Ανάπτυξη Λογισμικού στην Πράξη – Πρακτικές οδηγίες και χρήσιμες υποδείξεις

Συστάσεις για τη δομή

Ένα άριστα "δομημένο" τμήμα προγράμματος (π.χ. μία συνάρτηση) μπορούμε να το διαβάσουμε από την αρχή ως το τέλος χωρίς να χρειαστεί να κάνουμε άλματα προς τα πίσω ή προς τα εμπρός σε διάφορα σημεία και στο τέλος της ανάγνωσης έχουμε καταλάβει τι κάνει.

Αναπτύσσουμε το πρόγραμμά μας υποδιαιρώντας το σε ένα κύριο πρόγραμμα και υποπρογράμματα (συναρτήσεις) που καλούνται από αυτό.

Οι καλούμενες συναρτήσεις αναπτύσσονται και αυτές με ανάλογο τρόπο και παρόμοια λογική "δόμησης". Σε τελική ανάλυση έχουμε δηλώσεις (μεταβλητών, δομών δεδομένων, συναρτήσεων), εντολές ανάθεσης (απόδοσης τιμής),και "προγραμματιστικές δομές", δηλαδή επαναλήψεις και επιλογές. Καμιά φορά μπορεί να έχουμε και αναδρομικές κλήσεις.

Οι συναρτήσεις και δομές πρέπει να έχουν στην ιδανική περίπτωση, μία είσοδο στην "κορυφή" και μία έξοδο στη "βάση".

Για τις επαναλήψεις προτιμούμε το for ή το while και αποφεύγουμε το do...while γιατί στο τελευταίο εκτελείται οπωσδήποτε τουλάχιστον μία επανάληψη, πράγμα που ευνοεί τα bugs.
Το for καλύτερα να είναι "κλασσικό" δηλαδή ο έλεγχος της επανάληψης μέσα στο for και οι επαναλαμβανόμενες ενέργειες από κάτω, μέσα στο σώμα του.

Για τις πολλαπλές επιλογές προτιμούμε το if...else if Τα άλλα ας τα αποφεύγουμε αν δεν υπάρχει ιδιαίτερος λόγος.

Κάνουμε ένα ή περισσότερα αρχεία επικεφαλίδας τα οποία κάνουμε #include όπου χρειαστεί. Αυτά έχουν την εξής δομή:

```
#ifndef HEADERFILENAME_H
#define HEADERFILENAME_H
....
#endif
```

Εκεί μέσα μπορεί να υπάρχουν τα εξής:

- άλλες οδηγίες #include
- οδηγίες ορισμού συμβολικών σταθερών #define ή "απο-ορισμού" τους #undef
- δηλώσεις εξωτερικών μεταβλητών
- δηλώσεις δομών δεδομένων struct
- ονομασίες ή μετονομασίες τύπων typedef

κλπ.
Όπουδήποτε μέσα στη main και τις υπόλοιπες συναρτήσεις εμφανίζεται κάτι από τα παραπάνω δηλωμένα ή ορισμένα αντικείμενα, τύπους κλπ εκεί κάνουμε include και το αντίστοιχο αρχείο επικεφαλίδας.

Αποφεύγουμε εξωτερικές μεταβλητές και προτιμούμε μεταβίβαση πληροφορίας μέσω ορισμάτων σε συναρτήσεις. Αν χρειαστούν εξωτερικές μεταβλητές τις βάζουμε σε αρχεία επικεφαλίδας.

Αποφεύγουμε να ορίζουμε συναρτήσεις με προεπεξεργαστικές οδηγίες ορισμού διότι αποτελούν πηγές σφαλμάτων, αλλά κατ' εξαίρεση υπάρχει μία ιδιαίτερα χρήσιμη οδηγία ορισμού συνάρτησης:

```
#define NELEMS(array) ( sizeof(array) / sizeof(array[0]) )
```

Πρόληψη λαθών

Συνηθισμένα λάθη – Προσοχή στα παρακάτω!

- Τα άγκιστρα που ανοίγουν πρέπει να κλείνουν!
- Τα εισαγωγικά που ανοίγουν πρέπει να κλείνουν!
- Οι παρενθέσεις που ανοίγουν πρέπει να κλείνουν!

Πώς να αποφεύγουμε λάθη σχετικά με τα παραπάνω (ανοιχτά άγκιστρα, εισαγωγικά, παρενθέσεις):
1) τα γράφουμε "ιεραρχικά", δηλαδή πρώτα γράφουμε το ζευγάρι με τα άγκιστρα ή τις παρενθέσεις και μετά γράφουμε μέσα τους αυτά που πρέπει.
2) γράφουμε με ένα σταθερό και ομοιόμορφο ύφος, κατά προτίμηση με εσοχές ώστε να ξεχωρίζουν τα διάφορα "επίπεδα" ένθεσης

Προσοχή και στα ελληνικά ερωτηματικά στο τέλος! Μην τα ξεχνάτε!
Πού τα βάζουμε και πού **δεν** τα βάζουμε:
- Τα βάζουμε στο τέλος κάθε δήλωσης (μεταβλητών, πρωτότυπων συναρτήσεων, δομών ή ενώσεων δεδομένων, ονομασιών τύπων)
- Τα βάζουμε στο τέλος κάθε εντολής ανάθεσης X = κάτι.
- Τα βάζουμε και στο τέλος της εντολής do...while(...);

Αλλά **δεν** τα βάζουμε μετά από τις παρενθέσεις στις εντολές while(...) ή for(...) ούτε στο άγκιστρο } που κλείνει το σώμα τους! Αν γράψουμε for(...); ή while(...); είναι *συντακτικά* σωστό αλλά συνήθως δεν είναι αυτό που θέλουμε! Υπάρχουν και περιπτώσεις που μπορεί να το θέλουμε αλλά συνήθως όχι! Γι' αυτό προσοχή!
Αν πάλι, γράψουμε π.χ.for(...) { ... ... }; , δηλαδή με ; μετά από το άγκιστρο, αυτό είναι λάθος και θα "χτυπήσει" στο μεταγλωττιστή.
Όμως, στις εντολές που βρίσκονται *μέσα* στο σώμα (στα άγκιστρα) των for και while εκεί είναι σωστό και επιβάλλεται να βάζουμε το ελληνικό ερωτηματικό στο τέλος.

Τα ίδια ακριβώς ισχύουν και για τις δομές if.

Παρενθέσεις είναι καλό να βάζουμε όπου έχουμε αμφιβολίες για την προτεραιότητα για να έχουμε το κεφάλι μας ήσυχο. Παραπάνω παρενθέσεις δε βλάπτουν. Λιγότερες μπορεί και να βλάπτουν.

Διορθώσεις (debugging)
Όταν το πρόγραμμα τυπώνει λάθος αποτελέσματα, πρώτα κυττάζουμε μήπως έχουμε λάθος κωδικό μορφοποίησης σε κάποια εντολή printf ή fprintf, π.χ. θέλουμε να τυπώσουμε ένα float και αντί για %f βάζουμε %d ή θέλουμε να το αλλάξουμε και να τυπώσουμε ένα double αλλά ξεχνάμε να αλλάξουμε το %f σε %lf.

segmentation fault: οφείλεται σε αναφορά σε περιοχή της μνήμης που δεν έχει δεσμευτεί, π.χ. όταν ένας αριθμοδείκτης ξεπερνά τα όρια ενός πίνακα, για παράδειγμα, έχουμε έναν πίνακα με 5 στοιχεία, example[5] και ζητάμε το έκτο: i = 5; ...; x = example[i];
Αν δεν είναι αυτό, δείτε μήπως κάτι δεν πάει καλά με κάποιους δείκτες (pointers). Οι δείκτες πρέπει να αρχικοποιούνται παίρνοντας μία διεύθυνση άλλης μεταβλητής ή με την εντολή malloc ή calloc αλλιώς είναι NULL και είναι λάθος να τους χρησιμοποιήσουμε! Σε ειδικές περιπτώσεις μπορεί να πάρουν τιμή και από άλλες συναρτήσεις - βλ. άνοιγμα και κλείσιμο αρχείων.

Κάποιες Βασικές Αρχές
- *Απλότητα*: προγράμματα μικρά και εύχρηστα
- *Σαφήνεια*: προγράμματα κατανοητά τόσο από τους ανθρώπους όσο και από... τις μηχανές!
- *Γενικότητα*: να δουλεύουν σε πολλές διαφορετικές καταστάσεις και να είναι ευπροσάρμοστα σε νέα δεδομένα
- *Αυτοματοποίηση*: να κάνει η μηχανή τη δουλειά μας.

Αυτοί οι κανόνες δεν είναι απόλυτοι` καμιά φορά ο ένας έρχεται σε σύγκρουση με τους άλλους και πρέπει να εξισορροπήσουμε τα πράγματα (προσωπική άποψη: η σαφήνεια είναι σε πρώτη προτεραιότητα, η γενικότητα σε δεύτερη, η συντομία σε τρίτη και η αυτοματοποίηση σε τέταρτη).

Ύφος γραφής
Το ύφος δεν είναι διόλου αμελητέος παράγοντας! Συνεπές και ομοιόμορφο ύφος κάνει τον κώδικα πιο ευανάγνωστο και κατανοητό και συνεπώς διορθώνεται, συντηρείται και αναβαθμίζεται πιο εύκολα.

Ονόματα - Προτεινόμενοι κανόνες:
Μεγάλα και περιγραφικά ονόματα για μεταβλητές μεγάλης εμβέλειας, μικρά ονόματα για τοπικές μεταβλητές. Το ίδιο περιγραφικά να είναι και τα ονόματα συναρτήσεων και δομών δεδομένων. Όσον αφορά τοπικές μεταβλητές, συνήθη ονόματα για μετρητές: i, j, δείκτες: p, q, αλφαριθμητικά: s, t.
Άλλοι προτεινόμενοι κανόνες:

- να αρχίζουν από p όταν πρόκειται για δείκτες,
- κεφαλαία αρχικά γράμματα για εξωτερικές, συναρτήσεις και δομές δεδομένων
- όλα κεφαλαία για συμβολικές σταθερές
- όταν είναι σύνθετες λέξεις να χρησιμοποιείται σύμβαση "καμήλας" (δηλαδή το πρώτο γράμμα του δεύτερου συνθετικού να είναι κεφαλαίο) ή σύνθεση με _.
- Προσοχή! όποια σύμβαση κι αν επιλεγεί να χρησιμοποιείται με *συνέπεια!* Η συνέπεια είναι πιο σημαντική από κάποιο συγκεκριμένο ύφος.
- Ενεργητική φωνή συν αντικείμενο για συναρτήσεις που επιστρέφουν τιμές ή εκτελούν ενέργειες, και "είναι-κάτι" ειδικά για επιστροφή λογικής τιμής, π.χ. getThis, setThat, readNumber, writeResult, returnValue – αυτά τα ονόματα είναι σαφή και κατάλληλα για συναρτήσεις επειδή καταλαβαίνουμε αμέσως τι κάνουν.
- Τα ονόματα, ιδίως των συναρτήσεων, να είναι ακριβή και να υποδηλώνουν αυτό που κάνουν.

Παραστάσεις

- Γράψτε εκφράσεις όπως θα τις διαβάζατε. Αν είναι δυνατό, αποφύγετε αρνήσεις, αντικαταστήστε με ισοδύναμες εκφράσεις
- Αν η έκφραση αρχίζει και γίνεται πολύπλοκη, οι παρενθέσεις βοηθούν ακόμη κι αν δεν είναι απολύτως απαραίτητες!
- Σπάστε πολύπλοκες εκφράσεις σε απλούστερες δομές.
- Σαφήνεια: εδώ έχουμε και μια εξαίρεση από τον κανόνα να γράφουμε σύντομα προγράμματα` αν ένα κομμάτι κώδικα είναι πιο σαφές αν το γράψουμε αναλυτικά, τότε δεν πειράζει να γράψουμε μερικές γραμμές παραπάνω.
- Προσοχή στις παράπλευρες συνέπειες! Προσοχή π.χ. με τους τελεστές αύξησης και μείωσης, γιατί μπορεί να οδηγήσουν σε διαφορετικές τιμές από αυτές που περιμένουμε (για να το πούμε απλά: να τους χρησιμοποιούμε αλλά να μην το παρακάνουμε).

Ιδίωμα
Συνέπεια στο ύφος! (αλλά δεν αλλάζουμε το ύφος του άλλου όταν διαβάζουμε το πρόγραμμά του!)
Να βρεθεί το λάθος:

```
if (month == FEB) {
      if (year%4 == 0)
            if (day > 29)
                  legal = FALSE;
      else
            if (day > 28)
                  legal = FALSE;
}
```

Στην πραγματικότητα, το τελευταίο if είναι "ένα σώμα" με το else, δηλαδή:

```
      else if (day > 28)
```

άρα το σωστό είναι :

```
if (month == FEB) {
      if (year%4 == 0)  {
            if (day > 29)
                  legal = FALSE;
      } else {
            if (day > 28)
                  legal = FALSE;
      }
}
```

Κανόνας: όταν ένα if έρχεται αμέσως μετά από ένα άλλο χρησιμοποιούμε άγκιστρα.
Ιδίωμα δέσμευσης μνήμης για αλφαριθμητικά:

```
char *p, buf[256];
p = malloc(strlen(buf)+1);
strcpy(p, buf);
```

Το +1 χρειάζεται για το χαρακτήρα τερματισμού. Αν λείπει, κάτι δεν πάει καλά.
Για πολλαπλές αποφάσεις προτιμούμε το if...else if... Στο τέλος, είναι καλή ιδέα να βάλουμε και ένα else με ένα μήνυμα λάθους ακόμη και αν δε μας φαίνεται απαραίτητο.

Να ξαναγραφεί το παρακάτω:

```
for (k = 0; k++ < 5; x += dx)
      scanf("%f", &dx);
```

Απάντηση:

```
for (k = 0; k < 5; k++) {
      scanf("%f", &dx);
       x += dx
}
```

Μαγικοί αριθμοί

- Ορίστε με κάποιο όνομα τις σταθερές που χρησιμοποιείτε συχνά!
- Κατά προτίμηση όχι με #define αλλά με const ή enum. *Προσοχή! οι τιμές const δε μπορούν να χρησιμοποιηθούν ως διαστάσεις πινάκων, αλλά οι enum μπορούν!*
- Αν χρησιμοποιούμε sizeof(πίνακας ή στοιχείο πίνακα) αυτό το πρόγραμμα "αντέχει" σε αλλαγές.

Σχόλια

- Όχι σχόλια για τα προφανή!
- Αν τα ονόματα των μεταβλητών και συναρτήσεων κάνουν προφανές το τι γίνεται συνήθως δε χρειάζεται σχόλιο
- Σχόλια για τα σκοτεινά σημεία
- Σχόλια για να συνοψίσουν αυτό που γίνεται διάσπαρτο σε μεγάλη έκταση του κώδικα (π.χ. σαν επικεφαλίδες παραγράφων σε ένα κείμενο)
- Εισαγωγικά σχόλια στους ορισμούς συναρτήσεων (τι κάνουν)
- Σχόλια για μέρη του προγράμματος που εμφανίζονται από το πουθενά
- Σχόλια που να εξηγούν γιατί χρειάζεται να κάνουμε κάτι που δεν είναι προφανές ότι χρειάζεται σε ένα σημείο του κώδικα
- Αν χρειάζονται πάρα πολλά επεξηγηματικά σχόλια για ένα τμήμα του κώδικα, τότε ίσως πρόκειται για κακογραμμένο κώδικα που πρέπει να ξαναγραφεί!
- Ενημερώνετε τα σχόλια καθώς εξελίσσεται ο κώδικας ώστε να συμφωνούν μαζί του! (Έχουμε χάσει ένα μήνα σε project ανάπτυξης επιστημονικού λογισμικού επειδή ένα σχόλιο δεν ήταν ενημερωμένο με αποτέλεσμα να χρησιμοποιείται κάποια συνάρτηση με λάθος τρόπο δίνοντας, φυσικά και λάθος αποτελέσματα που δε μπορούσαμε να εξηγήσουμε!)