ΚΑΛΕΣ ΠΡΑΚΤΙΚΕΣ ΓΙΑ ΚΑΛΟΓΡΑΜΜΕΝΟ ΚΩΔΙΚΑ

Οι περισσότερες γλώσσες προγραμματισμού παρέχουν ευελιξία στη διάταξη του κώδικα κι έχουν λίγους περιορισμούς για την ονομασία συναρτήσεων και μεταβλητών. Οι ελάχιστες απαιτήσεις είναι το πρόγραμμα να μεταγλωττίζεται χωρίς λάθη και να λειτουργεί σωστά σύμφωνα με τις δεδομένες προδιαγραφές.

Αυτό που πολλοί αρχάριοι προγραμματιστές ξεχνούν είναι πως είναι εξίσου σημαντικό το πρόγραμμα να μπορεί να συντηρηθεί (maintainable code), δηλαδή να είναι εύκολο να γίνουν διορθώσεις, βελτιώσεις, και προσθήκες νέων λειτουργιών σε βάθος χρόνου.

Υπάρχουν πολλά στοιχεία σε ένα πρόγραμμα τα οποία συντελούν στη συντηρησιμότητά του. Θα τα αναλύσουμε ένα-ένα μέσα από ένα παράδειγμα.

Το παρακάτω πρόγραμμα C είναι συντακτικά σωστό και παράγει αποτέλεσμα, αλλά είναι γραμμένο με τρόπο που καθιστά δύσκολη την κατανόηση της λειτουργίας του και κατ' επέκταση οποιαδήποτε διόρθωση ή βελτίωση.

```
#include<stdio.h>
void func(int a[],int s){
int j,m, i; int t;
for(j=0;j<s;j++) {
m=j;
for (i=j+1;i<s;i++) {
if (a[i]<a[m])
m=i;
}


t=a[m];
a[m]=a[j]; a[j]=t;
}}
int main(int argc,char *argv[]) {
int i, a[10]={8,1,-2,9,0,4,3,8,5,10};
func(a,10);
for (i=0;i<10;i++)
printf("%d ",a[i]);
return 0;
}
```

Στις επόμενες σελίδες θα δούμε πώς το πρόγραμμα μπορεί να βελτιωθεί μέσα από αλλαγές στη διάταξη, τα ονόματα, κ.α.. Στο τέλος βρίσκεται μια περίληψη των συμβάσεων που θέλουμε να ακολουθείτε στο μάθημα του Προγραμματισμού.

ΔΙΑΤΑΞΗ

Η διάταξη του προγράμματος πρέπει να ακολουθεί τη λογική ροή του και να βελτιώνει την αναγνωσιμότητά του. Αυτό επιτυγχάνεται με τη σωστή και κυρίως συνεπή χρήση κενών (spaces), κενών γραμμών, στοίχισης και ομαδοποίησης συσχετιζόμενων εντολών. Οι ίδιοι κανόνες ισχύουν για κάθε γραπτό κείμενο. ΣκεφτείτεπωςθαήτανανδενξέχωΡΙΖΑμεσωστάΤι ςλέξειςήανδενξεχωρίζαμετιςπα

ραγράφουςμΕκενέςγραμμέςή αφήναΜΕΜΕγάλα κενάήχρησιμοποιούσαμεΤΥχαίΑκεφαλ ΑΙΑ.

Στοίχιση (Indentation)

Χρησιμοποιήστε κατάλληλη στοίχιση για να δείξετε τη λογική δομή του προγράμματος.

Κάθε εντολή πρέπει να βρίσκεται σε ξεχωριστή γραμμή. Εντολές που υπάγονται στο σώμα μιας σύνθετης εντολής πρέπει να γράφονται ένα tab πιο μέσα από την εντολή στην οποία υπάγονται. Το πιο σύνηθες και αποτελεσματικό μέγεθος για το tab είναι 4 κενά.

Υπάρχουν δύο δημοφιλείς τρόποι τοποθέτησης των αγκίστρων που εσωκλείουν μια ομάδα εντολών:

- Το αριστερό άγκιστρο βρίσκεται στην ίδια στήλη με την αρχή της σύνθετης εντολής, στην επόμενη γραμμή. Το δεξί άγκιστρο βρίσκεται στην ίδια στήλη με το αριστερό, σε γραμμή που δεν περιλαμβάνει τίποτα άλλο. Η μέθοδος είναι γνωστή ως 1TBS.
- Το αριστερό άγκιστρο βρίσκεται στην ίδια γραμμή με τη σύνθετη εντολή. Το δεξί άγκιστρο βρίσκεται στην ίδια στήλη με την αρχή της σύνθετης εντολής, σε ξεχωριστή γραμμή που δεν περιλαμβάνει τίποτα άλλο. Η μέθοδος είναι γνωστή ως Allman.

Και οι δύο μέθοδοι είναι εξίσου αποδεκτές. Χρησιμοποιήστε όποια σας αρέσει αλλά είναι σημαντικό να είστε συνεπείς στην επιλογή σας και να μην αλλάζετε μέθοδο στα μέσα του προγράμματος.

Συμβουλή: Είναι καλή ιδέα να χρησιμοποιείτε άγκιστρα για το "σώμα" σύνθετων εντολών ακόμη κι αν αυτό αποτελείται από μία μόνο εντολή. Αυτό μειώνει την πιθανότητα να γίνει λάθος αν αργότερα προστεθούν επιπλέον εντολές στο σώμα.

Παραδείγματα:

ΟΧΙ

```
for (i=0; i<10; i++) {
printf("%d ", i);
if (size>5) printf("B");
printf("\n");
}
```

ΝΑΙ (1TBS)

```
for (i=0; i<10; i++)
{
    printf("%d ", i);
    if (size>5)
    {
        printf("B");
    }
    printf("\n");
}
```

ΝΑΙ (Allman)

```
for (i=0; i<10; i++)  {
    printf("%d ", i);
    if (size>5) {
        printf("B");
    }
    printf("\n");
}
```

ΟΧΙ

```
switch(trafficLights[i]) {
case RED: case AMBER: stop(cars);
break;
case GREEN: move_forward(cars); break;
}
```

ΝΑΙ

```
switch(trafficLights[i]) {
    case RED:
    case AMBER:
        stop(cars);
        break;
    case GREEN:
        move_forward(cars);
        break;
}
```

Κενά/Κενές γραμμές

Αφήνετε κενά ανάμεσα σε ονόματα μεταβλητών κατά τη δήλωση. Σε εντολές for αφήνετε πάντα κενό ανάμεσα στην αρχικοποίηση, έλεγχο τερματισμού και ανανέωση μετρητή.

ΟΧΙ	**ΝΑΙ**
`double distance,velocity,gravity;`	`double distance, velocity, gravity;`
`tomorrow=today+1;`	`tomorrow = today + 1;`
`numpeople=initpeople=0;`	`numpeople = initpeope = 0;`
`for(i=0;i<size/2;i++)`	`for (i=0; i<size/2; i++)`
`setGrade(course,student,grade);`	`setGrade(course, student, grade);`

Αφήνετε πάντα μια κενή γραμμή ανάμεσα σε υλοποιήσεις διαφορετικών συναρτήσεων.

Αφήνετε μια κενή γραμμή ανάμεσα σε διαφορετικά νοητά "τμήματα" εντός της ίδιας συνάρτησης. Για παράδειγμα, ανάμεσα στο τμήμα δήλωσης μεταβλητών και στο τμήμα εντολών, ή ανάμεσα στο κομμάτι που διαβάζει τα δεδομένα και το κομμάτι που τα επεξεργάζεται.

Μην υπερβάλλετε. Μια κενή γραμμή κάθε μερικές γραμμές κώδικα είναι καλή. Πολλές διαδοχικές αλλοιώνουν τη ροή του κώδικα, αναγκάζουν το μάτι να “ταξιδεύει” πολύ στην οθόνη και μειώνουν αισθητά την ποσότητα κώδικα που χωρά στην οθόνη. Εξίσου άσχημο είναι να υπάρχει μια κενή γραμμή μετά από κάθε μία γραμμή κώδικα.

Αποφεύγετε να έχετε γραμμές μακρύτερες από 80 χαρακτήρες γιατί είναι πολύ δύσκολο να διαβαστούν. Αν μια έκφραση είναι πολύ μεγάλη, συνεχίστε τη στην επόμενη γραμμή.

Οταν μια έκφραση συνεχίζεται στην επόμενη γραμμή, συνίσταται να τη σπάτε έτσι ώστε η επόμενη γραμμή να ξεκινά με τελεστή όπως φαίνεται στο πρώτο παράδειγμα παρακάτω. Αν πρόκειται για συνάρτηση με μεγάλο αριθμό ορισμάτων, μπορείτε ακόμη και να τα βάλετε ένα σε κάθε γραμμή ή να τα ομαδοποιήσετε όπως φαίνεται στο δεύτερο παράδειγμα.

Εναλλακτικά, μπορείτε να βάλετε ένα \ στο τέλος της γραμμής που συνεχίζεται. Το \ όταν εμφανίζεται στο τέλος της γραμμής (αμέσως μετά από αυτό υπάρχει χαρακτήρας αλλαγής γραμμής) υποδηλώνει ότι η εντολή συνεχίζεται στην επόμενη γραμμή.

```
if ( strcmp( employeeName, employeeDB[i] ) == 0
    && employeeId == inputId
    && employeeStatus == CURRENT)
```

```
drawRectangle (lowerLeftX, lowerLeftY,
               length, height, slant,
               solidStyle, fillColorChoice, lineColorChoice);
```

```
if ( strcmp( employeeName, employeeDB[i] ) == 0  &&   \
    employeeId == inputId  &&  \
    employeeStatus == CURRENT)
```

Η εφαρμογή των κανόνων διάταξης στο αρχικό παράδειγμα οδηγεί στο παρακάτω πρόγραμμα:

```
#include<stdio.h>

void func(int a[], int s){

    int j, m, t, i;

    for (j = 0; j < s; j++) {
        m = j;
        for (i = j+1; i < s; i++) {
            if (a[i] < a[m]) {
                m = i;
            }
        }
        t = a[m];
        a[m] = a[j];
        a[j] = t;
    }
}

int main(int argc, char *argv[]) {

    int i, a[10]={8, 1, -2, 9, 0, 4, 3, 8, 5, 10};

    func(a, 10);

    for (i=0; i<10; i++) {
        printf("%d ", a[i]);
    }
    return 0;
}
```

Το πρόγραμμα βελτιώθηκε , αλλά είναι ακόμη δύσκολο να καταλάβει ο αναγνώστης τι ακριβώς κάνει.

ΟΝΟΜΑΤΑ ΜΕΤΑΒΛΗΤΩΝ, ΣΥΝΑΡΤΗΣΕΩΝ, ΤΥΠΩΝ

Ο βασικός λόγος που προγραμματίζουμε σε γλώσσες υψηλού επιπέδου είναι για να είναι τα προγράμματα πιο κατανοητά από τους προγραμματιστές. Αρα η δουλειά του προγραμματιστή είναι κατά μεγάλο βαθμό να επικοινωνεί αποτελεσματικά με ανθρώπους. Γι' αυτό είναι σημαντικό να χρησιμοποιεί σωστό λεξιλόγιο και, φυσικά, να διατάσσει σωστά τον κώδικά του.

Σωστό λεξιλόγιο στα πλαίσια του προγραμματισμού σημαίνει σωστή ονομασία των ονομάτων των μεταβλητών, συναρτήσεων και user-defined τύπων.

Βασικοί κανόνες

Σας δίνεται το παρακάτω κομμάτι κώδικα με την πληροφορία ότι υπολογίζει το νέο ποσό που χρωστά ένας πελάτης στην πιστωτική του κάρτα με βάση το προηγούμενο ποσό και τις νέες αγορές που έκανε.

```
x2 = x2 + fred;
x2 = x2 - x3;
x2 = x2 + nooo(x1, x2);
x2 = x2 + e(x1, x2);
```

Αν σας ζητηθεί να προσθέσετε μια γραμμή η οποία εκτυπώνει στην οθόνη το ποσό για τις νέες αγορές, ποια μεταβλητή θα χρησιμοποιήσετε? Μπορείτε να πείτε τι είναι το x1 και τι κάνουν οι συναρτήσεις?

Δείτε τώρα το ίδιο κομμάτι κώδικα με καλά ονόματα μεταβλητών και συναρτήσεων και πιο καθαρογραμμένους υπολογισμούς.

```
balance = balance + newPurchases;
balance = balance – lastPayment;
balance = balance + calcLateFee(cardPlan, balance);
balance = balance + calcInterest(cardPlan, balance);
```

Ο πιο σημαντικός κανόνας στην ονομασία μεταβλητών είναι ότι το όνομα πρέπει να περιγράφει πλήρως την οντότητα που εκπροσωπεί η μεταβλητή. Αντίστοιχα, το όνομα μιας συνάρτησης πρέπει να περιγράφει πλήρως τη λειτουργία της συνάρτησης. Ετσι, η μεταβλητή που αποθηκεύει το ποσό νέων αγορών λέγεται `newPurchases`, η συνάρτηση που υπολογίζει τον τόκο λέγεται `calcInterest` κτλ.

Παρατηρήστε ότι τα ονόματα μεταβλητών είναι ουσιαστικά ενώ τα ονόματα συναρτήσεων είναι ρήματα ή ρηματικές φράσεις. Ενας τρόπος να “βρείτε” καλά ονόματα είναι περιγράφοντας το πρόβλημα και τη λύση του σε μια παράγραφο και σημειώνοντας τα ουσιαστικά και τα ρήματα που χρησιμοποιήσατε.

Οταν διαλέγετε το νέο όνομα να είστε όσο γίνεται συνοπτικοί χωρίς να χάνεται η έννοια του ονόματος. Αν θεωρείτε ότι είναι απαραίτητη η συντόμευση του ονόματος γιατί διαφορετικά θα ήταν πολύ μακρύ, προσπαθήστε να βρείτε μια συντόμευση που κάνει φανερό το πλήρες όνομα. Για παράδειγμα, μια καλή συντόμευση του ονόματος calculateSalesTax είναι calcSalesTax. Κακές επιλογές θα ήταν salesTax (ουσιαστικό, άρα φέρνει στο μυαλό μεταβλητή), ή clcTx (μη προφανής σημασία).

Αποφεύγετε να χρησιμοποιείτε για διαφορετικές μεταβλητές/συναρτήσεις παρεμφερή ονόματα είτε στη μορφή είτε στη σημασία γιατί είναι εύκολο να τα μπερδέψετε και να χρησιμοποιήσετε το ένα στη θέση του άλλου. Για παράδειγμα, είναι κακή ιδέα να έχετε μεταβλητές total1 και total2. Εξίσου κακή ιδέα είναι να υπάρχουν στο ίδιο πρόγραμμα μεταβλητές totalAmount και finalAmount γιατί έχουν παρόμοια σημασία. Πιο λογικό θα ήταν, για παράδειγμα, totalBeforeTax, totalAfterTax.

Αποφεύγετε να χρησιμοποιείτε το γράμμα l (el) και τον αριθμό 1 (ένα) σε ονόματα μεταβλητών γιατί είναι δύσκολο να φανεί ποιο από τα δύο έχει χρησιμοποιηθεί. Για τον ίδιο λόγο αποφεύγετε τη χρήση του 0 (μηδέν) και O (κεφαλαίο ο).

Κατά σύμβαση, τα ονόματα συναρτήσεων και μεταβλητών είναι γραμμένα με πεζούς χαρακτήρες (σε αντίθεση με τα ονόματα σταθερών τα οποία είναι όλα κεφαλαία).

Σύνθετα ονόματα

Πολλά ονόματα είναι σύνθετα – αποτελούνται από δύο ή, πιο σπάνια, λέξεις. Είναι σημαντικό να υπάρχει κάποιος τρόπος να τις ξεχωρίσουμε. Υπάρχουν δύο μέθοδοι να γίνει αυτό:

- Το πρώτο γράμμα της δεύτερης (και τρίτης, αν υπάρχει) λέξης είναι κεφαλαίο. Λέμε calcSalesTax και όχι calcsalestax. Η μέθοδος είναι γνωστή ως camelCase.
- Οι λέξεις χωρίζονται με κάτω παύλα (underscore), για παράδειγμα calc_sales_tax.

Διαλέξτε όποια μέθοδο προτιμάτε, αλλά να είστε συνεπείς στην επιλογή σας. Χρησιμοποιείτε πάντα την ίδια μέθοδο μέσα σε ένα πρόγραμμα.

Αποφεύγετε ονόματα μήκους μεγαλύτερου του 15.

Ονόματα μετρητών

Μη χρησιμοποιείτε μεταβλητές του ενός χαρακτήρα (πχ. a, h, κτλ). Η μόνη εξαίρεση είναι η χρήση μετρητή σε for-loop που κατά κανόνα είναι i ή j ή k. Αν ο μετρητής πρόκειται να χρησιμοποιηθεί πέραν του loop, τότε ονομάστε τον περιγραφικά, σύμφωνα με τους συνήθεις κανόνες. Επίσης, αν έχετε εμφωλευμένα loop είναι συχνά καλύτερο να χρησιμοποιείτε “καλά” ονόματα για τους μετρητές για να μη μπερδεύεστε.

Για παράδειγμα:

Μετρητής που θα ξαναχρησιμοποιηθεί

```
for (wordLen = 0; word[wordLen] != '\0'; wordLen++) ;
```

Προσωρινός μετρητής

```
for (i=0; i < SIZE; i++) {
    printf("%s\n", names[i]);
}
```

Εμφωλευμένο loop που αρκεί η χρήση i, j

```
for (i=0; i < numRows; i++) {
    for (j=0; j < numCols; i++) {
        printf("%c", crossword[i][j]);
    }
}
```

Εμφωλευμένο loop που χρειάζεται καλά ονόματα μετρητών

```
for ( course = 0; course < numCourses; course++) {
   for (student= 0; student < roster[course]; student++) {
        createAccount(student, course);
   }
}
```

Εννοείται πως δεν είναι καλή ιδέα να χρησιμοποιείτε "γενικά" ονόματα μετρητών όπως counter ή count ή index. Αντίθετα χρησιμοποιείστε numElements, studentCount, κτλ. Το ίδιο ισχύει και για προσωρινές μεταβλητές. Αποφεύγετε τα γενικά ονόματα όπως temp.

Ονόματα δεικτών (pointers)

Κατά σύμβαση, τα ονόματα δεικτών ξεκινούν από p ή περιέχουν ptr. Για παράδειγμα, p_student, phead, nodePtr, κτλ.

Ονόματα τύπων

Κατά σύμβαση, τα ονόματα τύπων τελειώνουν σε T ή _t. Για παράδειγμα, colorT, list_t, κτλ.

Ονόματα σταθερών

Κατά σύμβαση, τα ονόματα σταθερών γράφονται πάντα με κεφαλαία. Το ίδιο ισχύει και για τιμές απαριθμήσεων (enum)

```
#define MAX_COURSE_ID 999
#define NUM_STUDENTS 10
```

```
const int PI = 3.14159;
```

```
typedef enum {RED, AMBER, GREEN} TrafficLightColor_t;
```

Ονόματα flag

Συχνά χρειάζεστε μια boolean μεταβλητή ως "σημαία" (flag) που δηλώνει αν κάποιο γεγονός έχει συμβεί ή όχι. Το όνομα flag είναι κακό όνομα μεταβλητής γιατί δεν προσφέρει καμιά πληροφορία για το τι εκπροσωπεί. Καλύτερα να χρησιμοποιείτε κάτι πιο αντιπροσωπευτικό όπως studentFound, fileExists, printerReady, κτλ. Οπως βλέπετε όλα αυτά τα ονόματα εκπροσωπούν ένα αληθές ή ψευδές γεγονός (fileExists: ή υπάρχει το αρχείο ή όχι).

Η εφαρμογή των κανόνων διάταξης και ονομασίας στο αρχικό παράδειγμα οδηγεί στο παρακάτω πρόγραμμα:

```
#include<stdio.h>

void selectionSort(int numbers[], int size){

    int boundaryIndex, minIndex, i;
    int savedValue;

    for (boundaryIndex = 0; boundaryIndex < size; boundaryIndex++) {

        minIndex = boundaryIndex;

        for (i = boundaryIndex + 1; i < size; i++) {
            if (numbers[i] < numbers[minIndex]) {
                minIndex = i;
            }
        }

        savedValue = numbers[minIndex];
        numbers[minIndex] = numbers[boundaryIndex];
        numbers[boundaryIndex] = savedValue;
    }
}


int main(int argc, char *argv[]) {

    int i, numbers[10] = {8, 1, -2, 9, 0, 4, 3, 8, 5, 10};

    selectionSort(numbers, 10);

    for (i=0; i<10; i++) {
        printf("%d ", numbers[i]);
    }

    return 0;
}
```

Είναι πια φανερό τι κάνει η συνάρτηση: χρησιμοποιεί τον αλγόριθμο selection sort για να ταξινομήσει τα στοιχεία ενός πίνακα ακεραίων. Ακόμα όμως το πρόγραμμα χρειάζεται βελτίωση. Παρόλο που ο κώδικας είναι αρκετά "καθαρός", λείπουν πληροφορίες για το πώς ακριβώς έχει υλοποιηθεί ο αλγόριθμος και πώς επιδρά στον πίνακα ακεραίων.

ΣΧΟΛΙΑ

Πολλοί προγραμματιστές θεωρούν ότι τα σχόλια είναι περιττά όταν ο κώδικας έχει σωστή διάταξη και καλά ονόματα. Υποστηρίζουν ότι ο καλός κώδικας είναι self-documenting, δηλαδή ό,τι χρειάζεται να ξέρεις είναι ήδη μέσα στον κώδικα. Αλλοι, ανάμεσά τους κι εμείς, υποστηρίζουν ότι τα σχόλια είναι απαραίτητα για την κατανόηση του κώδικα από αυτούς που προσπαθούν να τον διορθώσουν ή μεταβάλλουν. Ολοι πάντως συμφωνούν ότι κακογραμμένα σχόλια είναι χειρότερα από καθόλου σχόλια.

Τα σχόλια πρέπει να είναι συνοπτικά και να εξηγούν το σκοπό ενός κομματιού κώδικα. Δεν πρέπει απλά να επαναλαμβάνουν αυτό που κάνει κάθε εντολή. Για παράδειγμα, ηλοι καταλαβαίνουν ότι η έκφραση `col++` αυξάνει την τιμή της μεταβλητής `col` κατά ένα. Αλλά τι ακριβώς σημαίνει αυτό? Μπορεί, για παράδειγμα να είναι μέρος μιας συνάρτησης που μετακινεί τον παίκτη ενός παιχνιδιού και να σημαίνει ότι ο παίκτης μετακινήθηκε μια θέση δεξιά. *Αυτό ακριβώς* είναι που πρέπει να γραφτεί στο σχόλιο κι όχι το γεγονός ότι αυξήθηκε μια μεταβλητή.

ΟΧΙ	**ΝΑΙ**

`/* set product to num */`
`product = num;`
 	
`/* loop from 2 to power */`
`for (i = 2; i <= power; i++) {`
`  /* multiply num by product */`
`  product = product * num;`
`}`
 	
`/* print the result */`
`printf("Product = %d\n", product);`
	`/* Raise the integer 'num' to the power 'power' and print the result */`
	
	`product = num;`
	
	`for (i = 2; i <= power; i++) {`
	`  product = product * num;`
	`}`
	
	`printf("Product = %d\n", product);`

ΟΧΙ	**ΝΑΙ**

/*Η sygkekrimenh synarthsh pairnei ws orisma enan deikth. Se ayth th synarthsh elegxoyme an to c einai iso me 0 kai an symbainei ayto tote den yparxei xarakthras. An twra to c einai iso me A-Z tote to metatrepei se mikro xarakthra pairnwntas ton ASCII kwdiko toys kai to kataxwrei sto s[i].*/
 	
`void lower(char *s) {`
`  int i;`
`  for (i=0; 1; i+=1) {`
`  char c=s[i];`
`  if (c==0) break;`
`  if (c>='A' && c<='Z')  s[i]=c+'a'-'A';`
`  }`
`}`
	/* Η sunartisi lower pernei mia sumboloseira kai metatrepei ola ta kefalaia grammata tis se mikra.
	*/
	
	(Δε θα σχολιάσουμε εδώ το αν ο κώδικας είναι καλογραμμένος ή σωστός. Επικεντρωθείτε στο σχόλιο. Η παράγραφος που περιγράφει ουσιαστικά γραμμή-γραμμή τον κώδικα, τελικά δίνει λιγότερη πληροφορία από το σύντομο σχόλιο δύο γραμμών που λέει τι επιτυγχάνει ο κώδικας.)

ΣΙΓΟΥΡΑ ΟΧΙ	`/* if the student flag is zero */` `if (studentFlag == 0) {`
ΟΧΙ	`/* if the student is new */` `if (studentFlag == 0) {`
ΝΑΙ	`if (studentStatus == NEW) {`

Παρατηρείστε πως στο τελευταίο παράδειγμα όχι μόνο αλλάξαμε το όνομα της μεταβλητής αλλά χρησιμοποιήσαμε και κάποιο όνομα (πιθανώς κάποιο enum ή σταθερά) για την τιμή της κατάστασης του φοιτητή. Τώρα ο κώδικας είναι πιο καθαρός, δε χρειάζεται να αναρωτιόμαστε τι αντιπροσωπεύει η

μεταβλητή `studentFlag` ούτε τι ακριβώς σημαίνει να έχει την τιμή μηδέν. Το `studentStatus` είναι η κατάσταση φοίτησης του εν λόγω φοιτητή, και οι δυνατές τιμές του θα μπορούσαν να είναι για παράδειγμα `{NEW, CURRENT, FORMER}`. Το σχόλιο δε χρειάζεται καν, γιατί ο έλεγχος που γίνεται στην `if` προκύπτει από τα ονόματα.

Είναι σημαντικό να επαναλάβουμε ότι τα σχόλια πρέπει να εξηγούν το *σκοπό* ενός κομματιού κώδικα κι όχι απλά να δίνουν μια περίληψη του τι κάνει. Πολλές φορές, τα σχόλια περιέχουν πληροφορίες που δεν είναι προφανείς αν κανείς απλά διαβάζει τον κώδικα.

ΟΧΙ	**ΝΑΙ**

`/* check each character in "pass" until you find a $ or a @ or until all characters have been checked. */`
 	
`i = 0;`
`while (pass[i] != '\0') {`
`  if (pass[i] == '$'`
`  || pass[i] == '@') {`
`  return TRUE;`
`  }`
`  i++;`
`}`
`return FALSE;`
	`/* check whether a password contains at least one $ or @ */`
	
	`i = 0;`
	`while (password[i] != '\0') {`
	`  if (password[i] == '$'`
	`  || password[i] == '@') {`
	`  return TRUE;`
	`  }`
	`  i++;`
	`}`
	`return FALSE;`

ΠΡΟΣΟΧΗ: Η στοίχιση σχολίων πρέπει να ακολουθεί πάντα τη στοίχιση του κώδικα που σχολιάζουν. Σχόλια που τοποθετούνται δεξιά μιας εντολής πρέπει να είναι πολύ συνοπτικά και να μην υπερβαίνουν την 80ή στήλη. Αν αυτό δεν είναι δυνατό, τότε πρέπει να τοποθετούνται πριν τον κώδικα που σχολιάζουν, και όχι δίπλα. Σε καμία περίπτωση δε βάζουμε σχόλια στην επόμενη γραμμή από αυτή που σχολιάζουν.

Κάτι άλλο που πρέπει να αναφέρουμε είναι η τάση πολλών αρχάριων προγραμματιστών να χρησιμοποιούν μη περιγραφικά ονόματα μεταβλητών προσθέτοντας δίπλα ένα επεξηγηματικό σχόλιο. Αυτό είναι κακή τακτική γιατί αναγκάζει τον αναγνώστη να ανατρέχει στο σχόλιο κάθε φορά που συναντά τη μεταβλητή σε μεταγενέστερο σημείο του προγράμματος και δε θυμάται τι είναι. Εάν πρόκειται να εισαχθεί σχόλιο δίπλα σε όνομα μεταβλητής, αυτό θα είναι για να δώσει μια επιπλέον πληροφορία που πιθανώς χρειάζεται και όχι για να εξηγήσει το όνομα.

ΟΧΙ	**ΝΑΙ**
`double d; /* apostasi */`	`double distance; /* in kilometers */`

ΣΤΑΘΕΡΕΣ ΤΙΜΕΣ (LITERALS)

Τι θα συμβεί αν θέλουμε να αλλάξουμε το μέγεθος του πίνακα στο παράδειγμά μας? Θα πρέπει να βρούμε και να αλλάξουμε όλες τις εμφανίσεις του αριθμού 10. Δε γίνεται να κάνουμε find+replace γιατί τότε θα αλλαχτεί και το τελευταίο στοιχείο που είναι αποθηκευμένο στον πίνακα πράγμα που θα οδηγήσει σε λάθος αποτέλεσμα. Η λύση είναι η χρήση του preprocessor directive `#define` ώστε να ορίσουμε το μέγεθος σε ένα σημείο και να κάνουμε οποιεσδήποτε αλλαγές της τιμής μόνο σε αυτό το σημείο. Πρέπει να προσέχουμε να χρησιμοποιούμε παντού το όνομα που επιλέξαμε γι αυτή την τιμή.

ΟΧΙ	**ΝΑΙ**

`#define SIZE 10`
 	
`int nums[SIZE];`
`for (i=0; i<10; i++) {`
`  nums[i] = i * 2;`
`}`
	`#define SIZE 10`
	
	`int nums[SIZE];`
	`for (i=0; i<SIZE; i++) {`
	`  nums[i] = i * 2;`
	`}`

ΤΕΛΙΚΗ ΜΟΡΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

```c
#include<stdio.h>

#define SIZE 10 /* number of elements in array to be sorted */

void selectionSort(int numbers[], int size);

int main(int argc, char *argv[]) {

    int numbers[SIZE] = {8, 1, -2, 9, 0, 4, 3, 8, 5, 10};

    selectionSort(numbers, SIZE);

    for (i = 0; i < SIZE; i++) {
        printf("%d ", numbers[i]);
    }

    return 0;
}

/* selectionSort(numbers, size)

 * Sorts an array of integers in ascending order using selection sort.
 *
 * Parameters:
            numbers: an array of integers
            size: the size of the array
 * Preconditions: none
 * Postconditions: the array is sorted in ascending order
 */

void selectionSort(int numbers[], int size){

    int boundaryIndex, minIndex, i, savedValue;

    /* The part between indices 0 and boundaryIndex is maintained
    in sorted order, the rest is unsorted. */
    for (boundaryIndex = 0; boundaryIndex < size; boundaryIndex++) {

        /* find the smallest element in the unsorted part */
        minIndex = boundaryIndex;

        for (i = boundaryIndex + 1; i < size; i++) {
            if (numbers[i] < numbers[minIndex]) {
                minIndex = i;
            }
        }
        /* swap smallest unsorted element with element at boundary */
        savedValue = numbers[minIndex];
        numbers[minIndex] = numbers[boundaryIndex];
        numbers[boundaryIndex] = savedValue;
    }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

McConnell, Steve. Code Complete: A Practical Handbook of Software Construction. 2nd ed. Microsoft Press, 2004

Ranade, Jay and Nash, Alan. The Elements of C Programming Style. Mcgraw-Hill, 1992

Kernighan, Brian W. and Pike, Rob. The Practice of Programming. Addison-Wesley, 1999

Goodliffe, Pete. Code Craft: The Practice of Writing Excellent Code. No Starch Press, 2006

ΚΑΝΟΝΕΣ ΜΟΡΦΟΠΟΙΗΣΗΣ ΚΩΔΙΚΑ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ 1

Εκτός από τους παρακάτω κανόνες, περιμένουμε να έχετε διαβάσει το υπόλοιπο φυλλάδιο και να εφαρμόζετε τις τακτικές που αναφέρονται σε αυτό.

- Επιλέξτε μία από τις δύο προτεινόμενες μεθόδους στοίχισης και μείνετε συνεπείς σε αυτή. Τα σχόλια στοιχίζονται όπως ο κώδικας.
- Οι γραμμές δεν πρέπει να υπερβαίνουν την 80ή στήλη.
- Επιλέξτε μία από τις δύο προτεινόμενες μεθόδους ονομασίας σύνθετων μεταβλητών και μείνετε συνεπείς σε αυτή. Τα ονόματα πρέπει να είναι πάντα περιγραφικά.
- Τα ονόματα σταθερών, literals και τιμών enum γράφονται με κεφαλαία. Τα ονόματα μεταβλητών και συναρτήσεων ξεκινούν πάντα με μικρό.
- Σχόλια πρέπει να τοποθετούνται στα εξής σημεία:
  - Στην αρχή του αρχείου, με μια συνοπτική περιγραφή του σκοπού του προγράμματος και το ονοματεπώνυμο του ατόμου που το έγραψε.
  - Πριν από κάθε ορισμό συνάρτησης με περιγραφή του σκοπού της συνάρτησης, των παραμέτρων και της τιμής επιστροφής.
  - Σε δυσνόητα/πολύπλοκα κομμάτια κώδικα, πάντα πριν ή δεξιά από αυτά.
- Πρέπει να υπάρχει ακριβώς μία κενή γραμμή:
  - Μετά την τελευταία εντολή #include.
  - Μετά την τελευταία εντολή #define
  - Ανάμεσα σε ορισμούς συναρτήσεων (δηλαδή μετά το τελικό } της κάθε συνάρτησης)
  - Ανάμεσα στις δηλώσεις μεταβλητών και στην πρώτη εντολή μιας συνάρτησης.
  - Πριν από κάθε σχόλιο που βρίσκεται σε δική του γραμμή μέσα στον κώδικα.