ΕΝΟΤΗΤΑ IV

Αξιολόγηση - Τεκμηρίωση

Σκοπός ενότητας:

Να αποκτήσουμε ικανότητες τεκμηρίωσης και αξιολόγησης ενός προγράμματος.

Ειδικοί σκοποί:

- ❑ Να μπορούμε να αιτιολογούμε με πληρότητα και με ακρίβεια τη μεθοδολογία που εφαρμόσαμε για την επίλυση του προβλήματος.
- ❑ Να μπορούμε να κρίνουμε και να αξιολογούμε τα αποτελέσματα της εργασίας μας.
- ❑ Να επιδιώκουμε τη συγκριτική θεώρηση των προγραμμάτων μας.

Περιεχόμενα:

- Τεκμηρίωση του Προγράμματος
- Αξιολόγηση, Βελτιστοποίηση

ΚΕΦΑΛΑΙΟ 18
ΤΕΚΜΗΡΙΩΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Σκοπός κεφαλαίου:

Να γνωρίσουμε πώς γίνεται η τεκμηρίωση ενός προγράμματος συμπληρώνοντας τον αντίστοιχο φάκελο.

Ειδικοί σκοποί:

- Να κατανοήσουμε τη σημασία των σχολίων μέσα σε ένα πρόγραμμα.
- Να κατανοήσουμε τη σημασία της χρήσης μνημονικών ονομάτων για χρήση σταθερών, μεταβλητών, διαδικασιών και συναρτήσεων.

18.1. Η ΕΝΝΟΙΑ ΤΗΣ ΤΕΚΜΗΡΙΩΣΗΣ

Από τη στιγμή που ένα πρόγραμμα αρχίζει να χρησιμοποιείται, υπάρχουν πιθανότητες να τροποποιηθεί. Η τροποποίηση μπορεί να αφορά είτε σε κάποια διόρθωση λάθους που εμφανίζεται στη διάρκεια της χρήσης του προγράμματος είτε σε νέες απαιτήσεις των χρηστών που δημιουργούνται στην πορεία. Για ό,τι αφορά στις αλλαγές, πρέπει να μπορούν να επαναληφθούν οι φάσεις της επίλυσης του προβλήματος και της υλοποίησης της λύσης. Οι διαδικασίες αυτές αποτελούν τη **συντήρηση (maintenance)** του προγράμματος.

Η τεκμηρίωση είναι ένα απαραίτητο βήμα στον κύκλο ανάπτυξης του προγράμματος. Περιλαμβάνει την περιγραφή του προβλήματος, την αναπαράσταση του αλγορίθμου με λογικά διαγράμματα ή ψευδοκώδικα, πίνακες με δεδομένα ελέγχου και αποτελέσματα, τεχνικές λεπτομέρειες, οδηγίες για το χρήστη, οδηγίες για την εγκατάσταση του προγράμματος κλπ. Η σύνταξη αυτών των εντύπων θα πρέπει να αποτελεί μέρος του καθενός από τα στάδια προγραμματισμού και όχι να εκτελείται στο τέλος. Με τον τρόπο αυτό, η τεκμηρίωση θα βοηθήσει τον προγραμματιστή στον προγραμματισμό. Η καλή τεκμηρίωση θα βοηθήσει επίσης τη συντήρηση ενός προγράμματος, κατά τη διάρκεια του κύκλου ζωής του, και τη χρησιμοποίησή του από άλλους προγραμματιστές ή χρήστες.

Η σύνταξη των εντύπων της τεκμηρίωσης θα πρέπει να γίνεται ως μέρος καθενός από τα στάδια προγραμματισμού και όχι στο τέλος, ως αποτέλεσμα ωριμότερης σκέψης.

Πολλά προγράμματα, γραμμένα πριν από αρκετά χρόνια, μπορούν να χρησιμοποιηθούν ακόμη και σήμερα, εφόσον συντηρούνται κανονικά και έχουν

τροποποιηθεί σύμφωνα με τις τελευταίες απαιτήσεις. Ο καλύτερος τρόπος τεκμηρίωσης μπορεί να υλοποιηθεί μέσα από το ίδιο το πρόγραμμα.

18.2. ΣΥΣΤΑΤΙΚΑ ΣΤΟΙΧΕΙΑ ΤΕΚΜΗΡΙΩΣΗΣ

Η τεκμηρίωση είναι το γραπτό κείμενο και τα σχόλια τα οποία γράφονται από τους συγγραφείς του προγράμματος για να διευκολύνουν τους άλλους στην κατανόηση, τη χρήση και τη συντήρησή του.

Ενώ η φάση της ανάπτυξης ενός προγράμματος διαρκεί ένα συγκεκριμένο διάστημα μηνών, τις περισσότερες φορές η φάση της συντήρησης μπορεί να διαρκέσει πολλά χρόνια. Για τους λόγους αυτούς, η φάση της συντήρησης ενός προγράμματος έχει γενικά υψηλότερο κόστος από το κόστος της ανάπτυξής του. Χρειάζεται λοιπόν προσοχή στην αρχική φάση της σχεδίασης, της ανάπτυξης και της υλοποίησης ενός προγράμματος, γιατί αυτό θα μειώσει σημαντικά το κόστος στη διάρκεια της συντήρησής του. Οι φάσεις της επίλυσης του προβλήματος της υλοποίησης και της συντήρησης αποτελούν τον **κύκλο ζωής** του προγράμματος.

Η τεκμηρίωση είναι το γραπτό κείμενο και τα σχόλια τα οποία γράφονται από τους συγγραφείς του προγράμματος για να διευκολύνουν τους άλλους στην κατανόηση, τη χρήση και τη συντήρησή του.

Εκτός από τις φάσεις που αναφέρθηκαν παραπάνω, υπάρχει και μία άλλη, η **τεκμηρίωση (documentation)** η οποία παίζει σημαντικό ρόλο στη διαδικασία του προγραμματισμού. Πολλά προγράμματα στη διάρκεια του κύκλου ζωής τους χρησιμοποιούνται από πολλούς διαφορετικούς χρήστες για μεγάλα χρονικά διαστήματα. Θα πρέπει ο κώδικας να είναι έτσι γραμμένος, ώστε να γίνεται εύκολα κατανοητός.

Η τεκμηρίωση βοηθάει τους προγραμματιστές στη συντήρηση του προγράμματος. Τα χρήσιμα προγράμματα έχουν μεγάλο χρόνο ζωής, μέσα στον οποίον πρέπει να ενημερώνονται και να τροποποιούνται. Το ***καλό προγραμματιστικό στυλ*** διευκολύνει την τεκμηρίωση. Τα στοιχεία αυτά αποδεικνύονται χρήσιμα όχι μόνο για τους άλλους αλλά και για τον ίδιο τον προγραμματιστή που έχει κάνει την κωδικοποίηση, όταν του ζητηθεί ύστερα από κάποιο διάστημα να τροποποιήσει το πρόγραμμά του.

Προγραμματιστικό στυλ

Το προγραμματιστικό στυλ του καθενός διαμορφώνεται σύμφωνα με όσα έχει διδαχθεί τις εμπειρίες του και τις προσωπικές του επιλογές.
Το καλό προγραμματιστικό στυλ πρέπει να:
- είναι απλό,
- έχει συνέπεια,
- είναι εύκολο στο διάβασμα

Το ***καλό προγραμματιστικό στυλ*** και η τεκμηρίωση έχουν σχέση με τον άνθρωπο ο οποίος θα διαβάσει το πρόγραμμα και όχι με τον υπολογιστή. Ο υπολογιστής καταλαβαίνει μόνο τη μετάφραση του προγράμματος σε κώδικα μηχανής. Ο κώδικας μηχανής δεν περιλαμβάνει ούτε το προγραμματιστικό στυλ ούτε τα κείμενα της τεκμηρίωσης, παρά μόνο τη μετάφραση των εντολών. Όταν κανείς μαθαίνει προγραμματισμό, πρέπει να φροντίσει προοδευτικά να αποκτήσει ένα καλό προγραμματιστικό στυλ. Θα πρέπει καταρχήν να έχει ως στόχο να γράψει ένα πρόγραμμα που θα να μπορεί να διαβαστεί και να κατανοηθεί κι από άλλους χρήστες.

Μερικές φορές προσδιορίζεται το στυλ προγραμματισμού από άλλους, πχ από τον εκπαιδευτή μας ή από την εταιρεία που εργαζόμαστε. Όταν τροποποιούμε το πρόγραμμα που έχει γράψει κάποιος άλλος χρησιμοποιούμε το

δικό του στυλ για να υπάρξει συνέπεια στο πρόγραμμα. Η συνέπεια είναι βασικό στοιχείο για ένα πρόγραμμα. Χωρίς αυτήν το πρόγραμμα γίνεται δυσνόητο και μπορεί να οδηγήσει σε λάθος συμπεράσματα. Τελικά ο καθένας αναπτύσσει το προσωπικό του προγραμματιστικό στυλ σύμφωνα με όσα έχει διδαχθεί, τις εμπειρίες του και τις προσωπικές του επιλογές.

Το καλό προγραμματιστικό στυλ περιλαμβάνει

- τα **σχόλια (comments)**
- τη χρήση μνημονικών ονομάτων μεταβλητών (το όνομα της μεταβλητής αντιστοιχεί στο περιεχόμενό της, πχ. οι μεταβλητές ***misthos, kratiseis, vathmos***) **(*meaningful variable names*)**
- τις **εσοχές (Indentation)** στο πρόγραμμα και τις δομές του.

Τα στοιχεία που αναφέραμε βοηθούν τους άλλους να καταλάβουν και να δουλέψουν με το πρόγραμμά μας. Το καλό προγραμματιστικό στυλ πρέπει να είναι απλό, να έχει συνέπεια και να είναι εύκολο στο διάβασμα.

Σχόλια

Τα σχόλια είναι οι πληροφορίες που περιέχονται σε ένα πρόγραμμα για να το καταλαβαίνουν και οι άλλοι που θα το διαβάσουν. Όπου ο κώδικας προγραμματισμού είναι δυσνόητος, προσθέτουμε σχόλια. Βέβαια πρέπει να αποφεύγουμε την υπερβολή. Τα περιττά σχόλια έχουν συνήθως αντίθετα αποτελέσματα.

Μπορούμε να διακρίνουμε τα σχόλια σε:

1. Σχόλια επικεφαλίδας

Εμφανίζονται στην αρχή μετά την **επικεφαλίδα(header)** του προγράμματος. Μπορεί να περιλαμβάνουν το όνομα του προγραμματιστή, την ημερομηνία που γράφτηκε το πρόγραμμα και το σκοπό για τον οποίο γράφτηκε. Αποτελούν εν γένει την εισαγωγή του προγράμματος. Σχόλια επικεφαλίδων πρέπει να υπάρχουν και για τις διαδικασίες ή τις συναρτήσεις που ορίζονται στο πρόγραμμα.

Παράδειγμα σχολίων επικεφαλίδας από το πρόγραμμα που ακολουθεί

```
program  pinakas_min_max_1(input,output);
(*                                                              *)
(* Το πρόγραμμα αυτό                                           *)
(* 1.Κάνει εισαγωγή Δεδομένων σε ένα πίνακα(γέμισμα).          *)
(* 2.Υπολογίζει το ελάχιστο και το μέγιστο των στοιχείων του
(*   πίνακα καθώς και τη θέση που κατέχουν στον πίνακα.       *)
3. Εμφανίζει τα αποτελέσματα στην οθόνη                        *)
(*                                                              *)
(*    Προγραμματιστής: Κ. Ευσταθίου                            *)
(*    Ημερομηνία: ¨     1-1-2000                               *)
(*                                                              *)
(*    Είσοδος  Η εισαγωγή των δεδομένων γίνεται από το
      πληκτρολόγιο *)
(*    Έξοδος: Τα αποτελέσματα εμφανίζονται στην οθόνη          *)
(*    Χρησιμοποιούνται 3 διαδικασίες:  diabase_dedomena,
      bres_elaxisto_megisto,  emfanise_apotelesmata             *)
```

2. Σχόλια ορισμών

Όταν ορίζουμε ή δηλώνουμε μια μεταβλητή, καλό είναι να εξηγούμε μ' ένα σχόλιο το σκοπό της. Καλό είναι τα σχόλια αυτά να εμφανίζονται δεξιά του ορισμού της μεταβλητής.

Παράδειγμα σχολίων ορισμού από το πρόγραμμα που ακολουθεί

```
(*                                                                    *)
  uses wincrt; (*δήλωση που απαιτείται για να
          χρησιμοποιεί το πρόγραμμα οθόνη και
          πληκτρολόγιο  στην Turbo Pascal για Windows          *)
   const
       n=5; (*n=ο αριθμός των στοιχείων του πίνακα             *)
   var
      x:array[1..n] of integer;   (*x=όνομα του πίνακα με
                                     στοιχεία ακεραίους        *)
      m,min,max:integer;   (*min=το ελάχιστο στοιχείο του
                                 πίνακα                        *)

(*max=το μέγιστο στοιχείο του πίνακα                            *)
      pos_min,pos_max:integer;
        (*pos_min=θέση ελαχίστου στοιχ. του πίνακα             *)

        (*pos_max=θέση μεγίστου  στοιχ. του πίνακα             *)
```

3. Σχόλια γραμμής

Τα σχόλια γραμμής εμφανίζονται για να χωρίσουν ένα μεγάλο τμήμα κώδικα σε τμήματα. Αυτά είναι συνήθως ονόματα τμημάτων μέσα από την ιεραρχική, από πάνω προς τα κάτω, σχεδίαση. Μπορούμε όμως να προσθέσουμε και άλλες πληροφορίες, όπως γραμμές με αστερίσκους στην αρχή και στο τέλος της διαδικασίας, σχόλια μετά τα begin, end κλπ.

Παράδειγμα σχολίων γραμμών από το πρόγραμμα που ακολουθεί

```
(*                        ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ                    *)

(*************************************************************)
   procedure  diabase_dedomena;
   (*                                                          *)
   (* διαδικασία αυτή                                         *)
   (* Κάνει εισαγωγή δεδομένων σε ένα πίνακα(γέμισμα)         *)
   (* x[1..n] ο μονοδιάστατος πίνακας, x το όνομα,            *)
   (* n το πλήθος των στοιχείων του.                          *)

     var
       k:integer; (*k=μεταβλητή που καθορίζει τη σειρά του
                     στοιχείου του πίνακα*)
   begin   (*diabase_dedomena*)
     for k:=1 to n do
       begin
         write('δωσε το στοιχείο X[',k,']=');
         readln(x[k]);
       end;
   end;   (*diabase_dedomena*)

(*************************************************)
```

4. Σχόλια εντολών.

Τα σχόλια μπορεί να είναι επεξηγηματικά των εντολών. Έτσι, μπαίνουν δεξιά των εντολών, για όσες εντολές χρειάζονται επεξήγηση.

Παράδειγμα σχολίων εντολών από το πρόγραμμα που ακολουθεί

```
(* ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ                                          *)
for m:=2 to n do  (*  ΑΡΧΗ ΕΠΑΝΑΛΗΨΗΣ                  *)
   begin          (*                                      *)
     if x[m]<min  (*                                      *)
       then       (*                                      *)
         begin    (* αρχή ελαχίστου                                *)
           min:=x[m] ;  (* βρίσκει το ελάχιστο στοιχείο          *)
           pos_min:=m;  (* και τη θέση του στον πίνακα           *)
         end;           (* τέλος ελαχίστου                       *)
         if x[m]>max
           then
             begin          (* αρχή μεγίστου                     *)
               max:=x[m]  ;(* βρίσκει το μέγιστο στοιχείο       *)
               pos_max:=m; (* και τη θέση του στον πίνακα       *)
             end;           (* τέλος μεγίστου                    *)
         end;
(*  ΤΕΛΟΣ ΕΠΑΝΑΛΗΨΗΣ          *)
          (*******************************************)
```

Ακόμη εκτός από τους τέσερεις τύπους σχολίων που αναφέραμε, μπορούμε να χρησιμοποιήσουμε και άλλα σχόλια όπως:

- Μετά τα begin, end σε διαδικασίες, συναρτήσεις, στο κυρίως πρόγραμμα και γενικά όπου βλέπουμε ένα σύνολο εντολών ως μία εντολή, **σύνθετη εντολή (compound statement)**.
- Σειρά από αστερίσκους εμφανίζεται πριν και μετά από κάθε διαδικασία ή συνάρτηση, για να τονίζεται η παρουσία της.
- Σχόλια μετά το end που αντιστοιχούν σε δομές record ή case.

Αναγνωριστές και Λέξεις Κλειδιά

Το πιο βασικό στην επιλογή ενός ονόματος για ένα αντικείμενο ή μια διαδικασία σε ένα πρόγραμμα, είναι το όνομα να μεταφέρει όσο το δυνατόν μεγαλύτερη πληροφορία σχετικά με το περιεχόμενό τους. Ακόμη τα ονόματα θα πρέπει να είναι και ευανάγνωστα. Για παράδειγμα τα ονόματα xwr_gen και xwra_gennisis δίνουν και τα δύο πληροφορίες για το περιεχόμενό τους, όμως το δεύτερο είναι πιο ευανάγνωστο.

Αναγνωριστές(Identifiers) για τύπους, σταθερές ή μεταβλητές μπορεί να είναι ουσιαστικά, ενώ ονόματα για διαδικασίες είναι προτιμότερο να είναι ρήματα. Τα ονόματα των συναρτήσεων μπορεί να είναι ουσιαστικά ή μερικές φορές επίθετα. Ακολουθούν μερικά παραδείγματα:

Μεταβλητές: epwnimo_onoma, topos_katagwgis, vathmos,mesos_oros
Σταθερές: pi, fpa, mikos_pinaka, epitaxynsi_barititas
Διαδικασίες: diabase_dedomena, emfanise_apotelesmata, ypologise
Συναρτήσεις: tetragwniki_riza, power_5, megistos, el;axistos

Αν και οι αναγνωριστές μπορεί να αποτελούνται από περισσότερες από μία λέξεις, καλό είναι να μη γίνεται κατάχρηση. Μεγάλα ονόματα μπορεί να είναι κουραστικά και να κάνουν το πρόγραμμα δύσκολο στο διάβασμα. Πολλές φορές διευκολύνει το πρώτο γράμμα του αναγνωριστή να είναι κεφαλαίο.

Τέλος μπορούν οι **λέξεις κλειδιά (Keywords)** να γράφονται με κεφαλαία πράγμα ή **έντονα (bold)** που τις κάνει να ξεχωρίζουν από τους αναγνωριστές πράγμα που βοηθάει στο να εντοπίζουμε εύκολα τις εντολές σ' ένα μεγάλο πρόγραμμα.

Εσοχές.

Οι εσοχές σε ένα πρόγραμμα δίνουν πρόσθετες δυνατότητες εύκολης ανάγνωσης και εκσφαλμάτωσης του προγράμματος. Όταν οι εσοχές χρησιμοποιηθούν σωστά, η ομαδοποίηση των εντολών είναι φανερή από τον τρόπο που εμφανίζεται ο κώδικας. Γενικά, σε ένα πρόγραμμα χρησιμοποιούμε εσοχές δύο ή περισσοτέρων διαστημάτων κάθε φορά που αναφερόμαστε σε επανάληψη ή σε αντικείμενο χαμηλότερου επιπέδου. Στο παράδειγμα που ακολουθεί, εμφανίζεται ένα τμήμα προγράμματος με δύο τρόπους, χωρίς εσοχές και με εσοχές, για να συγκρίνουμε τους δύο τρόπους παρουσίασης.

Παράδειγμα τμήμα προγράμαμτος χωρίς εσοχές

```
 begin (*bres_elaxisto_megisto*)
(*αρχικές τιμές*)
min:=x[1];
max:=x[1];
pos_min:=1;
pos_max:=1;
(*              *)
for m:=2 to n do        (*  ΑΡΧΗ ΕΠΑΝΑΛΗΨΗΣ                *)
begin                  (*                                  *)
if x[m]<min          (*                                 *)
then               (*                                *)
begin            (* αρχή ελαχίστου                 *)
min:=x[m] ;   (* βρίσκει το ελάχιστο στοιχείο *)
pos_min:=m;   (* και τη θέση του στον πίνακα  *)
end;             (* τέλος ελαχίστου                *)
if x[m]>max
then
begin            (* αρχή μεγίστου                  *)
max:=x[m]   ; (* βρίσκει το μέγιστο στοιχείο  *)
pos_max:=m;   (* και τη θέση του στον πίνακα  *)
end;             (* τέλος μεγίστου                 *)
end;                 (*  ΤΕΛΟΣ ΕΠΑΝΑΛΗΨΗΣ             *)
end; (*bres_elaxisto_megisto*)
```

```
Παράδειγμα το ίδιο τμήμα προγράμματος με εσοχές
begin (*bres_elaxisto_megisto*)
   (*αρχικές τιμές*)
     min:=x[1];
     max:=x[1];
     pos_min:=1;
     pos_max:=1;
     (*                *)
     for m:=2 to n do          (*  ΑΡΧΗ ΕΠΑΝΑΛΗΨΗΣ              *)
       begin                   (*                               *)
         if x[m]<min           (*                               *)
           then                (*                               *)
             begin             (* αρχή ελαχίστου                *)
               min:=x[m] ;     (* βρίσκει το ελάχιστο στοιχείο  *)
               pos_min:=m;     (* και τη θέση του στον πίνακα   *)
             end;              (* τέλος ελαχίστου               *)
         if x[m]>max
           then
             begin             (* αρχή μεγίστου                 *)
               max:=x[m]   ;   (* βρίσκει το μέγιστο στοιχείο   *)
               pos_max:=m;     (* και τη θέση του στον πίνακα   *)
             end;              (* τέλος μεγίστου                *)
         end;                  (*  ΤΕΛΟΣ ΕΠΑΝΑΛΗΨΗΣ             *)

       end; (*bres_elaxisto_megisto*)
```

Το πρόγραμμα που ακολουθεί εμφανίζει όλα τα στοιχεία που είναι απαραίτητα για την τεκμηρίωση που αναφέραμε παραπάνω.

```
PROGRAM  Pinakas_Min_Max_1(INPUT,OUTPUT);
 (*Το πρόγραμμα χρησιμοποιεί ως μονάδα    *)
 (*εισόδου το πληκτρολόγιο και ως μονάδα *)
 (*εξόδου την οθόνη                       *)
 (*εισόδου το πληκτρολόγιο και ως μονάδα *)
 (*εξόδου την οθόνη                       *)
 (*                                       *)
(* Το πρόγραμμα αυτό                      *)
(* 1.Κάνει εισαγωγή δεδομένων σε ένα πίνακα(γέμισμα).          *)
(* 2.Υπολογίζει το ελάχιστο και το μέγιστο των στοιχείων του  *)
(* πίνακα καθώς και τη θέση που κατέχουν στον πίνακα.          *)
(* 3.Εμφανίζει τα αποτελέσματα στην οθόνη                      *)
(**)
(*    Προγραμματιστής: Κ. Ευσταθίου                            *)
(*    Ημερομηνία:      1-1-2000                                *)
(*                                                             *)
(*    Είσοδος: Η εισαγωγή των δεδομενων γίνεται από το *)
(*     πληκτρολόγιο  *)

(*    Χρησιμοποιούνται 3 διαδικασίες:  diabase_dedomena,
bres_elaxisto_megisto, emfanise_apotelesmata                    *)
(*                                                             *)
    USES WINCRT;   (*δήλωση που απαιτείται για να χρησιμοποιεί *)
                   (*  το πρόγραμμα οθόνη και πληκτρολόγιο    *)
                   (*  στην Turbo Pascal για Windows          *)
   CONST
       N=5;      (*N=ο αριθμός των στοιχείων του πίνακα       *)
   VAR
      X:ARRAY[1..N] OF INTEGER; (*X=όνομα του πίνακα με
                                  στοιχεία ακεραίους *)
      M,Min,Max:INTEGER;         (*Min=το ελάχιστο στοιχείο του
                                   πίνακα      *)
                                 (*Max=το μεγιστο στοιχείο του
                                   πίνακα      *)
      Pos_Min,Pos_Max:INTEGER;   (*Pos_Min=θέση ελαχίστου στοιχ.
                                   του πίνακα*)

(*Pos_Max=θέση μεγίστου  στοιχ. του πίνακα*)
```

```
(*************************************************************)
(*                    ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ                     *)
(*************************************************************)
PROCEDURE  Diabase_Dedomena;
   (*                                                         *)
   (* Η διαδικασία αυτή                                       *)
   (*     Κάνει εισαγωγή δεδομένων σε ένα πίνακα(γέμισμα).    *)
   (* X[1..N] ο μονοδιάστατος πίνακας, Χ το όνομα,Ν το πλήθος
    των στοιχείων του. *)
  VAR
    K:INTEGER;   (*Κ=μεταβλητή που καθορίζει τη σειρά του
                    στοιχείου του πίνακα*)
   BEGIN   (*Diabase_Dedomena*)
     FOR K:=1 TO N DO
       BEGIN
         WRITE('δωσε το στοιχείο X[',K,']=');
         READLN(X[K]);
       END;
   END;   (*Diabase_Dedomena*)
(*************************************************************)
(*          ΕΥΡΕΣΗ   ΕΛΑΧΙΣΤΟΥ / ΜΕΓΙΣΤΟΥ                    *)
(*************************************************************)
   PROCEDURE  Bres_Elaxisto_Megisto;
   (*                                                         *)
   (* Η διαδικασία αυτή                                       *)
   (*  Βρίσκει το ελάχιστο και το μέγιστο στοιχείο καθώς και τη
      θέση που κατέχουν στον πίνακα                          *)
   (*                                                         *)
    BEGIN (*Bres_Elaxisto_Megisto*)
   (*αρχικές τιμές*)
     Min:=X[1];
     Max:=X[1];
     Pos_Min:=1;
     Pos_Max:=1;
     (*              *)
     FOR M:=2 TO N DO          (*  ΑΡΧΗ ΕΠΑΝΑΛΗΨΗΣ           *)
       BEGIN                   (*                            *)
         IF X[M]<Min           (*                            *)
           THEN                (*                            *)
             BEGIN             (* αρχή ελαχίστου             *)
               Min:=X[M] ;     (* βρίσκει το ελάχιστο στοιχείο *)
               Pos_Min:=M;     (* και τη θέση του στον πίνακα *)
             END;              (* τέλος ελαχίστου            *)
         IF X[M]>Max
           THEN
             BEGIN             (* αρχή μεγίστου              *)
               Max:=X[M];      (* βρίσκει το μέγιστο στοιχείο *)
               Pos_Max:=M;     (* και τη θέση του στον πίνακα *)
             END;              (* τέλος μεγίστου             *)
       END;                    (*  ΤΕΛΟΣ ΕΠΑΝΑΛΗΨΗΣ          *)
       END; (*Bres_Elaxisto_Megisto*)
(*************************************************************)
(*                  ΕΜΦΑΝΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ                 *)
(*************************************************************)
PROCEDURE Emfanise_Apotelesmata;
   (*                                                         *)
   (* Η διαδικασία αυτή εμφανίζει τα αποτελέσματα στην οθόνη  *)
   (*                                                         *)
      BEGIN   (*Emfanise_Apotelesmata    *)
        WRITELN('Max=', Max:5,' στη θέση ',Pos_Max:3);
        WRITELN('Min=', Min:5,' στη θέση ',Pos_Min:3);
      END;    (*Emfanise_Apotelesmata    *)
(*************************************************************)
(*************************************************************)
(*                        ΚΥΡΙΟ ΠΡΟΓΡΑΜΜΑ                    *)
(*************************************************************)
   BEGIN {κύριο πρόγραμμα}
     Diabase_Dedomena;
     Bres_Elaxisto_Megisto;
     Emfanise_Apotelesmata;
   END. {κύριο πρόγραμμα}
(*************************************************************)
(*************************************************************)
```

Ανακεφαλαίωση

Κατανοήσαμε τη σημασία της τεκμηρίωσης ως μέσο για τη διόρθωση και συντήρηση του προγράμματος. Μάθαμε ότι η τεκμηρίωση πρέπει να γίνεται σε όλη τη διάρκεια της ανάπτυξης του προγράμματος και ιδιαίτερα κατά την κωδικοποίησή του και όχι μετά την ανάπτυξη. Ακόμη, τόσο η κωδικοποίηση και το προγραμματιστικό στυλ πρέπει να παρουσιάζουν συνέπεια και σωστή μεθοδολογία, διότι είναι βασικά συστατικά στοιχεία της τεκμηρίωσης. Είδαμε χαρακτηριστικό παράδειγμα κωδικοποίησης και προγραμματιστικού στυλ στην κωδικοποίηση ενός πρότυπου προγράμματος.

ΚΕΦΑΛΑΙΟ 19
ΑΞΙΟΛΟΓΗΣΗ, ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ, ΕΠΕΚΤΑΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Σκοπός κεφαλαίου:

Να γνωρίσουμε τη σημασία της αξιολόγησης ενός προγράμματος και την ανάγκη για επέκταση ενός προγράμματος.

Ειδικοί σκοποί:

- ❑ Να γνωρίσουμε τα χαρακτηριστικά της αξιολόγησης ενός προγράμματος.
- ❑ Να μπορούμε να αξιολογούμε ένα πρόγραμμα.
- ❑ Να μπορούμε να επεκτείνουμε ένα πρόγραμμα.

19.1. ΑΞΙΟΛΟΓΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Απαιτήσεις σε υλικό

Ένα πρόγραμμα πρέπει να εκμεταλλεύεται κατά τον καλύτερο τρόπο τους πόρους του συστήματος και ιδιαίτερα την κεντρική μονάδα επεξεργασίας και την κεντρική μνήμη, οι οποίες επιδρούν σημαντικά στην ταχύτητα εκτέλεσης του προγράμματος ιδιαίτερα σε περιβάλλον πολυπρογραμματισμού. Για το λόγο αυτό, είναι σημαντικό να επιλέγονται οι κατάλληλοι τύποι των μεταβλητών του προγράμματος, έτσι ώστε να μην γίνεται σπατάλη μνήμης και καθυστέρηση στην επεξεργασία. Για παράδειγμα, εάν μία μεταβλητή επιλεγεί ως **real** ενώ οι τιμές της είναι ακέραιες μεταξύ 1 και 20, θα υπάρχει καθυστέρηση στην επεξεργασία επειδή οι πράξεις με τους πραγματικούς αριθμούς είναι χρονοβόρες. Ακόμη, η παράσταση του ακέραιου αριθμού καταλαμβάνει λιγότερο χώρο από έναν πραγματικό. Επιπλέον, θα πρέπει να επιλέγονται οι κατάλληλοι αλγόριθμοι, εφόσον υπάρχον στην βιβλιογραφία για την ταχύτερη εκτέλεση, όπως οι αλγόριθμοι ταξινόμησης, αναζήτησης κλπ.

Παράγοντες που επηρεάζουν την αξιολόγηση ενός προγράμματος είναι:
- Απαιτήσεις σε υλικό,
- Φιλικότητα προς το χρήστη,
- Αξιοπιστία,
- Δόμηση προγράμματος,
- Παραμετρικότητα.

Φιλικότητα προς το χρήστη

Σημαντικό ρόλο στη χρήση ενός προγράμματος παίζει η φιλικότητα προς τον χρήστη ο οποίος είναι και ο τελικός αποδέκτης του προϊόντος αυτού. Το

πρόγραμμα πρέπει να επικοινωνεί διαλογικά με τον χρήστη. Οι οθόνες εισόδου να εμφανίζονται με τέτοιο τρόπο που να μην υπάρχει αμφισβήτηση για το τι και πού θα καταχωρίσει ο χρήστης. Η ύπαρξη άμεσης βοήθειας κατά τη διάρκεια της εκτέλεσης του προγράμματος σε συνδυασμό με το γραφικό η παραθυρικό περιβάλλον εργασίας βελτιώνουν την φιλικότητα του προγράμματος.

Αξιοπιστία

Η αξιοπιστία ενός προγράμματος διακρίνεται:

- Σε έλεγχο της ορθότητας των δεδομένων εισόδου πριν από την επεξεργασία. Για παράδειγμα, ο βαθμός ενός μαθητή σε ένα μάθημα πρέπει να βρίσκεται μέσα στο διάστημα [1, 20]. Κάθε άλλη τιμή δεν είναι αποδεκτή για επεξεργασία.
- Σε έλεγχο της ορθότητας των αποτελεσμάτων κάθε επεξεργασίας. Για παράδειγμα, αν υπολογίζω το μέσο όρο των βαθμών ενός μαθητή σε όλα τα μαθήματα, αυτός δεν μπορεί να βρίσκεται εκτός κλίμακας βαθμολογίας. Εάν δεχθούμε ότι έχει γίνει έλεγχος των δεδομένων εισόδου, τότε το σφάλμα θα οφείλεται:
  - σε λάθος του αλγόριθμου ή σε λάθος κωδικοποίησης.
  - σε λάθος επιλογή των τύπων των μεταβλητών. Για παράδειγμα αν **x=5** και **y=25000** και **z:=x*y** τότε το γινόμενο των δύο μεταβλητών θα είναι ο αριθμός **-6072** όπως φαίνεται αν εκτελεστεί το παρακάτω πρόγραμμα.

```
program test;
uses wincrt;
var
   x,y,z:integer;
begin
   x:5;
   y:=25000;
   z:=x*y;
   writeln(z);
end.
```

Το πρόβλημα αυτό οφείλεται στο ότι το αποτέλεσμα είναι εκτός των ορίων που επιτρέπει ο ακέραιος τύπος της Pascal.

Ο έλεγχος της αξιοπιστίας γίνεται κατά την διάρκεια της ανάπτυξης ενός προγράμματος με δεδομένα ελέγχου.

Δόμηση προγράμματος

Όταν η ανάπτυξη ενός προγράμματος ακολουθεί τις αρχές του δομημένου προγραμματισμού (ιεραρχική δομή, τμηματικότητα και χρήση των βασικών αλγοριθμικών δομών) διευκολύνεται ο έλεγχος και στη συνέχεια η συντήρηση και η πιθανή επέκτασή του.

Παραμετρικότητα

Ένα πρόγραμμα χαρακτηρίζεται ως παραμετρικό, όταν δίνεται η δυνατότητα στον χρήστη να αλλάξει τον τρόπο εκτέλεσης του προγράμματος αλλάζοντας κάποιες παραμέτρους όπως για παράδειγμα σε ένα πρόγραμμα μισθοδοσίας, τους συντελεστές υπολογισμού του φόρου.

19.2. ΕΠΕΚΤΑΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Μετά την ανάπτυξη ενός προγράμματος και κατά τη διάρκεια του κύκλου ζωής που διανύει είναι πιθανόν το πρόγραμμα αυτό να επεκταθεί. Η επέκταση του προγράμματος οφείλεται σε νέες απαιτήσεις των χρηστών που εμφανίζονται στο κύκλο ζωής του, ή σε αλλαγή των πόρων των υπολογιστικών συστημάτων σε Υλικό και Λογισμικό.

Η επέκταση λόγω αλλαγής απαιτήσεων των χρηστών, αναφέρεται σε προσθήκες νέων διαδικασιών ή συναρτήσεων στον αλγόριθμο, οι οποίες σχετίζονται με το πρόβλημα και πρέπει να αντιμετωπιστούν από το πρόγραμμα. Για παράδειγμα, στο πρόγραμμα επιλογής των υποψηφίων για ΑΕΙ και ΤΕΙ τα αποτελέσματα περιέχουν τη σχολή επιτυχίας, τη βαθμολογία και τις προτιμήσεις του υποψηφίου. Αν για στατιστικούς λόγους χρειαστούν κάποιες επεξεργασίες που δεν προβλέπονται στη διαδικασία εξαγωγής αποτελεσμάτων, τότε θα χρειαστεί μία επέκταση του προγράμματος με τα αντίστοιχα υποπρογράμματα για τη δημιουργία των στατιστικών αυτών. Ένα τέτοιο υποπρόγραμμα θα μπορούσε να είναι η σχέση προτίμησης με τη σχολή επιτυχίας.

Πολλές φορές οι νέες εκδόσεις των γλωσσών προγραμματισμού παρέχουν πολλές δυνατότητες για την ανάπτυξη των προγραμμάτων, κάτι που θα ήταν αδύνατο ή ασύμφορο να υλοποιηθεί με προηγούμενες εκδόσεις. Χαρακτηριστικό παράδειγμα είναι οι δυνατότητες γραφικών, δημιουργίας οθονών, αντικειμενοστραφούς προγραμματισμού κλπ. Για παράδειγμα η γλώσσα Pascal στην πορεία ανάπτυξής της πέρασε από πολλά στάδια όπου, το καθένα συμπλήρωνε το προηγούμενο με νέες προγραμματιστικές δυνατότητες με αποτέλεσμα η γλώσσα αυτή να διαθέτει όλες αυτές τις ευκολίες που αναφέρθηκαν παραπάνω.

Άλλες φορές, λόγω μείωσης του κόστους του υλικού ή της εμφάνισης νέου τύπου υλικού, τα προγράμματα επεκτείνονται για να μπορούν να χειριστούν το νέο υλικό. Για παράδειγμα στο πρόγραμμα καταχώρισης της βαθμολογίας ενός τμήματος στο σχολείο, αντί να γίνεται με πληκτρολόγηση, μπορεί να γίνει εισαγωγή των βαθμών μέσω σαρωτή (scanner). Το πρόγραμμα χρειάζεται επέκταση για τη χρήση του σαρωτή, όπως επίσης και για την αναγνώριση των χαρακτήρων.

Ανακεφαλαίωση

Γνωρίσαμε μερικά από βασικά στοιχεία τα οποία είναι απαραίτητα για την αξιολόγηση ενός προγράμματος και πρέπει να λαμβάνονται υπόψη κατά τη σχεδίαση και την ανάπτυξή του. Αντιληφθήκαμε την αναγκαιότητα για επέκταση ενός προγράμματος και τη συμβολή της καλής τεκμηρίωσης στην επέκταση του προγράμματος.