017 sec

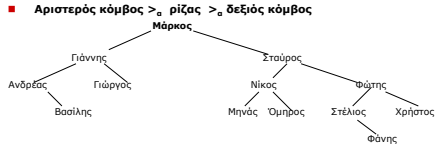
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

- Αφορά τη μορφοποίηση των δεδομένων ενός προγράμματος
- Βασικές δομές:
  - **Στοιχειοσειρά (array) {τύπος όνομα[μέγεθος]}**
  - **Ουρά (queue) { push >** **> pull**
  - **Στοίβα (stack) { get< >put**
  - **Σωρός (heap) {**
  - **Λίστα (list) {head, tail, next**
  - **Γράφος (graph) {G=(V,E)**
  - **Δένδρο (tree) { T=(root, L, R)**

ΔΥΑΔΙΚΟ ΔΕΝΔΡΟ

(Binary tree)

- Από κάθε κόμβο του ξεκινούν δύο το πολύ κλαδιά
- Tree= (root, L, R) = (ρίζα, α-κλαδί, δ-κλαδί)
- Πολύ χρήσιμη δομή
- Π.Χ., Ταξινομημένο δυαδικό δένδρο: για κάθε διακλάδωση ισχύει
  - **Αριστερός κόμβος >$_a$ ρίζας >$_a$ δεξιός κόμβος**



  - **Η ταξινόμηση με τη βοήθεια δυαδικού δένδρου έχει τις φάσεις:**
    - Κατασκευή ταξινομημένου δυαδικού δένδρου με είσοδο τα προς ταξινόμηση δεδομένα
    - Εκτύπωση του ταξινομημένου δυαδικού δένδρου

Ταξινόμηση με δυαδικό δένδρο

- Κτίσιμο δένδρου:

```
module buildtree(Λίστα,Tree)
while Lista δεν έχει εξαντληθεί do
    πάρε το επόμενο_όνομα από τη Λίστα
    addname(επόμενο_όνομα, Tree)

module addname(name, T)
if T άδειο
    then δημιούργησε T με name=root του T
    else if name >a root του T
        then addname(name, L_subtree_of_T)
        else addname(name, R_subtree_of_T)
```

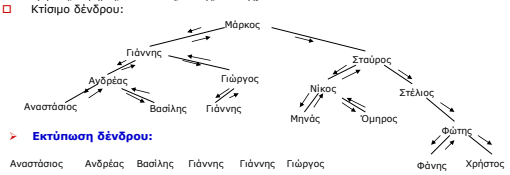
- Εκτύπωση δένδρου

```
module output_tree(T)
if T δεν είναι άδειο
    then output_tree( L_subtree_of_T)
         write root of T
         output_tree(R_subtree_of_T)
```

- Άρα έχουμε: module sort (L)
  build(L,T)
  output_tree(T)

Παράδειγμα

- Λίστα προς ταξινόμηση:
  {Μάρκος, Σταύρος, Νίκος, Γιάννης, Γιώργος, Μηνάς, Ανδρέας, Στέλιος, Φώτης, Βασίλης, Χρήστος, Όμηρος, Αναστάσιος, Φάνης, Γιάννης}
- Κτίσιμο δένδρου:



➢ **Εκτύπωση δένδρου:**

Αναστάσιος Ανδρέας Βασίλης Γιάννης Γιάννης Γιώργος
Μάρκος Μηνάς Νίκος Όμηρος Σταύρος Στέλιος Φάνης
Φώτης Χρήστος