



ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 2: Εισαγωγή -
Μορφοποίηση, Εκφράσεις,
Επιλογή, Επανάληψη
(Κεφάλαια 3-4-5-6, ΚΝΚ-2ΕΔ)

Δημήτρης Ζεϊναλιπούρ

<http://www.cs.ucy.ac.cy/courses/EPL232>

Εισαγωγική Επισήμανση



- Σε αυτή τη διάλεξη θα έχουμε τη δυνατότητα να δούμε πως υλοποιούνται **γνωστές εντολές** (if, for, while, switch, κτλ...) στην γλώσσα C
 - Ευτυχώς, **δεν υπάρχουν πολλές διαφορές** με την JAVA, άρα δε θα χρειαστεί να δώσουμε όλες τις **εισαγωγικές επεξηγήσεις**.
 - Παρακαλώ δείτε το υλικό του ΕΠΛ131, εάν χρειάζεται.
- Θα **δώσουμε** ωστόσο έμφαση στην ακριβή **σύνταξη, κοινά λάθη και παραλήψεις** όσο και σε **νέους τελεστές** που δεν έχετε διδαχθεί.
 - Δε θα δούμε πολλά παραδείγματα, αλλά θα **εμπεδώσετε τις έννοιες μέσω της Άσκησης 1**.

Περιεχόμενο Διάλεξης 2



- **Μορφοποίηση I/O – Formatted I/O (Κεφ. 3)**
 - printf, scanf, ορίσματα
- **Εκφράσεις – Expressions (Κεφ. 4)**
 - Αριθμητικοί Τελεστές, Τελεστές Ανάθεσης, Μοναδιαίοι & Σχισιακοί Τελεστές, Λογικοί Τελεστές
- **Εντολές Επιλογής – Selection (Κεφ. 5)**
 - If-then-else, macros, switch
- **Εντολές Επανάληψης – Repetition (Κεφ. 6)**
 - while, do, for, break, continue, goto

Μορφοποίηση Εξόδου της `printf`



- Η `printf` μας επιτρέπει να εκτυπώσουμε στην έξοδο, π.χ., `printf("Age: %d", 32);`
- **Προδιαγραφή Μετατροπής (Conversion Specification)**

- Όπου ***m*** (*minimum field width*) και ***p*** (*precision*) είναι σταθερές και ***X*** (*conversion specifier*) είναι γράμμα
- **`%m.pX`**: δεξιά στοίχιση, π.χ., (`"%5.3d", 40`) => `|__ _040|`
- **`%-m.pX`**: αριστερή, π.χ., (`"%-5.3d", 40`) => `|040__ _|`

Νόμιμες Δηλώσεις: ***m.p*** ή ***m*** ή ***.p*** και για ***X*** έχουμε:

- ***d*** (*decimal*, π.χ., 0-9) ***o*** (*octal* 0-7) και ***x*** (*hexadecimal* 0-F)
 - ***i*** (*integer*): Σε `scanf` σαρώνει σε ***d*** (**56**), ***o*** (**056**) ή ***x*** (**0x56**)
- ***e*** (*exponential*, π.χ., 8.392e+02) **Εκθετική**
- ***f*** (*float in fixed decimal*, π.χ., 8.7) **Πραγματική**
- ***g*** (*exponential | fixed decimal ανάλογα με το μέγεθος*).

Ακολουθίες Escape



- Οι ακολουθίες **Escape** επιτρέπουν σε συμβολοσειρές να περιέχουν **μη-εκτυπώσιμους χαρακτήρες (ελέγχου)**

- Alert (bell) \a
- Backspace \b
- New line \n
- Horizontal tab \t

ή χαρακτήρες με ειδικό νόημα

- \" : `printf(\"\\\"Hello!\\\"");` => "Hello!"
- \\ : `printf(\"\\\\\");` => \

Η Συνάρτηση `scanf` (Λογική Σάρωσης)



- Είχαμε πει ότι η συνάρτηση `scanf` διαβάζει δεδομένα από το Standard Input με ένα συγκεκριμένο (προσδιορισμένο) `format`, π.χ.,

```
- int i, j;          float x, y;  
- scanf("%d%d%f%f", &i, &j, &x, &y);
```

Η χρήση του `&` θα εξηγηθεί στη διάλεξη 5.
Μέχρι τότε να δίνεται σε όλες τις μεταβλητές

- Ουσιαστικά, η `scanf` επιχειρεί να βρει τους τύπους αγνοώντας τα **white-space χαρακτήρες** (`\n` newline, `\t` horizontal tab, `\v` vertical tab και form-feed `\f`.)

Τα `\v` και `\t` είχαν χρήση παλιά σε εκτυπωτές αλλά όχι σήμερα.

- Παράδειγμα με Δεδομένα σε Πολλαπλές γραμμές όπου `x` το `\n`:

```
1  
-20 .3  
-4.0e3
```

```
••1x-20•••.3x•••-4.0e3x  
ssrsrrrsssrrssssrrrrrr (s = skipped; r = read)
```

- Η `scanf` δεν διαβάζει το τελευταίο `\n`, άρα παραμένει στο input buffer για την επόμενη εντολή που διαβάζει από το stdin

Η Συνάρτηση scanf (Λογική Σάρωσης)



- **Ακέραιος (%d):** διαβάζει ένα-ένα χαρακτήρα μέχρι την εύρεση χαρακτήρα ψηφίου (0-9) ή χαρακτήρα προσήμου (+,-). Στη συνέχεια διαβάζει από το stdin μέχρι να βρει ένα χαρακτήρα που δεν είναι ψηφίο.
 - π.χ., abc: **+3432** ac
- **Συμβολοσειρά (%s):** διαβάζει ένα-ένα χαρακτήρα μέχρι την εύρεση του χαρακτήρα NUL (\0).
- **Αριθμός Κινητής Υποδιαστολής (%f %e %g):**
 - Χαρακτήρας προσήμου (προαιρετικός), μετά
 - Ψηφία (ενδεχομένως με δεκαδική ακρίβεια), μετά
 - Έκθετης (προαιρετικό): Χαρακτήρας e (ή E), προαιρετικό πρόσημο και 1 ή περισσότερα ψηφία
 - Π.χ., **+8.392e+02**
- **Σημείωση:** Πολλοί προγραμματιστές προτιμούν να σαρώνουν την είσοδο ως συμβολοσειρά και στη συνέχεια να κάνουν τις ανάλογες μετατροπές (για καλύτερο έλεγχο και ασφάλεια)

Η Συνάρτηση `scanf` (Λογική Σάρωσης)



- **Παράδειγμα 1:**

```
scanf ("%d%d%f%f", &i, &j, &x, &y);
```

Είσοδος: 1-20.3-4.0e3x

Πιο κάτω δείχνουμε τι θα έκανε η `scanf`:

- `%d`: Αποθηκεύει 1 στο `i` και βάζει το `-` πίσω στο `stdin`.
- `%d`: Αποθηκεύει -20 στο `j` και βάζει το `.` πίσω στο `stdin`.
- `%f`: Αποθηκεύει 0.3 στο `x` και βάζει το `-` πίσω.
- `%f`: Αποθηκεύει -4.0×10^3 στο `y` και βάζει το `\n` πίσω.

- **Παράδειγμα 2:** `scanf ("%d%d", &i, &j);`

Είσοδος: 2, 3 // αγνοούνται μόνο τα `whitespace`!

Πιο κάτω δείχνουμε τι θα έκανε η `scanf`:

- `i=2` και «`, 3`» μένει στο `stdin` για την επόμενη κλήση (το `","` δεν είναι `whitespace`).

Η Συνάρτηση scanf (Λογική Σάρωσης)



- Προβληματική Κλήση 1: `scanf("Hello: %d", &i);`

(Input) Something 34 => Δεν σαρώνει το 34 ☹

(Input) Hello: 34 => Σαρώνει το 34 στο i 😊

- Προβληματική Κλήση 2: `scanf("%d\n", &i);`

Δηλώσαμε ότι θέλουμε να σαρώσουμε το πρότυπο "%d\n" άρα για επιτυχή τερματισμό του προγράμματος πρέπει να δώσουμε: α) <ακέραιο>\n, ή β) τουλάχιστο ένα χαρακτήρα και ένα \n (για διακοπή).

- Προβληματική Κλήση 3: `#define MAX 50 char name[MAX]; scanf("%s", name); printf("%s", name);`

Δεν υπάρχει & μπροστά στο name

(Input) Hello34 => Σαρώνει το "Hello34\0" στο name 😊

(Input) Hello 34 => Σαρώνει το "Hello\0" στο name ☹

Για σάρωση λέξεων που χωρίζονται με space έχουμε δυο επιλογές:

- `fgets(name, sizeof(name), stdin);`
- `scanf("%" MAX "[^\n]", name);`

Περιεχόμενο Διάλεξης



- **Μορφοποιημένη Είσοδος/Εξοδ. (Κεφ. 3)**
 - printf, scanf, ορίσματα
- **Εκφράσεις & Τελεστές (Κεφ. 4)**
 - Αριθμητικοί Τελεστές, Τελεστές Ανάθεσης, Μοναδιαίοι & Σχισιακοί Τελεστές, Λογικοί Τελεστές
- **Εντολές Επιλογής (Κεφ. 5)**
 - If-then-else, macros, switch
- **Εντολές Επανάληψης (Κεφ. 6)**
 - while, do, for, break, continue, goto

Εκφράσεις & Τελεστές (Expressions & Operators)



- Ορισμοί
 - **Έκφραση (Expression):** εξίσωση η οποία δείχνει πως θα υπολογιστεί μια τιμή (π.χ., $b + c$)
 - **Τελεστής (Operator):** πράξη (π.χ., $+$)
 - **Τελεσταίοι (Operands):** Όροι μιας πράξης (π.χ., b, c)
- Η C έχει όλους τους **τελεστές** που είδατε και στα πλαίσια της JAVA, συμπεριλαμβανομένων:
 - Αριθμητικοί τελεστές: $+$, $-$, $*$, $/$, $\%$
 - Τελεστές ανάθεσης, π.χ., $a = b$
 - Τελεστές αύξησης / μείωσης, π.χ., $a++$, $a--$
 - Λογικοί τελεστές, π.χ., $a > 3 \ \&\& \ b < 1$
 - κτλ.

Εταιρικότητα Τελεστών (Operator Associativity)



- Το θέμα της **Εταιρικότητα (Associativity)** προκύπτει όταν έχουμε δυο ή περισσότερους τελεστές με **ΙΣΗ** προτεραιότητα (χωρίς παρενθέσεις!) π.χ., $i - j - k$
- **Αριστερή Εταιρικότητα (left associative):** εάν ομαδοποιεί τους τελεσταίους από αριστερά στα δεξιά.
 - Π.χ., οι **δυναμικοί αριθμητικοί** τελεστές ($*$, $/$, $\%$, $+$, and $-$)
 - $i - j - k$ ισοδύναμο με $(i - j) - k$
 - $i * j / k$ ισοδύναμο με $(i * j) / k$
- **Δεξιά Εταιρικότητα (right associative):** εάν ομαδοποιεί τους τελεσταίους από δεξιά στα αριστερά
 - Π.χ., οι **μοναδιαίοι αριθμητικοί** τελεστές ($+$ και $-$)
 - $- + i$ ισοδύναμο με $-(+i)$
 - $i + j / k$ ισοδύναμο με $i + (j / k)$
- Ανεξάρτητα από Εταιρικότητα, η C δεν ορίζει τη **σειρά εκτέλεσης των όρων** μιας έκφρασης, π.χ., $(a + b) * (c - d)$

**Χρησιμοποιείτε
Παρενθέσεις!**

Τελεστής Ανάθεσης (Assignment Operator)



Παραδείγματα

- `int i; i = 72.99f; /* i is now 72 */`
- `i = j = k = 0; /* μαζική αρχικοποίηση */`
- **Ο τελεστής ανάθεσης έχει δεξιά Εταιρικότητα:**
`i=(j=(k=0)); /* μαζική αρχικοποίηση */`
`int i; float f; f = i = 33.3f; // το i έχει τιμή
33 και το f 33.0`
- **Σύνθετη Ανάθεση (Compound) έχει δεξιά Εταιρικότητα:**

<code>+=</code>	<code>--</code>	<code>*=</code>	<code>/=</code>	<code>%=</code>
<code>i += 2; /* ίδιο με i = i + 2; */</code>				
<code>i =+ 2; /* Προσοχή: ίδιο i = (+2); */</code>				

Τελεστής Αύξησης/Μείωσης (Increment/Decrement Operator)



- Ο τελεστής **αύξησης (++)** / **μείωσης (--)** μπορεί να χρησιμοποιηθεί ως **προθεματικός (prefix)** ή **επιθεματικός (postfix)** τελεστής ΚΑΙ επηρεάζει την τιμή μιας μεταβλητής

- Παράδειγμα:

Άμεση αύξηση i: Πρώτα αύξηση μετά εκτύπωση

```
i = 1;  
printf("i is %d\n", ++i); /* prints "i is 2" */  
printf("i is %d\n", i); /* prints "i is 2" */
```

Πρώτα εκτύπωση μετά αύξηση

```
i = 1;  
printf("i is %d\n", i++); /* prints "i is 1" */  
printf("i is %d\n", i); /* prints "i is 2" */
```

Αποτίμηση Αριθμητικών Εκφράσεων (Arithmetic Expression Evaluation)



- Υπάρχουν **ασαφείς (ambiguous)** εκφράσεις στη C που οδηγούν σε απροσδιόριστη συμπεριφορά (***undefined behavior***) ενός προγράμματος.
- Δηλ., διαφορετικοί μεταγλωττιστές => διαφορετική συμπτ.
 $a = 5;$
 $c = (b = a + 2) - (a = 1);$ Αρισ. Εταιρ Έκφρασης
 - Εάν εκτελεστεί πρώτα $(b = a + 2)$ τότε $b=7$, $a=1$ και **$c=6$ (GCC)**
 - Εάν εκτελεστεί πρώτα το $(a = 1)$ τότε $a=1$, $b=3$ και **$c=2$**
 - Κάποιοι μεταγλωττιστές μπορεί να δώσουν προειδοποίηση:
“operation on ‘a’ may be undefined” ή **“warning: unsequenced modification and access to ‘a’ [-Wunsequenced]”**
- **Συμπέρασμα:**
 - Χρησιμοποιείτε ΠΑΝΤΑ παρενθέσεις σε εκφράσεις
 - **Αποφεύγετε ΠΑΝΤΑ έντεχνες εκφράσεις** που μπορεί να οδηγήσουν σε απροσδιόριστη συμπεριφορά

Αποτίμηση Λογικών Εκφράσεων (Logical Expression Evaluation)



- **Λογική Έκφραση στη C:** Έκφραση που αποτιμάται σε 0 (false) ή 1 (true)
 - π.χ., $(a > 3)$ ή $(3 < a \ \&\& \ a < 5)$
- Σε πολλές γλώσσες (όπως η JAVA), μια λογική έκφραση παράγει ένα “**Boolean**” ή “**λογικό**” τύπο (π.χ., βασικός τύπος `boolean` ή τύπος κλάσης `Boolean`).
 - Στη C99 υπάρχει όπως είδαμε ο `_Bool` τύπος που είναι και αυτός ουσιαστικά μια ακέραια τιμή.

Boolean Τιμές στη C (και C99)



- Για να γίνουν τα προγράμματα πιο κατανοητά, οι προγραμματιστές παραδοσιακά όριζαν macros με ονόματα όπως TRUE και FALSE:

```
#define TRUE 1
#define FALSE 0
```

```
if (flag == TRUE)   ή  if (flag)
```

Προσοχή:
Zero: 0 FALSE
Positives: ≥ 1 TRUE
Negatives: ≤ -1 TRUE

- Η C99 παρέχει τον τύπο `_Bool` (στην πράξη απλά ένας ακέραιος ο οποίος λαμβάνει μόνο τις τιμές 0, 1).
- Εάν περιλάβουμε την `<stdbool.h>` τότε μπορούμε να χρησιμοποιήσουμε τα macros `true` και `false` αλλά και τις δηλώσεις: `bool flag;`

Αποτίμηση Λογικών Εκφράσεων (Logical Expression Evaluation)



- **Σχισιακοί Τελεστές (Relational Operators)**

< less than
> greater than
<= less than or equal to
>= greater than or equal to
== equal to
!= not equal to

Προσοχή:
Διαφορετικό από & και | που είναι οι
λεγόμενοι δυαδικοί τελεστές που θα
δούμε προς το τέλος του μαθήματος.

- **Τελεστές Ισότητας (Equality Operators)**

== equal to
!= not equal to

- **Λογικοί Τελεστές (Logical Operators)**

! logical negation (μοναδιαίος τελεστής, δεξιά εταιρικότητα)
&& logical *and* (δυαδικός τελεστής, αριστερή εταιρικότητα)
|| logical *or* (δυαδικός τελεστής, αριστερή εταιρικότητα)



Αποτίμηση Λογικών Εκφράσεων (Logical Expression Evaluation)



Παραδείγματα

- **Προτεραιότητα λογικών τελεστών μικρότερη από τους αριθμητικούς τελεστές:**
π.χ., $i + j < k - 1$ σημαίνει $(i + j) < (k - 1)$
- **Η Εταιρικότητα των σχεσιακών τελεσ. είναι αριστερή:**
π.χ., $i < j < k$ σημαίνει $(i < j) < k$
 - Εάν η πρόθεση μας ήταν να ελέγξουμε ότι j μεταξύ i και k έπρεπε να γράψουμε: $i < j \ \&\& \ j < k$.
- Τόσο το $\&\&$ όσο και το $||$ **“βραχυκυκλώνουν” (“short-circuit”)** μια αποτίμηση: αποτιμούν το αριστερότερο σκέλος και μετά τα δεξιότερα (εάν χρειάζεται).
 - $(i \neq 0) \ \&\& \ ((j / i) > 0) \ //$ εάν δεν ισχύει το πρώτο σκέλος της έκφρασης δεν εκτελείται το δεξιό

Περιεχόμενο Διάλεξης



- **Μορφοποιημένη Είσοδος/Έξοδ. (Κεφ. 3)**
 - printf, scanf, ορίσματα
- **Εκφράσεις (Κεφ. 4)**
 - Αριθμητικοί Τελεστές, Τελεστές Ανάθεσης, Μοναδιαίοι & Σχισιακοί Τελεστές, Λογικοί Τελεστές
- **Εντολές Επιλογής (Κεφ. 5)**
 - If-then-else, macros, switch
- **Εντολές Επανάληψης (Κεφ. 6)**
 - while, do, for, break, continue, goto

Η Εντολή Επιλογής `if..else`



- Η εντολή επιλογής `if` επιτρέπει την εκτέλεση μιας ή περισσότερων εντολών, εάν η τιμή της λογικής έκφρασης `expression` αποτιμάται σε μη-μηδενική τιμή

```
if ( expression ) { stmt1; stmt2; ... }  
else if (expression) { stmt1; stmt2; ... }  
else { stmt1; stmt2; ... }
```

- **Συνηθισμένο Λάθος I:** `if (i = 0)`, ενώ εννοούσαμε `if (i == 0)`
 - Μπορείτε να γράψετε `(0 == i)`, έτσι ώστε εάν σας ξεφύγει ένα `=` να πάρετε λάθος μεταγλώττισης (lvalue)
- **Έλεγχος Εύρους**
 - $0 \leq i < n$: `if (0 <= i && i < n) ...`
 - $i < 0$ ή $n \leq i$: `if (i < 0 || n <= i) ...`

Η Εντολή Επιλογής `if . . else`



- **Συνηθισμένο Λάθος II ('dangling else'):**

```
if (y != 0)
    if (x != 0)
        result = x / y;
else
    printf("Error: y is equal to 0\n");
```

- Σε ποιο if ανηκει το “else”;

- **Μάθημα:** Χρησιμοποιείται πάντα { } παρενθέσεις σε εντολές if, while, do, κτλ. !

- Στις διαφάνεια ίσως να μην χρησιμοποιούνται πάντα για εξοικονόμηση χώρου.

- Στον κώδικα σας ωστόσο να χρησιμοποιούνται ΠΑΝΤΑ.

Η Μακροεντολή – Τριαδικός Τελεστής



(expr1) ? expr2 : expr3

- Άλλος τρόπος διατύπωσης If-then-else είναι με τη χρήση **Μακροεντολής (Macro)**:
 - (expr1) ? expr2 : expr3 ίδιο με:
 - If (expr1 != 0) { return expr2; } else { return expr3; }
 - Έχει το **πλεονέκτημα** ότι το **αποτέλεσμα** μπορεί να ανατεθεί σε μεταβλητή αλλά **δυσχεραίνει τον δομημένο προγραμματισμό**.

• Παράδειγμα:

```
int i = 1, j = 2, k;  
k = i > j ? i : j; /* k is now 2 */  
k = (i >= 0 ? i : 0) + j; /* k is now 3 */  
printf("%d\n", i > j ? i : j); /* prints 2 */  
return i > j ? i : j; /* returns 2 */
```

Η Εντολή Επιλογής `switch`



- Η εντολή πολλαπλής επιλογής `switch` διαβάζεται ευκολότερα από πολλαπλά `if` και είναι συχνά και γρηγορότερη.

```
switch ( control-expression ) {  
    case constant-expression : statements; break;  
    ...  
    case constant-expression : statements; break;  
    default : statements  
}
```

int or char

Π.χ., 5 ή 5+10

- Θεληματική Παράληψη `break`

```
switch (grade) {  
    case 2:  
    case 1: printf("Passing"); break;  
    case 0: printf("Failing"); break;  
    default: printf("Illegal grade");  
}
```


Περιεχόμενο Διάλεξης



- **Μορφοποιημένη Είσοδος/Έξοδ. (Κεφ. 3)**
 - printf, scanf, ορίσματα
- **Εκφράσεις & Τελεστές (Κεφ. 4)**
 - Αριθμητικοί Τελεστές, Τελεστές Ανάθεσης, Μοναδιαίοι & Σχισιακοί Τελεστές, Λογικοί Τελεστές
- **Εντολές Επιλογής (Κεφ. 5)**
 - If-then-else, macros, switch
- **Εντολές Επανάληψης (Κεφ. 6)**
 - while, do, for, break, continue, goto

Εντολές Επανάληψης της C



- **while (expression) statement;**
 - το *expression* αποτιμάται πριν το *loop*.
 - `while (expr) { stmt1; stmt2;... }`
 - `while (1)` Άπειρο (έξοδος με `break`, `return`, `exit`)
 - Κοινό λάθος: `while (expr); stmt` ή `if (expr);`
- **do statement while (expression) ;**
 - το *statement* εκτελείται πάντα τουλάχιστο μια φορά
 - Καλή ιδέα να χρησιμοποιούνται ΠΑΝΤΑ οι `{ }` και να υπάρχει στοίχιση.
- **for (expr1 ; expr2 ; expr3) statement**
 - Διατυπώνεται και με *while*: `i = 10; while (i > 0) { i--; }`

Εντολές Επανάληψης της C (Επανάληψη For)



- Για απαρίθμηση n στοιχείων
 - Απαρίθμηση (πάνω) από 0 σε $n-1$: `for (i = 0; i < n; i++)`
 - Απαρίθμηση (πάνω) από 1 σε n : `for (i = 1; i <= n; i++)`
 - Απαρίθμηση (κάτω) από $n-1$ σε 0: `for (i = n - 1; i >= 0; i--)`
 - Απαρίθμηση (κάτω) από n σε 1: `for (i = n; i > 0; i--)`
- **ΠΟΤΕ** δεν χρησιμοποιείται **πραγματική τιμή** στον μετρητή για αποφυγή προβλημάτων (π.χ., `f==10.0f`)
- Επεξήγηση Σύνταξης
 - `for (; i > 0; --i)` => Χωρίς αρχικοποίηση
 - `for (; i > 0;)` => Ίδιο με `while (i>0)`
 - `for (;;)` => Infinite Loop
 - `for (int i = 0; i < n; i++)` => OK για C99 (gcc -std=c99 ... το `i` δεν έχει τιμή εκτός του scope του loop)
 - `for (int i = 0, j = 0; i < n; i++)` // όπως και στη JAVA

Εντολές Επανάληψης της C (Έξοδος από Loops)



- Το κανονικό σημείο εξόδου μιας επανάληψης είναι η αρχή (**while** ή **for**) ή το τέλος (**do**).
- Έλεγχος ροής: **break** (loops, switch), **continue** (μόνο σε loops), **goto** τα οποία δουλεύουν για ένα επίπεδο.

```
while (...) {  
    switch (...) { break; }  
}
```

- Το **goto** μπορεί να ανακατευθύνει τη ροή εκτέλεσης σε οποιαδήποτε έκφραση σε συνάρτηση, η οποία είναι σημειωμένη (labeled).

– **Ενάντια στον Δομημένο Προγραμματισμό => Να αποφεύγεται ΠΑΝΤΑ**, εφόσον οδηγεί σε κώδικα spaghetti.