ΤΟ ΑΣΧΗΜΟΠΑΠΟ ΠΟΥ ΕΓΙΝΕ ΚΥΚΝΟΣ

ή ΠΩΣ ΝΑ ΓΡΑΦΕΤΕ ΚΑΛΟ ΚΩΔΙΚΑ

Βάνα Ντουφεξή

Οι περισσότερες γλώσσες προγραμματισμού επιτρέπουν στον προγραμματιστή να διατάξει τον κώδικα όπως επιθυμεί και 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.

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 τυπικό παράδειγμα κώδικα γραμμένου από αρχάριο προγραμματιστή. Το πρόγραμμα είναι συντακτικά σωστό, αλλά είναι υπερβολικά δύσκολο να βρεθούν τυχόν λογικά λάθη ή να γίνουν αλλαγές.

```
#include<stdio.h>
int i;
void func(int a[],int s){
int j,m; 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 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;
}
```

Μπορείτε να βρείτε τι κάνει το πρόγραμμα αυτό χωρίς να το τρέξετε?

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, κενών γραμμών, στοίχισης και ομαδοποίησης συσχετιζόμενων εντολών.

Οι ίδιοι κανόνες ισχύουν για κάθε γραπτό κείμενο. Σκεφτείτεπωςθαήτανανδεχρησιμοποιούσαμεποτέκενάστ ιςπροτασειςήανδενξεχω ρίζαμετιςπαραγράφουςμεγραμμέςήανχωρίζαμετιςλ έξειςσελάθοςσυλλαβές.

Στοίχιση (Indentation)

Χρησιμοποιήστε 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 αποτελεσματικό μέγεθος για το tab είναι 4 κενά.

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

Μέθοδος 1:
Το αριστερό άγκιστρο βρίσκεται στην ίδια στήλη με την αρχή της σύνθετης εντολής, στην ακριβώς επόμενη γραμμή. Το δεξί άγκιστρο βρίσκεται στην ίδια στήλη με το αριστερό, σε ξεχωριστή γραμμή που δεν περιλαμβάνει τίποτα άλλο.

Μέθοδος 2:
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.

Και οι δύο μέθοδοι είναι εξίσου αποδεκτές. Μπορείτε να χρησιμοποιήσετε 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 το σώμα αποτελείται από μία μόνο εντολή. Ο λόγος είναι ότι μειώνεται η πιθανότητα να γίνει κάποιο λάθος αν αργότερα προστεθούν επιπλέον εντολές στο σώμα.

ΟΧΙ	**ΝΑΙ**

`for (i=0; i<10; i++) {`
`printf("%d ", i);`
`if (size>5) printf("Too big");`
`}`
	Μέθοδος 1
	
	`for (i=0; i<10; i++)`
	`{`
	`  printf("%d ", i);`
	`  if (size>5)`
	`  printf("Too big");`
	`}`
	Μέθοδος 2
	
	`for (i=0; i<10; i++) {`
	`  printf("%d ", i);`
	`  if (size>5)`
	`  printf("Too big");`
	`}`

OXI	NAI

`for (id = 0; id < 10; id++) {`
`  printf("%d", grade[id]);`
`  if (grade[id] >= 5)`
`  printf("Pass");`
`  printf("\n");`
`}`
	`for (id = 0; id < 10; id++) {`
	`  printf("%d", grade[id]);`
	`  if (grade[id] >= 5)`
	`  printf("Pass");`
	`  printf("\n");`
	`}`
	Ακολουθώντας τη συμβουλή:
	
	`for (id = 0; id < 10; id++) {`
	`  printf("%d", grade[id]);`
	`  if (grade[id] >= 5) {`
	`  printf("Pass");`
	`  }`
	`  printf("\n");`
	`}`

`switch(trafficLights[i]) {`
`case RED: stop(cars); break;`
`case AMBER: move_forward(cars, 1);`
`stop(cars, 2); break;`
`case GREEN: move_forward(cars); break;`
`}`
	`switch(trafficLights[i]) {`
	`  case RED:`
	`  stop(cars);`
	`  break;`
	`  case AMBER:`
	`  move_forward(cars, 1);`
	`  stop(cars, 2);`
	`  break;`
	`  case GREEN:`
	`  move_forward(cars);`
	`  break;`
	`}`

Κενά

Αφήνετε κενά ανάμεσα σε αναγνωριστικά (identifiers). Σε πολύπλοκες εκφράσεις, είναι συχνά καλό να αφήνετε κενό ανάμεσα σε ένα τελεστή (operator) και τελεσταίους (operands). Σε εντολές for αφήνετε πάντα κενό ανάμεσα στην αρχικοποίηση, έλεγχο τερματισμού και ανανέωση μετρητή.

OXI	NAI
`tomorrow=today+1;`	`tomorrow = today + 1;`
`x=y+2*z-5;`	`x = y + 2 * z - 5;`
`numpeople=initpeople=0;`	`numpeople = initpeope = 0;`
`for(i=0;i<size/2;i++)`	`for (i=0; i < size/2 ; i++)`

OXI	`setStudentGrades(classlist,classId,numstudents);`
NAI	`setStudentGrades( classlist, classId, numstudents);`

OXI	`crossword[startRow+wordLength][startCol]=BLACK;`
NAI	`crossword[ startRow + wordLength ][ startCol ] = BLACK;`

Κενές γραμμές

Αφήνετε πάντα τουλάχιστον μια κενή γραμμή ανάμεσα σε υλοποιήσεις διαφορετικών συναρτήσεων.

Μπορείτε να αφήνετε μια κενή γραμμή ανάμεσα σε διαφορετικά "τμήματα" εντός της ίδιας συνάρτησης, ακριβώς όπως θα αφήνατε κενό ανάμεσα σε παραγράφους. 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.

```
double average (int scores[], int size) {

    int i;

    sum = 0;
    for (i=0; i < size; i++) {
        sum = sum + scores[i];
    }

    return (sum/size);
}
```

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.

Όταν μια έκφραση συνεχίζεται στην επόμενη γραμμή κάντε το εμφανές με το να "χωρίσετε" την έκφραση σε κάποιο τελεστή. 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 παρακάτω παράδειγμα.

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

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

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

```
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>

int i;

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

    int j, m;
    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 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 ήδη βελτιώθηκε πολύ, 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 και, ως αποτέλεσμα, να έχουν λιγότερα λάθη. Άρα 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.

Γενικά ονόματα

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.

```
x = x - xx;
xxx = maria + f(maria);
x = x + p(x1, x) + xxx;
x = x + e(x1, x);
```

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

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

```
balance = balance - lastPayment;
monthlyTotal = newPurchases + calcSalesTax(newPurchases);
balance = balance + calcLateFee(customerID, balance) + monthlyTotal;
balance = balance + calcInterest(customerID, 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. Έτσι, η μεταβλητή που αποθηκεύει το καθαρό ποσό νέων αγορών λέγεται newPurchases, η συνάρτηση που υπολογίζει τον τόκο λέγεται calcInterest κτλ.

Παρατηρείστε ότι τα ονόματα μεταβλητών είναι ουσιαστικά ενώ τα ονόματα συναρτήσεων είναι ρήματα ή ρηματικές φράσεις. Ένας εύκολος 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.

Όταν διαλέγετε το νέο όνομα αποφύγετε να χρησιμοποιείτε συντομογραφίες. Αν θεωρείτε ότι είναι απαραίτητη η συντόμευση του ονόματος γιατί διαφορετικά θα είναι πολύ μακρύ, προσπαθήστε να βρείτε μια συντόμευση που κάνει φανερό το πλήρες όνομα. Για παράδειγμα, η συνάρτηση που υπολογίζει το φόρο πωλήσεων ονομάστηκε calcSalesTax έναντι του calculateSalesTax.

Αποφεύγετε να χρησιμοποιείτε 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, στο πρόγραμμα που υπολογίζει το νέο οφειλόμενο ποσό της πιστωτικής κάρτας, θα ήταν κακή ιδέα να χρησιμοποιούσαμε total1 και total2 για τα επιμέρους σύνολα. Εξίσου κακή ιδέα θα ήταν να χρησιμοποιήσουμε totalAmount και finalAmount γιατί έχουν παρόμοια σημασία.

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).

Ίσως έχετε παρατηρήσει τα πιο πολλά ονόματα είναι σύνθετα – αποτελούνται από δύο λέξεις. Είναι λοιπόν σημαντικό να υπάρχει κάποιος τρόπος να τις ξεχωρίσουμε. Υπάρχουν δύο μέθοδοι να γίνει αυτό:

Μέθοδος 1:
Το πρώτο γράμμα της δεύτερης (και τρίτης, αν υπάρχει) λέξης είναι κεφαλαίο. Λέμε calcSalesTax και όχι calcsalestax.

Μέθοδος 2:
Οι λέξεις χωρίζονται με underscores. Λέμε calc_sales_tax και όχι calcsalestax.

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

Ονόματα μετρητών

Μη χρησιμοποιείτε μεταβλητές του ενός χαρακτήρα (πχ. a, x, 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 (wordLength = 0; word[wordLength] != '\0'; wordLength++);
```

Προσωρινός μετρητής

```
for (i=0; i < SIZE; i++) {
    printf("%s\n", studentNames[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 ( courseIndex = 0; courseIndex < numCourses; courseIndex++) {
   for (studentId= 0; studentId < courseSize[courseIndex]; studentId++) {
        createAccount(studentId, courseIndex);
   }
}
```

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

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 έχει καμιά πληροφορία για το τι εκπροσωπεί (είναι σα να ονομάζατε μια μεταβλητή data). Καλύτερα να χρησιμοποιείτε κάτι πιο αντιπροσωπευτικό όπως elementFound, fileExists, printerReady, κτλ. Όπως βλέπετε όλα αυτά τα ονόματα εκπροσωπούν ένα αληθές ή ψευδές γεγονός (fileExists: ή υπάρχει το αρχείο ή όχι).

Ονόματα δεικτών (pointers)

Κατά σύμβαση, τα ονόματα δεικτών ξεκινούν από p ή περιέχουν ptr. Για παράδειγμα, p_node, p_row, node_ptr, κτλ.

Ονόματα τύπων

Κατά σύμβαση, τα ονόματα τύπων ξεκινούν από t ή τελειώνουν σε T. Για παράδειγμα, colorT, t_color, t_list, 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} t_TrafficLightColor;
```

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

```
#include<stdio.h>

int i;

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

    int boundaryIndex, minIndex;
    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 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

Κάποιοι 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. Άλλοι, ανάμεσά τους κι εμείς, υποστηρίζουν ότι τα σχόλια είναι απαραίτητα για την κατανόηση του κώδικα από αυτούς που προσπαθούν να τον διορθώσουν ή μεταβάλλουν. Όλοι πάντως συμφωνούν ότι κακογραμμένα σχόλια είναι χειρότερα από καθόλου σχόλια.

Ας δούμε λοιπόν πώς πρέπει να γράφει κανείς σχόλια σε κώδικα. Συγκεκριμένες προδιαγραφές για τη σύνταξη και το περιεχόμενο σχολίων στα προγράμματα του Προγραμματισμού 1 θα σας δοθούν σε φυλλάδια του εργαστηρίου.

Τα σχόλια πρέπει να είναι συνοπτικά και να εξηγούν το σκοπό ενός κομματιού κώδικα. Δεν πρέπει να επαναλαμβάνουν αυτό που κάνει κάθε εντολή. Δεν πρέπει να ξεχνάμε ότι τα σχόλια απευθύνονται σε άλλους προγραμματιστές. Όλοι καταλαβαίνουν ότι έκφραση row++ αυξάνει τη μεταβλητή row κατά ένα. Αλλά τι ακριβώς σημαίνει αυτό? Ποιος είναι ο σκοπός του? 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* είναι που πρέπει να γραφτεί στο σχόλιο, 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 **(παράδειγμα από εργασία χειμ. 2007)**	**ΝΑΙ**

/*H 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 συνάρτηση αυτή παίρνει μια συμβολοσειρά και μετατρέπει όλα τα κεφαλαία γράμματά της σε μικρά */
	
	(Δε θα σχολιάσουμε εδώ το αν ο κώδικας είναι καλογραμμένος ή σωστός)

ΣΙΓΟΥΡΑ ΟΧΙ	/* if the student flag is zero */
	`if (studentFlag == 0) {`
	`  ...`
	`}`

ΟΧΙ	/* if the student is new */
	`if (studentFlag == 0) {`
	`  ...`
	`}`

ΝΑΙ	/* create a new student account */
	`if (studentStatus == NEW) {`
	`  ...`
	`}`

Παρατηρείστε πως στο τελευταίο παράδειγμα αλλάξαμε το όνομα και τύπο της μεταβλητής. Αντί να χρησιμοποιήσουμε ένα απλό flag χρησιμοποιήσαμε μια απαρίθμηση (enum). 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 είναι για παράδειγμα {NEW, CURRENT, FORMER}.

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

Η στοίχιση σχολίων πρέπει να ακολουθεί πάντα τη στοίχιση του κώδικα.

Ας δούμε και πάλι τον αρχικό κώδικα, αυτή τη φορά με σχόλια.

```
#include<stdio.h>

int i;

/* selectionSort(numbers, size)

 * Purpose: Sort an array of integers in ascending order
        using the selection sort algorithm
 * 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;
    int savedValue;

    /* The section between indices 0 and boundary_index-1
       is maintained in sorted order. */

    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;
    }
}

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

    int 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;
}
```

Ο κώδικάς μας είναι σχεδόν έτοιμος. Μένουν μόνο δύο θέματα να αγγίξουμε.

Το πρώτο είναι η χρήση της καθολικής μεταβλητής i. Ο προγραμματιστής που έγραψε το αρχικό πρόγραμμα αναμφίβολα σκέφτηκε πως αφού το i χρησιμοποιείται και στη main και στη selectionSort απλά ως μετρητής, θα είναι περιττό να το δηλώσει δύο φορές, μία σε κάθε συνάρτηση. Δυστυχώς, τέτοιες σκέψεις είναι επικίνδυνες.

Το δεύτερο θέμα είναι η χρήση της τιμής 10 σε αρκετά σημεία του προγράμματος, το οποίο επίσης δυσκολεύει τη συντηρησιμότητά του όπως θα δούμε.

Σημείωση: Στην τάξη του Προγραμματισμού Ι θα έχουμε συγκεκριμένα πρότυπα για το σχολιασμό προγραμμάτων τα οποία πρέπει να ακολουθούνται σε όλες τις εργασίες. Τα πρότυπα σχολιασμού περιγράφονται σε διαφορετικό φυλλάδιο που μπορείτε να βρείτε στην ιστοσελίδα του εργαστηρίου.

ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Οι καθολικές μεταβλητές είναι προσβάσιμες σε οποιοδήποτε σημείο του προγράμματος. Ο κίνδυνος σε αυτό είναι ότι μπορεί κατά λάθος να αλλαχθεί η τιμή της μεταβλητής σε ένα σημείο του προγράμματος αλλά αυτή η ενδεχόμενη αλλαγή να μη ληφθεί υπόψη σε κάποιο άλλο σημείο.

Ένα άλλο πρόβλημα είναι ότι μπορεί να οριστεί κάποια τοπική μεταβλητή ή παράμετρος με το ίδιο όνομα, πράγμα που θα δημιουργήσει "σύγκρουση" ονομάτων (η τοπική μεταβλητή κερδίζει). Αυτό είναι ένα λάθος που εμφανίζεται πολύ συχνά σε κώδικα αρχάριων προγραμματιστών.

```
int player_pos;

void move_player (int new_pos) {
    int player_pos;
    player_pos = new_pos;
}

void play_round() {
   player_pos = 0;
   move_player(3);
   /* Η τιμή του player_pos σε αυτό το σημείο εξακολουθεί να είναι 0 ! */
   ...
}
```

ΣΤΑΘΕΡΕΣ ΤΙΜΕΣ (LITERALS)

Τι θα συμβεί αν θέλουμε να αλλάξουμε το μέγεθος του πίνακα στο παράδειγμά μας? Θα πρέπει να ψάξουμε όλες τις εμφανίσεις του αριθμού 10 και να τις αλλάξουμε. 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 αποτελέσματα. Η λύση είναι η χρήση του preprocessor directive #define η οποία μας επιτρέπει να θέσουμε το μέγεθος σε ένα σημείο και να κάνουμε οποιεσδήποτε αλλαγές μόνο σε αυτό το σημείο.

Ο ΚΥΚΝΟΣ

Μπορούμε τώρα να δούμε το πρόγραμμα στην τελική του μορφή (έγινε πια κύκνος...)

```
#include<stdio.h>

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


/* selectionSort(numbers, size)

 * Purpose: Sort an array of integers in ascending order
        using the selection sort algorithm
 * 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;
    int savedValue;

    /* The section between indices 0 and boundary_index-1
       is maintained in sorted order. */

    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;
    }
}

/*
 * main
 */

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;
}
```

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
http://www.oualline.com/style/index.html

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

-- Βάνα Ντουφεξή