Οι Δυσκολίες των Αρχάριων Προγραμματιστών

Βασίλειος Εφόπουλος[1], Γεώργιος Ευαγγελίδης[1]
Βασίλειος Λαγδιλέλης[2], Αλέξανδρος Κλεφτοδήμος[3]
[1] Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας
[2] Τμήμα Εκπαιδευτικής & Κοινωνικής Πολιτικής, Πανεπιστήμιο Μακεδονίας
[3] 1ο ΤΕΕ Άργους Ορεστικού
efop@uom.gr, gevan@uom.gr, dagdil@uom.gr, alexkleft@sch.gr

ΠΕΡΙΛΗΨΗ
Την τελευταία εικοσαετία έχουν γίνει, σε διεθνές επίπεδο, αρκετές έρευνες σχετικές με τη διδασκαλία και την εκμάθηση του προγραμματισμού των ηλεκτρονικών υπολογιστών. Έχει παρατηρηθεί ότι οι αρχάριοι προγραμματιστές συναντούν αρκετές δυσκολίες στην κατανόηση και εφαρμογή βασικών αρχών και κανόνων του προγραμματισμού. Στη παρούσα εργασία γίνεται μια προσπάθεια επισκόπησης των ερευνών που σχετίζονται με τις δυσκολίες που αντιμετωπίζουν οι αρχάριοι προγραμματιστές στα πρώτα τους βήματα, τα λάθη που συχνά κάνουν και τα προβλήματα κατανόησης που συναντούν στις βασικές έννοιες και δομές μιας γλώσσας προγραμματισμού.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ*: Διδασκαλία Προγραμματισμού, Αρχάριοι, Δομές Δεδομένων*

ΕΙΣΑΓΩΓΗ
Τα τελευταία χρόνια, είναι γενικώς αποδεκτό ότι η διδασκαλία και η εκμάθηση του προγραμματισμού χαρακτηρίζονται κατά κανόνα από ορισμένες "δυσκολίες", οι οποίες εκδηλώνονται κυρίως κατά την κατασκευή ενός αλγορίθμου ή ενός προγράμματος. Ορισμένες δυσκολίες εκδηλώνονται και σε άλλες περιπτώσεις: όταν, για παράδειγμα, ο προγραμματιστής επιχειρεί να αιτιολογήσει ή να προβλέψει τη συμπεριφορά ενός αλγορίθμου ή επιχειρεί να τον διορθώσει. Ορισμένες δυσκολίες μοιάζουν να είναι, κατά κάποιο τρόπο, εγγενείς στον ίδιο τον προγραμματισμό, με την έννοια ότι συναντώνται κατά τρόπο συστηματικό, και είναι σχεδόν ανεξάρτητες από τη μέθοδο διδασκαλίας του αντίστοιχου αντικειμένου. Επίσης, πολλές απ' αυτές παρουσιάζουν εξαιρετική ανθεκτικότητα στο χρόνο και συναντώνται είτε σε μαθητές δημοτικού είτε σε σπουδαστές και φοιτητές (Pea 1986). Έτσι, οι έννοιες ή οι μέθοδοι αυτές αποτελούν πηγή δυσκολιών για μακρό χρονικό διάστημα κι επιπλέον εμφανίζονται κατά τρόπο σχεδόν ανεξάρτητο από το κοινό (αν και βέβαια η υπέρβαση τους δεν γίνεται πάντοτε με την ίδια ταχύτητα). Ένα μέρος από σχετικά θέματα προγραμματισμού (όπως για παράδειγμα η χρήση των βρόχων ή των αναδρομικών διαδικασιών από μη πεπειραμένους προγραμματιστές) έχει διερευνηθεί εκτεταμένα σε διεθνές επίπεδο. Σήμερα γνωρίζουμε αρκετά καλά τα προβλήματα που αντιμετωπίζουν οι αρχάριοι

Πρακτικά Εργασιών 3ου Πανελλήνιου Συνεδρίου «Διδακτική της Πληροφορικής»
Α. Τζιμογιάννης (επιμ.)
Πανεπιστήμιο Πελοποννήσου
Κόρινθος, 7-9 Οκτωβρίου 2005

προγραμματιστές στους τομείς αυτούς - και σε μερικές περιπτώσεις γνωρίζουμε διδακτικές μεθόδους για την υπέρβαση τους. Υπάρχουν πολλές θεωρίες που προσπαθούν να εξηγήσουν τι είναι αυτό που καθιστά την εκμάθηση του προγραμματισμού τόσο δύσκολη (Brusilovsky 1997, Du Boulay 1989, Brooks 1977). Θέματα που έχουν ερευνηθεί κυρίως αφορούν:

- προβλήματα της διδασκαλίας και κατανόησης των μεταβλητών,
- προβλήματα της διδασκαλίας και κατανόησης των επαναληπτικών δομών (βρόχων),
- προβλήματα της διδασκαλίας και κατανόησης των εντολών επιλογής σε διαδικαστικές γλώσσες προγραμματισμού (λ.χ. Pascal, σε αντιδιαστολή με γλώσσες συναρτησιακού προγραμματισμού όπως η LISP ή λογικού προγραμματισμού όπως η Prolog κλπ),
- προβλήματα της διδασκαλίας της αναδρομικότητας κυρίως σε γλώσσες στις οποίες η αναδρομικότητα είναι κυρίαρχη (όπως η LOGO),
- ιδιαίτερα θέματα όπως η μεταφορά γνώσεων από ένα προγραμματιστικό περιβάλλον σε άλλο ή η εισαγωγή στο λογικό και τον παράλληλο προγραμματισμό.

Στις επόμενες ενότητες αναλύονται τα προβλήματα διδασκαλίας και κατανόησης των μεταβλητών, των επαναληπτικών δομών, των δομών επιλογής και της αναδρομής.

Η ΔΥΣΚΟΛΙΑ ΕΚΜΑΘΗΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Ο Du Boulay (1989) περιγράφει πέντε βασικές περιοχές όπου επικεντρώνεται η δυσκολία της εκμάθησης του προγραμματισμού.

Η πρώτη περιοχή (Orientation) καλείται «Προσανατολισμός: Τι είναι ο προγραμματισμός και σε τι μας είναι χρήσιμος». Οι μαθητές αρκετές φορές συναντούν δυσκολίες με τις διάφορες μορφές προγραμματισμού. Σήμερα πολύ περισσότερο από ποτέ άλλοτε, ο προγραμματισμός καλύπτει μια ευρεία περιοχή των εφαρμογών λογισμικού και δεν συναντάται αποκλειστικά στα περιβάλλοντα των κλασσικών γλωσσών. Πολλά πακέτα λογισμικού ενσωματώνουν δυνατότητες προγραμματισμού (συνήθως μέσω μακροεντολών), όπως λογιστικά φύλλα, επεξεργαστές κειμένου, προγράμματα δημιουργίας γραφικών κ.ά. Αυτό το γεγονός έχει καταστήσει πλέον θολά τα όρια μεταξύ του χρήστη εφαρμογών και του προγραμματιστή.

Η δεύτερη περιοχή (Notional Machine) καλείται «Νοητή μηχανή – Πώς λειτουργεί ο υπολογιστής». Ο μαθητής καλείται να «ελέγξει» μια εικονική μηχανή. Τι μορφή παίρνει η εικονική μηχανή και ποιο είδος εντολών αναμένεται να καταλαβαίνει; Πώς γίνεται η επικοινωνία, η έκδοση και η ανάγνωση των εντολών; Τα διανοητικά μοντέλα που κατασκευάζουν οι μαθητές είναι κρίσιμα για την κατανόηση κάθε νέας έννοιας στην οποία εισάγονται. Επιπλέον, η κατοχή ενός «φτωχού» διανοητικού μοντέλου μπορεί να οδηγήσει τους μαθητές να αναπτύξουν φτωχές στρατηγικές εκμάθησης με αποτέλεσμα την απουσία κινήτρου, την έλλειψη ενδιαφέροντος, την αποθάρρυνση και απογοήτευση (Kessler 1986).

Η τρίτη περιοχή δυσκολίας (Notation), που περιγράφει ο Du Boulay (1989), αναφέρεται στα προβλήματα που προκύπτουν από την ίδια τη γλώσσα προγραμματισμού, συμπεριλαμβανομένων των συντακτικών και σημασιολογικών

κανόνων. Αυτά περιλαμβάνουν σχεδιασμούς που οδηγούν εύκολα σε λάθη ή περιλαμβάνουν δυσκολονόητες διαδικασίες.

Η τέταρτη περιοχή δυσκολίας (Structures) περιγράφει ένα απαραίτητο στοιχείο της μετάβασης από τον αρχάριο στον έμπειρο προγραμματιστή: την εκμάθηση και πραγματική κατάκτηση των δομών, με τέτοιο τρόπο, ώστε μελλοντικά να μπορούν να ανακληθούν εύκολα και να ενσωματωθούν σε μια λύση ενός προβλήματος. Παραδείγματα τέτοιων δομών είναι ο υπολογισμός ενός αθροίσματος με χρήση βρόχου, ένας αλγόριθμος αναζήτησης, ένας αλγόριθμος ταξινόμησης, κώδικας για την αντιμετάθεση τιμών κ.α. Οι έμπειροι προγραμματιστές έχουν στο μυαλό τους έτοιμη τη λύση προς χρήση ανά πάσα στιγμή για τέτοιες δομές, αφού έχουν αντιμετωπίσει ανάλογες καταστάσεις στο παρελθόν. Οι αρχάριοι, λόγω απειρίας, στερούνται τέτοιας δυνατότητας με αποτέλεσμα να βρίσκει εμπόδια η προσπάθειά τους να λύσουν ανάλογα προβλήματα.

Η πέμπτη περιοχή δυσκολίας (Pragmatics) παραμελείται συχνά στα μαθήματα προγραμματισμού, παρά την προφανώς ιδιαίτερη σημασία της. Έχει να κάνει με τις βοηθητικές δεξιότητες που είναι απαραίτητες για τον προγραμματισμό και αφορούν την ικανότητα προσαρμογής και ελέγχου ενός περιβάλλοντος στον υπολογιστή που θα χρησιμοποιηθεί για τη συγγραφή κώδικα, τη μεταγλώττιση και αποσφαλμάτωση των λαθών ενός προγράμματος. Οι μαθητές ορισμένες φορές έχουν δυσκολία να εγκλιματισθούν στο περιβάλλον ανάπτυξης προγραμμάτων και να μάθουν να χρησιμοποιούν τα διαθέσιμα εργαλεία, πριν ακόμα αρχίσουν να εξετάζουν την ίδια τη γλώσσα προγραμματισμού.

ΤΑ ΣΥΧΝΑ ΛΑΘΗ ΤΩΝ ΜΑΘΗΤΩΝ

Ο Du Boulay (1989) θεωρεί ότι τα λάθη των μαθητών μπορούν να ταξινομηθούν στις ακόλουθες τρεις κατηγορίες:

1. Κακή εφαρμογή της αναλογίας (Misapplication of analogy): η μεταβλητή παρομοιάζεται ένα κουτί αποθήκευσης, οπότε αρκετοί μαθητές θεωρούν ότι θα μπορούσε να αποθηκεύσει περισσότερες από μία τιμές.
2. Υπεργενίκευση (Overgeneralization): ό,τι λειτουργεί σωστά για έναν τύπο αντικειμένου θα πρέπει να λειτουργεί σωστά και για κάποιον άλλο. Λ.χ. η χρήση του χαρακτήρα “;” για το διαχωρισμό ορισμάτων σε ένα βρόχο FOR μπορεί να οδηγήσει τους μαθητές να χρησιμοποιήσουν τον ίδιο χαρακτήρα (“;”) ως διαχωριστικό των παραμέτρων στην κλήση μιας συνάρτησης.
3. Λάθος χειρισμός της πολυπλοκότητας και της αλληλεπίδρασης: κακή διαχείριση και τοποθέτηση σε λάθος σημείο των υποτμημάτων ενός μεγάλου προγράμματος.

Ο Du Boulay (1989) παρατηρεί επίσης ότι οι μαθητές συχνά θεωρούν ότι ο υπολογιστής (ή το πρόγραμμα) θα επιστρέψει ως αποτέλεσμα αυτό που οι ίδιοι πιστεύουν (και θέλουν) και όχι αυτό που προκύπτει από την εκτέλεση των εντολών του προγράμματος που έχουν γράψει. Ακόμη θεωρεί ότι η χρησιμοποίηση αγγλικών λέξεων σε μια γλώσσα προγραμματισμού μπορεί να παραπλανήσει τους μαθητές κάνοντας τους

να σκεφτούν ότι ο υπολογιστής διαθέτει (όπως ο άνθρωπος) την ικανότητα να συμπεραίνει τι εννοεί κάποιος με τα λεγόμενα του.

Ο Putnam (1986) μελέτησε τους μαθητές γυμνασίου που μαθαίνουν να προγραμματίζουν σε γλώσσα BASIC. Διαπίστωσε ότι όταν η προγενέστερη γνώση μεταφέρεται λανθασμένα στον προγραμματισμό υπολογιστών, τότε ενδεχομένως μπορεί να προκαλέσει αρκετά λάθη. Οι σπουδαστές τείνουν να καλύψουν την ελλιπή γνώση τους στη BASIC, υποθέτοντας ότι η μηχανή έχει ικανότητες φυσικής γλώσσας, εξάγοντας (προφανώς λανθασμένα) συμπεράσματα για τη χρήση των λέξεων κλειδιών της γλώσσας σύμφωνα με την αγγλική τους σημασία.

Οι Bruckman & Edwards (1999) θεωρούν ότι τα λάθη που σχετίζονται με τη φυσική γλώσσα είναι πιθανό να εμφανιστούν ανεξάρτητα από το πόσο κοντά βρίσκεται η γλώσσα προγραμματισμού με τη φυσική γλώσσα. Οι παραπάνω ερευνητές διαπίστωσαν ότι τα λάθη που σχετίζονται με τη φυσική γλώσσα, όταν χρησιμοποιείται γλώσσα προγραμματισμού που βρίσκεται πολύ κοντά στη φυσική γλώσσα, αποτελούν ένα μικρό ποσοστό των συνολικών λαθών (10,6%) και από αυτά το 61,8% διορθώνεται από τους μαθητές χωρίς εξωτερική βοήθεια. Καταλήγουν στο συμπέρασμα ότι τα λάθη που σχετίζονται με τη φυσική γλώσσα δεν αποτελούν ικανοποιητικό λόγο για να αποφεύγεται η χρήση φυσικής γλώσσας στη σχεδίαση γλωσσών προγραμματισμού.

Ο Du Boulay (1989) επίσης σημειώνει ότι οι μαθητές αρκετές φορές αγνοούν ή είναι απληροφόρητοι για την κατάλληλη αντιμετώπιση ενός λάθους με αποτέλεσμα να καταφεύγουν σε λάθος ενέργειες, ακόμα και σε υπερπροσπάθεια για ένα ασήμαντο συντακτικό λάθος. Για παράδειγμα μπορεί να καταφύγουν σε διαγραφή όλου του προγράμματος ή ακόμα και σε επανεκκίνηση του υπολογιστή.

Ο Perkins (1989) μελέτησε τις στρατηγικές των μαθητών που μαθαίνουν να προγραμματίζουν. Όπως και ο Du Boulay, διαπίστωσε ότι οι αρνητικές εμπειρίες αναγκάζουν μερικούς μαθητές να σταματήσουν την προσπάθεια τους. Αυτοί οι μαθητές (Stoppers) εγκαταλείπουν την προσπάθεια τους μόλις βρουν ένα πρόβλημα. Διαπιστώθηκε ότι αυτή η τάση θα μπορούσε να αντιμετωπιστεί αν δοθεί στους μαθητές μια μικρή θετική εμπειρία για να τους ενθαρρύνει. Το άμεσο συμπέρασμα είναι ότι η πρώτη επαφή και εμπειρία στον προγραμματισμό, στο στάδιο όπου οι μαθητές καταστρώνουν την τεχνική προγραμματισμού τους, είναι ιδιαίτερα σημαντική. Η άλλη κλάση των μαθητών (Movers) ακολουθεί διαφορετικές στρατηγικές προκειμένου να συντάξει ένα σωστό πρόγραμμα, αν και όχι πάντα με επιτυχία. Οι μαθητές αυτοί πολλές φορές αλλάζουν τυχαία τον κώδικα ενός προγράμματος (προφανώς πειραματίζονται) χωρίς να κατανοούν που ακριβώς είναι το πρόβλημα.

ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑΝΟΗΣΗΣ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ

Η ανάθεσης τιμής σε μεταβλητή παρουσιάζει δυσκολίες και είναι γενικά αποδεκτό ότι αποτελεί ένα από τα προβλήματα που συναντούν οι μαθητές στη διδασκαλία του προγραμματισμού. Πολλοί ερευνητές έχουν ασχοληθεί με την έννοια της μεταβλητής και το ρόλο της στην αλγοριθμική και τον προγραμματισμό, όπως οι Samurcay (1985), Bayman (1983) και Rogalski (1989).

Η δυσκολία που παρουσιάζει η έννοια της μεταβλητής πηγάζει ορισμένες φορές από την επιλογή του συμβόλου ανάθεσης τιμής. Ως αποτέλεσμα, εκφράσεις της μορφής “X=X+1” προκαλούν σύγχυση και σύγκρουση μεταξύ Προγραμματισμού και Μαθηματικών. Η συγκεκριμένη έκφραση δεν έχει νόημα στα Μαθηματικά, αφού δεν είναι δυνατό ένας αριθμός Χ να είναι ίσος με τον εαυτό του συν τη μονάδα, ενώ στον Προγραμματισμό παριστάνει μια εκχώρηση. Ορισμένοι ερευνητές διατυπώνουν επιπλέον την άποψη ότι στις εκφράσεις του τύπου “X=Y+1”, υπάρχει μια σύγχυση σχετικά με το αν το σύμβολο "=" είναι το σύμβολο ισότητας ή το σύμβολο της εκχώρησης τιμής. Αρκετές γλώσσες προγραμματισμού κάνουν χρήση του “=” ως συμβόλου εκχώρησης τιμής, προφανώς για λόγους απλότητας, γεγονός που δημιουργεί σύγχυση με το αλγεβρικό σύμβολο της ισότητας. Στην Pascal για να αποφεύγονται τέτοια προβλήματα χρησιμοποιείται το σύμβολο “:=” για την εκχώρηση τιμής.

Ο Du Boulay (1989) θεωρεί ότι οι παρερμηνείες στην κατανόηση των μεταβλητών βασίζονται στα παραδείγματα (παρομοιώσεις - αναλογίες) που χρησιμοποιούν οι εκπαιδευτικοί στην τάξη (λ.χ. η παρομοίωση της μεταβλητής με ένα κουτί ή ένα συρτάρι με μια ετικέτα, μπορεί να παρερμηνευτεί από τους μαθητές και να θεωρήσουν ότι η μεταβλητή μπορεί να έχει περισσότερες από μία τιμές). Έτσι ορισμένοι μαθητές θεωρούν ότι η μεταβλητή «θυμάται» κάθε τιμή που της δίνεται, εκτός αν σβηστούν τα περιεχόμενα της μνήμης. Δεν κατανοούν ότι η νέα τιμή καταχωρείται στη θέση της παλιάς η οποία και χάνεται.

Άλλη περίπτωση, που αναφέρει ο Du Boulay, είναι η παρομοίωση της μεταβλητής με πλάκα (slate) όπου πάνω μπορούν να γραφούν τιμές. Και αυτή η αναλογία μπορεί να παρερμηνευτεί από μαθητές, που δεν αντιλαμβάνονται ότι η υπάρχουσα τιμή επικαλύπτεται από τη νέα. Θεωρούν τη μεταβλητή σαν μια λίστα που περιέχει όλες τις τιμές που έχουν εκχωρηθεί στη μεταβλητή, οι οποίες και μπορούν να ανακτηθούν.

Μια συνηθισμένη παρερμηνεία της εντολής ανάθεσης τιμής A = B είναι η ακόλουθη: Το Β συνδέεται με το Α και έτσι όποιες αλλαγές πρόκειται να γίνουν στο Α θα επηρεάσουν άμεσα και το Β. Παρατηρείται επίσης ότι ένα συχνό λάθος είναι η μη απόδοση αρχικής τιμής (λ.χ. 0 ή άλλη τιμή) σε μετρητές βρόχων. Στην περίπτωση αυτή οι μαθητές μπερδεύονται με την αναλογία μεταβλητής – κουτιού, όπου το κουτί είναι άδειο έως ότου τοποθετήσουμε κάτι μέσα του, οπότε ανάλογα θεωρούν ως 0 και το περιεχόμενο της μεταβλητής.

Οι Bayman & ayer (1983) καταλήγουν στο συμπέρασμα ότι δύο είδη εκχώρησης τιμών σε μεταβλητές μπορεί να οδηγήσουν σε παρερμηνείες. Η πρώτη περίπτωση είναι η αρχικοποίηση (X=0) και η άλλη είναι η εξίσωση (X=Y+1). Στις περιπτώσεις αυτές οι μαθητές συχνά θεωρούν ότι ο υπολογιστής έχει καταγράψει κάπου την πληροφορία ή την έχει εκτυπώσει στην οθόνη ενώ αντίθετα η πληροφορία έχει αποθηκευθεί σε συγκεκριμένη θέση στη μνήμη. Πολλοί μαθητές θεωρούν ότι αποθηκεύεται η εξίσωση και όχι η τιμή. Οι συγγραφείς καταλήγουν στο συμπέρασμα ότι "*οι αρχάριοι προγραμματιστές χρειάζονται ειδική εκπαίδευση στα θέματα που αφορούν θέσεις μνήμης και κάτω από ποιες συνθήκες οι τιμές που αποθηκεύονται στις θέσεις αυτές μπορούν να αντικατασταθούν*".

Η Samurcay (1985) καταλήγει σε τέσσερις τρόπους ανάθεσης τιμών σε μεταβλητές:

- Ανάθεση σταθερής τιμής (constant value, A=3).
- Ανάθεση τιμής που προκύπτει από υπολογισμό (calculated value, A=2*B+1).
- Αντιγραφή (duplication, A=B).
- Συσσώρευση (accumulation, A=A+1).

Το μαθηματικό υπόβαθρο ενός μαθητή για τις έννοιες της μεταβλητής και της ισότητας τον βοηθά στους πρώτους τρεις πρώτους τρόπους ανάθεσης τιμών. Δεν συμβαίνει κάτι τέτοιο όμως στην περίπτωση της συσσώρευσης όπου η έννοια της μεταβλητής προκαλεί σύγχυση και απαιτεί διαφορετική αντιμετώπιση. Στην περίπτωση αυτή πρέπει να γίνει σαφής διάκριση μεταξύ αριστερού και δεξιού τμήματος της εντολής ανάθεσης, και να κατανοήσουν οι μαθητές ότι το αριστερό τμήμα σχετίζεται με τη θέση μνήμης ενώ το δεξί τμήμα με την τιμή που θα πάρει η μεταβλητή.

Η Samurcay επίσης κατατάσσει τις μεταβλητές σε δύο κατηγορίες - εσωτερικές και εξωτερικές - και περιγράφει τη χρήση των τεσσάρων τρόπων ανάθεσης σε κάθε κατηγορία μεταβλητών. Εξωτερικές είναι οι μεταβλητές που αποτελούν είσοδο ή έξοδο (αποτέλεσμα) σε ένα πρόγραμμα. Αυτές είναι υπό τον έλεγχο του χρήστη, όταν εκτελεί το πρόγραμμα. Εσωτερικές είναι οι μεταβλητές που είναι αναγκαίες μόνο στη (προγραμματιστική) λύση ενός προβλήματος και είναι υπό τον έλεγχο του προγραμματιστή. Η Samurcay πιστεύει ότι οι αρχάριοι δυσκολεύονται περισσότερο με τον χειρισμό των εσωτερικών μεταβλητών. Αυτό συμβαίνει γιατί, στην περίπτωση αυτή, απαιτείται μια διαρκής αναπαράσταση της εσωτερικής λειτουργίας του υπολογιστή. Προκειμένου να ερευνηθεί αυτή η δυσκολία, μελετήθηκε η χρήση της μεταβλητής κατά την επίλυση προγραμμάτων με βρόχους, που αναλύεται στην επόμενη παράγραφο.

ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑΝΟΗΣΗΣ ΤΩΝ ΕΠΑΝΑΛΗΠΤΙΚΩΝ ΔΟΜΩΝ

Οι επαναληπτικές δομές (βρόχοι) αποτελούν δομικό στοιχείο σχεδόν κάθε προγράμματος, πέραν εκείνων των σύντομων εκπαιδευτικών προγραμμάτων που χρησιμοποιούνται στη διαδικασία εκμάθησης μια γλώσσας. Αποτέλεσαν δε, ιδιαίτερα τα τελευταία 25 χρόνια, ένα από τα πλέον διερευνημένα θέματα από πολλούς ερευνητές κι ερευνητικές ομάδες,όπως οι Soloway (1983), Rogalski (1986), Du Boulay (1989), Hoc (1989), Δαγδιλέλης (1996) κ.α.

Οι ερευνητές συμφωνούν στο συμπέρασμα ότι οι επαναληπτικές δομές αποτελούν μια περιοχή όπου παρατηρούνται δυσκολίες στους αρχάριους προγραμματιστές. Οι δυσκολίες αυτές μπορεί να οφείλονται είτε σε αδυναμία γενίκευσης (Hoc 1989) είτε σε αδυναμία ανάπτυξης ενός κατάλληλου νοητικού μοντέλου της δομής του βρόχου (Kessler 1989). Ο Hoc (1989) παρατηρεί ότι υπάρχει τάση στους νέους προγραμματιστές να γράφουν ξανά το ίδιο κομμάτι κώδικα αντί να χρησιμοποιήσουν βρόχο. Ο Du Boulay (1989) θεωρεί ότι οι βρόχοι δημιουργούν στους αρχαρίους αρκετά προβλήματα. Αρκετοί δεν κατανοούν πως γίνεται η αλλαγή τιμής στον μετρητή ενός βρόχου FOR, μια και δεν αποτυπώνεται η ενέργεια αυτή στον κώδικα, αλλά γίνεται εξ υποθέσεως. Οι Rogalski και Samurcay (1990) καταγράφουν τις ενέργειες που

λαμβάνουν χώρα στη δημιουργία ενός βρόχου. Τρεις τύποι διεργασιών μεταβλητών εμπεριέχονται σε ένα βρόχο:

- Αρχικοποίηση (initialization), όπου γίνεται η απόδοση των αρχικών τιμών στις μεταβλητές του βρόχου.
- Ενημέρωση (updating), όπου γίνεται η (απαραίτητη) αναπροσαρμογή των τιμών των μεταβλητών.
- Έλεγχος (test), όπου καθορίζεται η συνθήκη τερματισμού του βρόχου.

Οι αρχάριοι συνήθως δεν μπορούν να καθορίσουν το τμήμα ενημέρωσης και συχνά γράφουν τον κώδικα ενός βρόχου παραλείποντάς το. Επίσης δυσκολεύονται να «συνθέσουν» τη συνθήκη εξόδου. Οι παραπάνω ερευνήτριες κατέληξαν στο συμπέρασμα ότι η αρχικοποίηση των μεταβλητών που περιλαμβάνονται στο σώμα ενός βρόχου είναι μια σύνθετη νοητική διαδικασία και παρουσιάζει περισσότερες δυσκολίες από τον έλεγχο και την ενημέρωση.

Οι επαναληπτικές δομές μπορούν να ταξινομηθούν με διάφορες προσεγγίσεις. Ανάλογα με τη θέση που γίνεται ο έλεγχος της συνθήκης, η έξοδος από το βρόχο διακρίνεται (Pane & Myers 1996) σε:

- Top-exit, όπου ο έλεγχος της συνθήκης γίνεται στην αρχή, πριν την εκτέλεση εντολών του σώματος του βρόχου, λ.χ. ο βρόχος while στην Pascal.
- Bottom-exit, όπου ο έλεγχος της συνθήκης γίνεται στο τέλος, μετά την εκτέλεση του σώματος του βρόχου, λ.χ. ο βρόχος repeat στην Pascal.
- Middle-exit, όπου ο έλεγχος της συνθήκης γίνεται εντός του σώματος του βρόχου.
- Daemon-exit, όπου η έξοδος μπορεί να γίνει οπουδήποτε, αν ο έλεγχος της συνθήκης αποτύχει.

Ο Wu (1991) επισημαίνει ότι οι αρχάριοι στη φάση της περιγραφής, ακολουθούν τη στρατηγική bottom-exit όταν πιστεύουν ότι είναι ευκολότερη, αλλά μετά γυρίζουν σε στρατηγική middle-exit για όλες τις άλλες περιπτώσεις. Η έρευνα των Rogalski και Samurcay (1990) ενισχύει την παραπάνω θέση, αφού βρίσκει ότι η στρατηγική top-exit είναι πιο δύσκολη από την bottom-exit. Και αυτό γιατί οι νέοι προγραμματιστές δυσκολεύονται να αναπαραστήσουν και να εκφράσουν μια συνθήκη τερματισμού για ένα τμήμα κώδικα (το σώμα του βρόχου) με το οποίο δεν έχουν ακόμα καταπιαστεί. Η έρευνα του Soloway (1983) καταλήγει στο συμπέρασμα ότι οι αρχάριοι γράφουν πιο σωστά το τμήμα των βρόχων ενός προγράμματος αν χρησιμοποιήσουν μια δομή που επιτρέπει την έξοδο από το μέσο του βρόχου (middle-exit) και όχι από την αρχή ή το τέλος. Προτείνουν τη δομή loop...leave...again για την καλύτερη κατανόηση των βρόχων. Αντίθετα ο Ledgard (1975) υποστηρίζει ότι η έξοδος από το βρόχο πρέπει να γίνεται είτε στην αρχή είτε στο τέλος του βρόχου. Παρόλο που αυτό μπορεί να είναι στην αρχή πιο δύσκολο, το αποτέλεσμα τελικά είναι η καλύτερη δόμηση και διαχείριση του προγράμματος. Η έξοδος από το μέσο ενός βρόχου, ενώ φαινομενικά είναι σωστή, οδηγεί σε λάθος προγραμματιστική λογική και δυσχεραίνει την αναγνωσιμότητα (readability) του προγράμματος. Στο ίδιο συμπέρασμα καταλήγει και ο Sheppard (1979) σε έρευνα που έγινε με δείγμα ομάδα προγραμματιστών. Συγκρίνει δύο τμήματα κώδικα που περιέχουν βρόχο, που το πρώτο χαρακτηρίζεται «απόλυτα δομημένο» και δεν

επιτρέπει την έξοδο από το μέσο του βρόχου ενώ το δεύτερο αντίθετα είναι «φυσικά δομημένο» και επιτρέπει την έξοδο από το μέσο. Συμπεραίνει ότι δεν παρατηρήθηκε αξιοσημείωτη διαφορά μεταξύ των δύο τμημάτων στην απόδοση των προγραμματιστών στη συγγραφή κώδικα. Επίσης δεν παρατηρεί διαφορά και σε θέματα τροποποίησης και αποσφαλμάτωσης κώδικα. Αρκετοί μαθητές θεωρούν ότι ένας βρόχος WHILE τερματίζεται από τη στιγμή που η συνθήκη εξόδου παύει να ισχύει (daemon-exit), ανεξάρτητα από το σε ποια εντολή του βρόχου βρίσκεται εκείνη τη στιγμή η εκτέλεση του προγράμματος, ενώ κανονικά θα πρέπει να περιμένουν έως ότου η συνθήκη ελεγχθεί στην αρχή του βρόχου (top-exit) (Sleeman 1988).

Τα πιο συχνά λάθη και παρανοήσεις των μαθητών που σχετίζονται με βρόχους, σύμφωνα με τον Sleeman (1988) είναι:

- Θεωρούν ότι μια εντολή που βρίσκεται αμέσως μετά το τέλος του βρόχου, συμπεριλαμβάνεται σε αυτόν.
- Η τελευταία εντολή ενός βρόχου εκτελείται πολλές φορές, ενώ όλες οι άλλες εντολές εκτελούνται μία φορά.
- Δίνουν χαρακτηριστικά βρόχου στο τμήμα των εντολών που εμπεριέχεται στο begin-end.
- Πιστεύουν ότι μια μεταβλητή κρατά περισσότερες από μία τιμές και έτσι χειρίζονται μια εντολή επιλογής (conditional) σαν βρόχο.
- Πιστεύουν ότι η μεταβλητή που χρησιμοποιείται ως μετρητής στο βρόχο FOR ή δεν έχει τιμή μέσα στο βρόχο, ή ότι είναι σωστό να αλλάζει η τιμή της μέσα στο βρόχο.

Συμπερασματικά, οι έρευνες φαίνεται να δείχνουν ότι, αντίθετα από τους έμμεσους ή άμεσους ισχυρισμούς της σχολής του δομημένου προγραμματισμού, όλες οι επαναληπτικές δομές δεν είναι εξίσου περίπλοκες για τον προγραμματιστή.

ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑΝΟΗΣΗΣ ΤΩΝ ΔΟΜΩΝ ΕΠΙΛΟΓΗΣ

Οι λανθασμένες αντιλήψεις των αρχάριων προγραμματιστών έχουν συχνά την αιτία τους στην καθημερινή ζωή. Η λύση ενός προβλήματος προγραμματισμού συνήθως προκύπτει από τη μεταφορά της επίλυσης αποτυπωμένης με χρήση φυσικής γλώσσας (Δαγδιλέλης 1996), που δε σημαίνει όμως ότι με τον τρόπο αυτό οδηγούμαστε στο σωστό αποτέλεσμα. Γενικότερα οι αρχάριοι προγραμματιστές συναντούν τις ακόλουθες δυσκολίες στην κατανόηση των δομών επιλογής:

- Εμφωλευμένες δομές επιλογής: Εδώ ο βαθμός δυσκολίας αυξάνεται ανάλογα με το βάθος εμφώλευσης (Rogalski 1990).
- Σύνθετες εκφράσεις – συνθήκες: Οι πολύπλοκες εκφράσεις που χρησιμοποιούν Boolean συναρτήσεις ή συνδυασμό προτάσεων με τους λογικούς τελεστές AND, OR και NOT, δυσχεραίνουν σαφώς την κατανόηση της λειτουργίας μιας εντολής ελέγχου (Δαγδιλέλης 1996).
- Εντοπισμός των ορίων εμβέλειας (begin .. end): Η απουσία εντολών που καθορίζουν τα όρια εμβέλειας των τμημάτων μετά τα THEN και ELSE κάνει πιο δύσκολη την αναγνώριση τους από τους μαθητές (Sime 1977).

Επίσης έχει παρατηρηθεί ότι:

- Οι μαθητές προτιμούν να χρησιμοποιούν εντολές “GOTO” ή “JUMP” στις δομές ελέγχου. Ο τρόπος αυτός φαίνεται να είναι πιο εύκολος σε σχέση με τις δομές υψηλού επιπέδου (εμφωλευμένα IF..THEN..ELSE), αλλά παρουσιάζει σημαντικές δυσκολίες διαχείρισης του ελέγχου ροής του προγράμματος (Rogalski 1990).
- Οι μαθητές που έχουν καλό μαθηματικό υπόβαθρο μαθαίνουν τις νέες δομές ελέγχου γρηγορότερα (Δαγδιλέλης 1996).

Ο δομημένος προγραμματισμός βασίζεται κατά ένα μέρος σε εμφωλευμένες εντολές. Κλασική περίπτωση αποτελούν τα εμφωλευμένα if..then..else. Για πολλούς ερευνητές οι εμφωλευμένες εντολές ελέγχου προκαλούν σύγχυση. Αντίθετα άλλοι ερευνητές θεωρούν ότι, αντίθετα με την διαδεδομένη αντίληψη, η παράθεση ελέγχων είναι σε ελάχιστο βαθμό και σε ορισμένες μόνο περιπτώσεις πιο κατανοητή από τους εμφωλευμένους ελέγχους. Θεωρούν ότι οι εμφωλευμένες εντολές ελέγχου δεν πρέπει να καταργηθούν. Αυτό όμως που φαίνεται ουσιαστικό είναι να υπάρχει μια ορατή αντιστοιχία ανάμεσα στη δομή του κώδικα και τη "φυσική" θέση των εντολών μέσα στο πρόγραμμα. Έχει προταθεί ακόμη κι η χρήση ειδικών συμβόλων που χρησιμοποιούνται από πολλούς συγγραφείς για αναπαράσταση αλγοριθμικών γλωσσών.

Σύμφωνα με τους Putnam (1986) και Sleeman (1988), τα πιο συχνά λάθη και παρανοήσεις των μαθητών που σχετίζονται με τις δομές επιλογής είναι:

- Στην περίπτωση που η δομή επιλογής δεν έχει τμήμα ELSE, αναμένουν τη διακοπή εκτέλεσης του προγράμματος και την εμφάνιση μηνύματος λάθους αν η συνθήκη της εντολής IF είναι ψευδής (false).
- Στην περίπτωση που η δομή επιλογής έχει και τμήμα ELSE, αναμένουν την εκτέλεση τόσο του τμήματος THEN όσο και του ELSE.
- Αναμένουν την εκτέλεση του τμήματος THEN ανεξάρτητα από το αν η συνθήκη είναι αληθής ή όχι.
- Στην περίπτωση που η δομή επιλογής δεν έχει τμήμα ELSE, διαχειρίζονται την αμέσως επόμενη εντολή (που δεν ανήκει στην IF..THEN) όπως η ELSE, πιστεύουν δηλαδή ότι η εντολή αυτή εκτελείται μόνο όταν η συνθήκη είναι ψευδής.

ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑΝΟΗΣΗΣ ΤΗΣ ΑΝΑΔΡΟΜΗΣ

Η έννοια της αναδρομής από μόνη της δεν είναι απλή στην αναπαράσταση της. Ακόμα περισσότερο δύσκολη είναι η διδασκαλία και η εκμάθηση της. Οι μαθητές δυσκολεύονται να κατανοήσουν τη ροή εκτέλεσης ενός προγράμματος με αναδρομή. Σύμφωνα με τους Pane & Myers (1996), οι αρχάριοι συχνά χρησιμοποιούν ένα επαναληπτικό μοντέλο για να αναπαραστήσουν την αναδρομή που είναι συμβατό με την αναδρομή ουράς (tail recursion) αλλά αποτυγχάνει σε πιο γενικές περιπτώσεις. Γι’ αυτό το λόγο η Rogalski (1990) συστήνει τη διδασκαλία της αναδρομής πριν από τις επαναληπτικές δομές. Σε αναλυτική έρευνα πάνω στα μοντέλα που χρησιμοποιούν οι αρχάριοι για την αναδρομή (Kahney 1989), βρέθηκε ότι ενώ ένας μεγάλος αριθμός από αρχάριους (>50%) φαίνεται να υιοθετούν ένα επαναληπτικό μοντέλο, στην πραγματικότητα οι περισσότεροι δεν κατέχουν κανένα μοντέλο. Ο Kessler (1989) ανέλυσε τη διαδικασία μεταβίβασης της διδασκαλίας ανάμεσα στις επαναληπτικές δομές

και στην αναδρομή. Ανακάλυψε θετικά στοιχεία όταν γίνεται μεταβίβαση από τις επαναληπτικές δομές στην αναδρομή αλλά όχι και αντίστροφα. Κατέληξε να προτείνει τη διδασκαλία των επαναληπτικών δομών πριν από την αναδρομή.

ΑΛΛΑ ΠΡΟΒΛΗΜΑΤΑ

Οι μαθητές δεν είναι εξοικειωμένοι με την έννοια του πίνακα. Συγχέουν πολλές φορές τις γραμμές και τα κελιά, τους δείκτες και τις τιμές. Δυσκολία παρατηρείται και σε γλώσσες προγραμματισμού στις οποίες η αρίθμηση των γραμμών αρχίζει από τη θέση 0, κάτι που αντιτίθεται στο μαθηματικό υπόβαθρο των μαθητών, όπου η δομή του πίνακα αρχίζει από τη θέση 1.

Η κατανόηση των διαδικασιών εισόδου και εξόδου δεδομένων δεν είναι εύκολη σε αρκετές γλώσσες προγραμματισμού. Περισσότερα προβλήματα δημιουργούνται στις περιπτώσεις που η ανάθεση τιμής δεν είναι προφανής. Έτσι αρκετοί μαθητές δεν αντιλαμβάνονται τη λειτουργία εντολών όπως READ/INPUT, δηλαδή τη διακοπή της εκτέλεσης του προγράμματος έως ότου ο χρήστης πληκτρολογήσει μια τιμή. Σε τέτοιες περιπτώσεις, δηλαδή όταν τα δεδομένα εισάγονται από το πληκτρολόγιο, ο αρχάριος χρήστης ενδέχεται να ερμηνεύει την "ακινησία" της οθόνης σαν ένα είδος βλάβης ή ως ένα πρόβλημα ή δυσκολεύεται να εξηγήσει την κίνηση της πληροφορίας (εισαγωγή από πληκτρολόγιο και αποθήκευση της σε μια θέση μνήμης). Εξάλλου, εντολές όπως η INPUT A ή READ A, πολλές φορές ερμηνεύονται ως αποθήκευση στη μνήμη του ίδιου του συμβόλου A (Bayman 1983). Στα σύγχρονα περιβάλλοντα ωστόσο, η είσοδος των δεδομένων επισημαίνεται με τη χρήση ειδικών παραθύρων και έτσι η υπέρβαση της σχετικής δυσκολίας πραγματοποιείται σε σύντομο χρονικό διάστημα.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα αποτελέσματα των ερευνών που καταγράφουν τις δυσκολίες των αρχαρίων όταν μαθαίνουν προγραμματισμό αποτελούν ένα ιδιαίτερα χρήσιμο εργαλείο για τους εκπαιδευτικούς της πληροφορικής στη διαδικασία σχεδίασης του διδακτικού τους υλικού. Όπως αναφέρει και ο Spohrer (1986), ο συντονισμός και η ολοκληρωμένη προσέγγιση για την επίλυση προβλημάτων είναι η βάση και για τη συγγραφή προγραμμάτων. Δεν είναι κάτι που μπορεί να γίνει εύκολα αλλά είναι πολύ σημαντικό.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Bayman P. & Mayer R. E. (1983), A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements, *Communications of the ACM*, 26(9), 677-679

Brooks R. (1977), Towards a theory of the cognitive processes in computer programming, *International Journal of Man-Machine Studies*, 9, 737-751

Bruckman A. & Edwards E. (1999), Should we leverage natural-language knowledge? An analysis of user errors in a Natural-Language-Style Programming Language, *Computer Human Interaction 1999 (CHI'99)*, Pittsburgh, USA, May

Brusilovsky P., Calabrese E., Hvorecky E., Kouchnirenko A. & Miller P. (1997), Mini-languages: A way to learn programming principles, *Education and Information Technologies*, 2(1), 65-83

Du Boulay B. (1989), Some difficulties of learning to program, in E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*, 283-299, Hillsdale, NJ: Lawrence Erlbaum Associates

Hoc J.-M. (1989), Do we really have conditional statements in our brains?, in E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*, 179-90, Hillsdale, NJ: Lawrence Erlbaum Associates

Kahney H. (1989), What do novice programmers know about recursion, ". in E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*, 209-228, Hillsdale, NJ: Lawrence Erlbaum Associates

Kessler C. M. & J. R. Anderson (1986), A model of novice debugging in LISP, in E. Soloway & S. Iyengar (Eds.), *Empirical Studies of Programmers*, 198-212, Washington, DC, Ablex Publishing Corporation

Ledgard H. F. & Marcotty M. (1975), A genealogy of control structures, *Communications of the ACM,* 18(11), 629-638

Pane J. & Myers B. (1996), *Usability issues in the design of novice programming systems*, School of Computer Science Technical Report, Carnegie Mellon University, CMU-CS-96-132, Pittsburgh

Pea R. (1986), Language-independent conceptual 'bugs' in novice programming, *Journal of Educational Computing Research*, 2(1), 25-36

Perkins D. N., Hancock C., Hobbs R., Martin F. & Simmons R. (1989), Conditions of learning in novice programmers, in E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*, Hillsdale, NJ: Lawrence Erlbaum Associates

Putnam R. T., Sleeman D., Baxter J. A. & Kuspa L. K. (1986), A summary of misconceptions of high school basic programmers, *Journal of Educational Computing Research*, 2(4)

Rogalski J. & Samurcay R., (1990), Acquisition of programming knowledge and skills, in J.-M. Hoc, T. R. G. Green, R. Samurcay & D. Gilmore (Eds.), *Psychology of Programming*, 157-174, London: Academic Press

Samurcay R. (1985), The concept of variable in programming: Its meaning and use in problem-solving by novice programmers, *Education Studies in Mathematics*, 16(2), 143-161

Schneiderman B. (1980), *Software psychology: Human factors in computer and information systems*, Winthrop Publishers Inc., Cambridge

Sheppard. S. B., Curtis. B., Milliman, P. & Love. T. (1979), Modern coding practices and programmer performance, *Computer*, Dec., 41-49

Sime M. E., Green T. R. G. & Guest D. J. (1977), Scope marking in computer conditionals: A psychological evaluation, *International Journal of Man-Machine Studies*, 9, 107-118

Sleeman D., Putnam R. T., Baxter J. & Kuspa L. (1988), An introductory Pascal class: A case study of students' errors, in R. E. Mayer (Ed.), *Teaching and Learning Computer Programming: Multiple Research Perspectives*, 237-257, Hillsdale, NJ: Lawrence Erlbaum Asociates

Spohrer J. & Soloway E. (1986), Novice mistakes: are the folk wisdoms correct?, *Communications of the ACM*, 29(7), 624-632

Wu Q. & Anderson J. R. (1991), Strategy selection and change in Pascal programming, in J. Koenemann-Belliveau, T. G. Moher & S. P. Robertson (Eds.), *Empirical Studies of Programming: Fourth Workshop*, 227-238, New Brunswick, NJ: Ablex Publishing Corporation

Δαγδιλέλης Β., (1996), *Διδακτική της Πληροφορικής. Η διδασκαλία του προγραμματισμού: αντιλήψεις των σπουδαστών για την κατασκευή κι επικύρωση προγραμμάτων και διδακτικές καταστάσεις για τη διαμόρφωσή τους*, Διδακτορική διατριβή, Τμήμα Εφ. Πληροφορικής, Πανεπιστήμιο Μακεδονίας