Αναγνωσιμότητα και Συντηρησιμότητα κώδικα

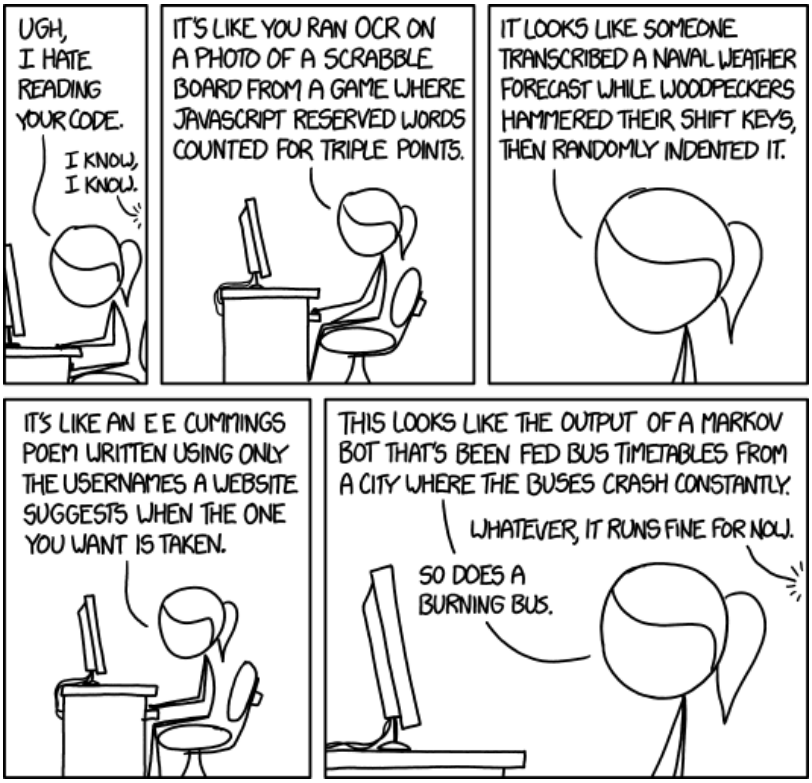
Περιεχόμενα

Εισαγωγή

Οι περισσότερες γλώσσες προγραμματισμού παρέχουν ευελιξία στη διάταξη του κώδικα και στην ονομασία συναρτήσεων και μεταβλητών, αρκεί το πρόγραμμα να μεταγλωττίζεται χωρίς λάθη και να λειτουργεί σωστά σύμφωνα με τις δεδομένες προδιαγραφές.

Όμως, ένα καλό πρόγραμμα δεν πρέπει μόνο να "τρέχει". Πρέπει να είναι ευανάγνωστο (readable), επεκτάσιμο και συντηρήσιμο (maintainable), ώστε να διευκολύνονται τυχόν διορθώσεις καθώς και βελτιώσεις ή προσθήκες νέων λειτουργιών σε βάθος χρόνου.

Σε αυτό το φυλλάδιο συνοψίζονται οι πιο συνηθισμένες πρακτικές για τη συγγραφή καθαρού και κατανοητού κώδικα.



Πηγή: xkcd.com

Ονομασία μεταβλητών, συναρτήσεων και σταθερών

Το όνομα μιας μεταβλητής ή σταθεράς πρέπει να είναι συνοπτικό και να περιγράφει με ακρίβεια την αποθηκευμένη ποσότητα. Αντίστοιχα, το όνομα μιας συνάρτησης πρέπει να περιγράφει συνοπτικά και με ακρίβεια τη λειτουργία της συνάρτησης. Επομένως, τα ονόματα μεταβλητών και σταθερών είναι ουσιαστικά και τα ονόματα συναρτήσεων ρήματα.

Αυτός ο κανόνας ισχύει για όλα τα ονόματα ενός προγράμματος. Για παράδειγμα, μια μεταβλητή που λειτουργεί ως "σημαία" (flag), δηλαδή δηλώνει αν κάποιο γεγονός ισχύει ή όχι, δεν πρέπει να λέγεται flag ή temp αλλά κάτι αντιπροσωπευτικό, π.χ. userExists (εκφράζει αν υπάρχει ή όχι ο χρήστης) ή isSymmetric (εκφράζει αν ένας πίνακας είναι συμμετρικός ή όχι).

Συμβάσεις γραφής σύνθετων ονομάτων (snake case vs. camel case)

Σε σύνθετα ονόματα, που αποτελούνται από δύο ή περισσότερες λέξεις, αυτές είτε χωρίζονται με μια κάτω παύλα (π.χ. num_students) ή γράφονται κολλητά, με το πρώτο γράμμα κάθε λέξης μετά την πρώτη να είναι κεφαλαίο (π.χ. numStudents).

Πηγή: wikipedia
Δημιουργός: Emoji One

Επιλέξτε μία από αυτές τις συμβάσεις και ακολουθήστε τη σε όλο σας το πρόγραμμα.

Συμβάσεις χρήσης κεφαλαίων/πεζών γραμμάτων

Τα ονόματα σταθερών γράφονται πάντα με όλα τα γράμματα κεφαλαία, για παράδειγμα PI ή NUM_STUDENTS.

Τα ονόματα μεταβλητών και συναρτήσεων γράφονται με όλα τα γράμματα μικρά με μόνη εξαίρεση τη γραφή camelCase.

Συμβάσεις ονομασίας μετρητών

Σε επαναλήψεις for να προτιμάτε i, j, k ως μετρητές ή row, col όταν διατρέχετε δισδιάστατους πίνακες. Είναι προτιμότερο να επαναχρησιμοποιείτε αυτά τα ονόματα αντί να ορίζετε νέα. Πιο περιγραφικά ονόματα χρησιμοποιούνται (α) όταν η τιμή θα χρησιμοποιηθεί μετά την επανάληψη και (β) σε εμφωλευμένες for αν βελτιώνουν την αναγνωσιμότητα.

Συμβάσεις ονομασίας σύνθετων τύπων

Σε τύπους δεικτών προστίθεται στο όνομα μια ένδειξη δείκτη, είτε p είτε ptr ως πρόθεμα ή επίθεμα, για παράδειγμα p_student ή studentPtr.

Αντίστοιχα προστίθεται ως πρόθεμα ή επίθεμα ένα t ή T σε ονόματα τύπων struct ή enum, ειδικά αν έχουν οριστεί με χρήση typedef, για παράδειγμα list_t ή colorT.

Τι να αποφεύγετε

Αποφεύγετε να χρησιμοποιείτε παρεμφερή ονόματα. Για παράδειγμα, αν στο ίδιο πρόγραμμα υπάρχουν δύο μεταβλητές με ονόματα finalAmount και totalAmount, είναι πολύ εύκολο να υπάρξει σύγχυση και χρησιμοποιηθεί λάθος μεταβλητή σε κάποιο υπολογισμό. Θα πρέπει η διαφορά ανάμεσα στις ποσότητες που αποθηκεύονται σε αυτές τις μεταβλητές να γίνεται σαφής μέσα από τα ονόματά τους, για παράδειγμα, amountBeforeTax, amountAfterTax.

Αποφεύγετε ονόματα που υπερβαίνουν τους 15 χαρακτήρες.

Αποφεύγετε συντομογραφίες που αφαιρούν τόσα πολλά γράμματα ώστε να σημαίνουν κάτι διαφορετικό ή κάτι ασαφές.

Για παράδειγμα, το clSlsTx είναι κακή συντομογραφία του ονόματος συνάρτησης calcSalesTax γιατί λείπουν πολλά γράμματα για να καταλάβει κανείς τι ακριβώς σημαίνει.

Το cat είναι επίσης άσχημο όνομα μεταβλητής γιατί μπορεί να σημαίνει πολλά διαφορετικά πράγματα: category, catalog ή γάτα.

Διάταξη κώδικα

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

Σκεφτείτεπόσοδυσνόητοείναιένακείμενοσ τοοποιοδενυπΑΡΧΟουνπαράγραφοιήδενξ ΕχωΡΙΖΟυμεσωστάτιςεπιμερουςλέξεισμεκ ενάανάμεσάτουςήαφήνου ΜΕΜΕ γάλα   κενάανάμεσασεπαραγράφους.ακομηχειρ οτεραΑΝΟΙλεΞΕιστουκειμενουδενακολουθο υντιςδιαδεδομενεςσυμβάσειςγραφηςτωνλ εξεωνοσοναφοραστηχρήσηπεζωνκεφαλαι ωνη ακμη κ ορθγρφς.

Στοίχιση (indentation) και τοποθέτηση αγκίστρων (brace style)

Κάθε εντολή πρέπει να βρίσκεται σε ξεχωριστή γραμμή. Για παράδειγμα,

ΓΡΑΨΤΕ:	ΜΗ ΓΡΑΦΕΤΕ:

`if (userFound) {`
`  return 1;`
`}`
	`if (userFound) return 1;`

ΚΑΝΟΝΑΣ ΣΤΟΙΧΙΣΗΣ: Οι εντολές που ανήκουν στο σώμα μιας συνάρτησης ή σύνθετης εντολής πρέπει να ξεκινούν 4 κενά (ή 1 tab μήκους 4) πιο δεξιά από τη στήλη στην οποία βρίσκεται η συνάρτηση ή σύνθετη εντολή:

```
if (tax_applies) {
    amount *= (1 + TAX_PERCENTAGE);
}
```

Οι συναρτήσεις και οι σύνθετες εντολές (π.χ. while, if) χρησιμοποιούν άγκιστρα για να οριοθετήσουν το "σώμα" τους, δηλαδή τις εντολές που εκτελούνται όταν καλείται η συνάρτηση ή όταν είναι αληθής η συνθήκη.

ΣΥΣΤΑΣΗ: Συνίσταται να χρησιμοποιείτε άγκιστρα ακόμη και όταν το σώμα αποτελείται από μία μόνο εντολή γιατί:

- Μειώνεται η πιθανότητα λάθους αν αργότερα προστεθούν κι άλλες εντολές στο σώμα
- Είναι άμεσα ξεκάθαρο πού τερματίζει το σώμα

Υπάρχουν διάφοροι αποδεκτοί τρόποι τοποθέτησης αγκίστρων. Οι δύο πιο δημοφιλείς είναι:

1TBS (One True Brace Style)

Το σώμα κάθε σύνθετης εντολής περικλείεται σε άγκιστρα ακόμη κι αν περιέχει μία μόνο εντολή.

Το αριστερό άγκιστρο { βρίσκεται στην ίδια γραμμή με την επικεφαλίδα της συνάρτησης ή σύνθετη εντολή.

Το δεξί άγκιστρο } βρίσκεται σε νέα γραμμή, στην ίδια στήλη με την αρχή της επικεφαλίδας της συνάρτησης ή σύνθετης εντολής, χωρίς τίποτα άλλο στη γραμμή (παρατηρήστε ότι το else στο παράδειγμα βρίσκεται στην επόμενη γραμμή από το } που αντιστοιχεί στο σώμα της if.

```
int charExists(char *str, char ch) {

    int i;

    for (i=0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            return 1;
        }
    }
    return 0;
}
```

```
if (x > 0) {
    printf("Positive\n");
}
else {
    printf("Negative\n");
}
```

Allman style

Παρεμφερές με το 1TBS αλλά το αριστερό άγκιστρο { βρίσκεται στην επόμενη γραμμή από την επικεφαλίδα της συνάρτησης ή σύνθετη εντολή, και στην ίδια στήλη.

Οι γραμμές με τα { και } δεν περιλαμβάνουν τίποτα άλλο.

```
if (x > 0)
{
    printf("Positive\n");
}
else
{
    printf("Negative\n");
}
```

Κενά και κενές γραμμές

Αφήνετε ένα κενό πριν το αριστερό άγκιστρο που οριοθετεί το σώμα μια συνάρτησης ή σύνθετης εντολής αν χρησιμοποιείτε τη σύμβαση 1TBS.

Αφήνετε ένα κενό μετά από κάθε κόμμα (π.χ. σε δηλώσεις πολλαπλών μεταβλητών ίδιου τύπου).

ΓΡΑΨΤΕ:

```
double principal, compoundInterest, annualRate, numPeriods, time;
```

ΜΗ ΓΡΑΦΕΤΕ:

```
double principal,compoundInterest,annualRate,numPeriods,time;
```

ΚΑΝΟΝΕΣ:

- 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 τμήμα που διαβάζει τα δεδομένα και στο τμήμα που τα επεξεργάζεται. Το πρόγραμμα σας δεν πρέπει να είναι "wall of text".

- Αφήνετε πάντα μία κενή γραμμή μετά το τελικό άγκιστρο } μιας συνάρτησης, ώστε να ξεχωρίζει πού τελειώνει η μία και πού ξεκινά η επόμενη.

- Μην υπερβάλλετε. Μια κενή γραμμή κάθε μερικές γραμμές κώδικα είναι καλή. Όταν υπάρχουν πολλές διαδοχικές γραμμές ή κάθε δεύτερη γραμμή είναι κενή, αλλοιώνεται η ροή του κώδικα και μειώνεται αισθητά η ποσότητα κώδικα που χωρά στην οθόνη.

- Αφήνετε ένα κενό πριν και μετά από τελεστές σε εκφράσεις, ειδικά πριν και μετά τον τελεστή ανάθεσης =. Για παράδειγμα,

ΓΡΑΨΤΕ:	ΜΗ ΓΡΑΦΕΤΕ:
`if (i < 0 || i >= SIZE)` ✓	`if (i<0||i>=SIZE)` 👎

Αποφεύγετε γραμμές μακρύτερες από 100 χαρακτήρες γιατί είναι δύσκολο να διαβαστούν. Ιδανικά, μην υπερβαίνετε τους 80. Αν μια έκφραση είναι πολύ μεγάλη, προσπαθήστε να τη χωρίσετε σε δύο εκφράσεις. Ομοίως αν έχετε δηλώσεις πολλών μεταβλητών ίδιου τύπου. Για παράδειγμα,

ΓΡΑΨΤΕ: ✓

```
double principal, interest, annualRate, numPeriods, time;
double taxRate, totalTax;
```

ΜΗ ΓΡΑΦΕΤΕ: 👎

```
double principal, interest, annualRate, numPeriods, time, taxRate,
totalTax;
```

Αν μια συνάρτηση έχει πολλές παραμέτρους, συνεχίστε τη στην επόμενη γραμμή όπως παρακάτω:

```
drawRectangle (lowerLeftX, lowerLeftY, length, height, slant,
               solidStyle, fillColor, lineColor);
```

Σημειώστε όμως πως πρέπει να αποφεύγετε περισσότερες από 4 παραμέτρους σε μια συνάρτηση. Ένας τρόπος να επιτευχθεί αυτό είναι ομαδοποιώντας σχετικές παραμέτρους σε struct. Στο συγκεκριμένο παράδειγμα, αυτό θα μπορούσε να γίνει με το ζευγάρι παραμέτρων `lowerLeftX, lowerLeftY` καθώς και με το ζευγάρι `length, height`.

Αν μία σύνθετη συνθήκη έχει πολλά τμήματα, μπορείτε ή να τη σπάσετε όπως φαίνεται παρακάτω:

```
if ( (in.day < out.day) || (in.day == out.day && in.hour < out.hour)
   || (in.day == out.day && in.hour == out.hour) )
```

Η επιλογή να "σπάσει" η γραμμή έτσι ώστε ο τελεστής να βρίσκεται στη γραμμή της επόμενης κάνει πιο σαφές το γεγονός ότι πρόκειται για συνέχεια από την προηγούμενη γραμμή.

Ακόμη καλύτερη επιλογή όμως είναι να δημιουργήσετε μια συνάρτηση που κάνει τον έλεγχο που σας ενδιαφέρει και να καλέσετε τη συνάρτηση:

```
if (isValidTimePeriod(in, out))
```

Σχόλια

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 γνωρίζει ο αναγνώστης φαίνεται ήδη μέσα από τον κώδικα. Άλλοι, ανάμεσά τους κι εμείς, υποστηρίζουν ότι κάποια σχόλια είναι απαραίτητα για την κατανόηση του κώδικα από αυτούς που καλούνται να τον διορθώσουν ή να τον μεταβάλλουν σε μελλοντικό χρόνο. Όλοι πάντως συμφωνούν ότι κακογραμμένα σχόλια είναι χειρότερα από καθόλου σχόλια.

Τα σχόλια πρέπει να είναι συνοπτικά και να εξηγούν το σκοπό ενός κομματιού κώδικα.

Τα σχόλια δεν πρέπει απλά να επαναλαμβάνουν αυτό που κάνει κάθε εντολή.

Πού τοποθετούνται σχόλια και τι περιέχουν

- Στην αρχή κάθε αρχείου .c
    - Περιγράφει συνοπτικά τη λειτουργία του προγράμματος
    - (Προαιρετικά) Τα ονόματα των κύριων προγραμματιστών
    - (Προαιρετικά) Ημερομηνία τελευταίας τροποποίησης
    - (Προαιρετικά) Πληροφορίες για τυχόν προγραμματισμένες βελτιώσεις του κώδικα.
- Πριν από κάθε ορισμό συνάρτησης
    - Περιγράφει συνοπτικά τη λειτουργία της συνάρτησης
    - Σημασία παραμέτρων
    - Σημασία επιστρεφόμενης τιμής
    - (Προαιρετικά) Τυχόν παραδοχές
    - (Προαιρετικά) Τυχόν παρενέργειες
- Ενδιάμεσα στον κώδικα (μόνο όταν είναι απαραίτητο, δείτε τα παραδείγματα)
    - Σύντομο σχόλιο δίπλα σε μια εντολή
    - Σχόλιο 2-3 γραμμών πριν από ένα κομμάτι κώδικα που υλοποιεί μια δυσνόητη λειτουργία

Παράδειγμα σχολίου προγράμματος

```
/* contacts.c
 * Manage a list of contacts (add, remove, find, display)
 * Created: Jan 2024
 * Last updated: Mar 2024, Added the option to display contacts
 *               by category
 * To do: Add merge functionality
 */
```

Παράδειγμα σχολίου συνάρτησης

```
/* add_contact
 * Adds a new contact to the contact database.
 * Created: Jan 2024
 * Parameters:
 *    contactT *db - Reference to the contact database
 *    nameT name   - The name of the contact
 *    char *phone  - The phone number of the contact
 * Returns:
 *    0 if contact added successfully
 *    1 if contact already exists
 */
```

Παράδειγμα ενδιάμεσου σχολίου

```
/* insert contact in sorted order by last name*/
```

Ακολουθεί κώδικας που βρίσκει πού πρέπει να εισαχθεί η νέα επαφή ώστε όλες οι επαφές να παραμένουν ταξινομημένες αλφαβητικά με βάση το επώνυμο.

Παραδείγματα κακών και καλών σχολίων

Συνάρτηση προς σχολιασμό

```
void lower (char *str) {
    int i;
    for (i=0; str[i] != '\0'; i++) {
        char c = str[i];
        if (c >= 'A' && c <= 'Z') {
            str[i] = c + 'a' - 'A';
        }
    }
}
```

Παράδειγμα μη-αποτελεσματικού σχολίου:

/* Η συγκεκριμένη συνάρτηση παίρνει ως όρισμα ένα δείκτη. Σε αυτή τη συνάρτηση ελέγχουμε αν το c είναι ανάμεσα σε A-Z. Τότε το μετατρέπει σε μικρό παίρνοντας τον ASCII κωδικό του και το καταχωρεί στο str[i]*/

Γιατί δεν είναι αποτελεσματικό;

Διότι περιγράφει λέξη προς λέξη τι κάνει κάθε μία γραμμή κώδικα, χωρίς να προσθέτει ουσιαστική πληροφορία. Δεν εξηγεί το σκοπό του κώδικα, μόνο το περιεχόμενο του. Με άλλα λόγια απλώς "διαβάζει φωναχτά" αυτό που ήδη βλέπει ο αναγνώστης του κώδικα.

Είναι σαν τη διπλανή εικόνα όπου η δεύτερη ταμπέλα ("this is a stop sign") απλώς περιγράφει αυτό που φαίνεται καθαρά στην πρώτη ταμπέλα.

Παράδειγμα αποτελεσματικού σχολίου:

```
/* Παίρνει ως παράμετρο μια συμβολοσειρά και μετατρέπει όλα τα κεφαλαία της γράμματα σε μικρά. */
```

Γιατί είναι αποτελεσματικό;

Διότι περιγράφει την ουσία της λειτουργίας της συνάρτησης χωρίς να μπαίνει σε λεπτομέρειες για το πώς υλοποιείται.

Καλά ονόματα μεταβλητών και χρήση συναρτήσεων ή βοηθητικών τύπων συχνά αρκούν για να είναι κατανοητός ο κώδικας:

ΝΑΙ

```
if (clientStatus == NEW) {
```

ΟΧΙ

```
/* if this is a new client */
if (clientFlag == 0) {
```

ΣΙΓΟΥΡΑ ΟΧΙ

```
/* if the client flag is zero */
if (clientFlag == 0) {
```

Στο παραπάνω παράδειγμα παρουσιάζονται τρείς τρόποι να ελεγχθεί αν ένας πελάτης είναι νέος. Ο καλύτερος τρόπος να γίνει αυτό είναι έχοντας ορίσει ένα τύπο enum που αναπαριστά την κατάσταση ενός πελάτη (π.χ. με τιμές NEW, EXISTING) και να ελέγξουμε αυτό. Παρατηρήστε πως το όνομα της μεταβλητής clientStatus περιγράφει ακριβώς αυτό που θέλουμε και το η τιμή NEW είναι σαφής.

Στα παραδείγματα της κάτω γραμμής δεν είναι σαφές τι ελέγχεται γιατί τόσο το όνομα clientFlag όσο και η τιμή 0 δεν είναι ξεκάθαρο σε τι αναφέρονται. Επιπλέον, είναι εύκολο να γίνει λάθος από τον προγραμματιστή (μπορεί το 0 να σημαίνει ότι ο πελάτης δεν είναι νέος).

Ειδικά το τελευταίο παράδειγμα είναι ιδιαίτερα άστοχο γιατί ούτε ο κώδικας είναι σαφής ούτε το σχόλιο δίνει χρήσιμη πληροφορία.

ΠΡΟΣΟΧΗ:

- Η στοίχιση σχολίων πρέπει να ακολουθεί πάντα τη στοίχιση του κώδικα που σχολιάζουν.
- Τα σχόλια εισάγονται ακριβώς πριν από τον κώδικα που σχολιάζουν.
- Ένα σχόλιο μπορεί να εισαχθεί δεξιά από τον κώδικα που σχολιάζει, αλλά μόνο αν είναι πολύ συνοπτικό (3-4 λέξεις) και δεν υπερβαίνει την 100ή στήλη. Επίσης, δεν πρέπει να βρίσκεται μακριά από τον κώδικα που σχολιάζει.
- Σχόλια για την επεξήγηση μεταβλητών εισάγονται μόνο αν δίνουν επιπλέον πληροφορίες για την τιμή που αποθηκεύεται σε αυτές κι εφόσον έχουν ήδη επιλεγεί περιγραφικά ονόματα.

ΓΡΑΨΤΕ:

```
double distance; /* in kilometers */
```

ΜΗ ΓΡΑΦΕΤΕ:

```
double d;
```

Χρήση συναρτήσεων

Η διάσπαση του κώδικα σε μικρές, σαφώς ορισμένες συναρτήσεις βελτιώνει σημαντικά την αναγνωσιμότητα και συντηρησιμότητα. Κάθε συνάρτηση πρέπει να έχει μία και μόνο ευθύνη και να έχει περιγραφικό όνομα που αποκαλύπτει το σκοπό της.

Εξετάστε τη συνάρτηση main στο παρακάτω παράδειγμα:

```
int main (int argc, char *argv[]) {
    double grades[NUM_COURSES] = {0};
    double average;
    read_grades(grades, NUM_COURSES);
    average = calc_average(grades, NUM_COURSES);
    print_letter_grade(average);
    return 0;
}
```

Μια γρήγορη ματιά αρκεί για να καταλάβει ο αναγνώστης τι ακριβώς κάνει αυτό το πρόγραμμα. Επιπλέον, διευκολύνεται η προσθήκη πρόσθετων λειτουργιών όπως η εκτύπωση κάποιου ιστογράμματος, η εύρεση του μεγαλύτερου βαθμού, η καταμέτρηση των προβιβάσιμων βαθμών κτλ.