ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΩΝ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΣΥΜΒΟΛΟΣΕΙΡΩΝ ΜΕ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PASCAL, C++ ΚΑΙ JAVA

ΑΙΓΙΝΗΤΟΥ–ΜΠΙΜΑ ΜΑΡΙΑ

ΑΛΕΞΑΚΗ ΥΠΑΤΙΑ

ΓΟΥΡΓΟΥΡΙΝΗ ΕΛΕΝΗ

ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗΤΗΣ

ΜΠΑΚΑΛΗΣ ΑΡΙΣΤΕΙΔΗΣ

ΠΑΤΡΑ 2009

Περίληψη

Θέμα αυτής της πτυχιακής εργασία είναι η μελέτη των αλγορίθμων εύρεσης συμβολοσειράς (αλφαριθμητικό, string) μέσα σε κείμενο. Αναλύεται περισσότερο ο σκοπός αυτός στο κεφάλαιο 1, ενώ στο κεφάλαιο 2 αναφερόμαστε στις γλώσσες προγραμματισμού που έχουν αναπτυχθεί και στις κατηγοριοποιήσεις τους. Στο κεφάλαιο 3 αναφερόμαστε στις συμβολοσειρές, στον τρόπο που αναπαρίστανται εσωτερικά στον υπολογιστή και σε αλγορίθμους εύρεσης συμβολοσειράς μέσα σε κείμενο. Στα κεφάλαια 4, 5 και 6 μελετάμε τα χαρακτηριστικά των γλωσσών προγραμματισμού Pascal, Java και C++ και τον τρόπο με τον οποίο η κάθε μία χειρίζεται τις συμβολοσειρές. Τέλος, αναπτύσσουμε τους αλγόριθμους Brute Force, KMP, Not So Naive και Quick Search σε κάθε μία από αυτές τις γλώσσες και αναλύουμε, παράλληλα με τη λειτουργία του καθενός, την ιδιαίτερη υλοποίηση που απαιτεί η κάθε γλώσσα γι' αυτούς.

Πίνακας Περιεχομένων

1. ΕΙΣΑΓΩΓΗ

Στον αιώνα που διανύουμε, κανείς δε μπορεί να φανταστεί τη δημιουργία ενός εργασιακού αλλά και προσωπικού χώρου, χωρίς τη παρουσία αυτού του «μαγικού κουτιού» που ονομάζεται υπολογιστής. Ο ηλεκτρονικός υπολογιστής αποτελεί αναπόσπαστο κομμάτι της καθημερινής μας ζωής. Αποτελεί τη πηγή αποθήκευσης και επεξεργασίας απλών και σημαντικών δεδομένων. Η επιστήμη των υπολογιστών έχει καταφέρει να φέρει με το πέρασμα των χρόνων ραγδαίες εξελίξεις στον τομέα της, έχοντας σαν απώτερο στόχο τη βελτίωση των συστημάτων παροχής πληροφοριών με όσο το δυνατόν ταχύτερη και ακριβέστερη εξυπηρέτηση και υποστήριξη. Στην καθημερινή μας ζωή η αντιμετώπιση των προβλημάτων γίνεται με μια κλασική τροπή. Αρχικά εντοπίζεται το πρόβλημα, ακολουθεί η ανάλυση του και τέλος παρουσιάζονται οι πιθανοί τρόποι αντιμετώπισης του. Έτσι συμβαίνει και εδώ. Αναζητούνται τα δεδομένα, καταχωρούνται, επεξεργάζονται και έπειτα παρουσιάζονται τα αποτελέσματα που ζητούνται.

Ένας υπολογιστής αποτελείται από τα μέρη που απαρτίζουν την εξωτερική του όψη, το Υλικό (Hardware), και από το σύνολο των προγραμμάτων που αποφασίζουν για τη συμπεριφορά του, το λογισμικό (Software). Η κεντρική μονάδα επεξεργασίας (ΚΜΕ) αποτελεί τον κεντρικό συντονιστή μαζί με τις μονάδες εισόδου-εξόδου (πληκτρολόγιο, οθόνη, εκτυπωτής κ. α.) που συνδέονται γύρω από αυτήν. Η επικοινωνία της με το εξωτερικό περιβάλλον επιτυγχάνεται με τα κυκλώματα επικοινωνίας (Interfaces), με τα οποία συνδέονται οι μονάδες εισόδου-εξόδου. Η ΚΜΕ αποτελείται από την αριθμητική και τη λογική μονάδα - όπου γίνονται οι αριθμητικές και οι λογικές πράξεις – και από την μονάδα ελέγχου, που αποφασίζει για την εκτέλεση των εντολών. Οι εντολές εκτελούνται με βάση κάποιο πρόγραμμα που είναι αποθηκευμένο στη μνήμη του Η/Υ. Εκεί κρατούνται όλες οι πληροφορίες που απαιτούνται για την εκτέλεση ενός προγράμματος.

Η κύρια μνήμη (main memory) του υπολογιστή είναι βασικά δύο ειδών:

- ROM (Read Only Memory): Η μνήμη αυτή είναι μικρή σε χωρητικότητα αλλά το περιεχόμενο της διατηρείται ανέπαφο ακόμα και όταν χάνεται η επαφή με το ηλεκτρικό ρεύμα.
- RAM (Random Access Memory): Σε αυτή τη μνήμη αποθηκεύονται προγράμματα και δεδομένα. Κάθε θέση της έχει μια διεύθυνση που είναι μοναδική και μέσω αυτής υπάρχει μόνο πρόσβαση. Το περιεχόμενο της χάνεται με τη διακοπή του ηλεκτρικού ρεύματος.

Επίσης, καθίσταται αναγκαία η ύπαρξη της βοηθητικής –περιφερειακής μνήμης, καθώς η ιδιότητα τη κεντρικής μνήμης να χάνει το περιεχόμενο της με την διακοπή του ηλεκτρικού ρεύματος καθιστούν αναγκαία την ύπαρξη μαγνητικής μνήμης (δισκέτες, σκληροί δίσκοι, μαγνητικές ταινίες, οπτικοί δίσκοι), όπου μπορούν να αποθηκεύονται προγράμματα και δεδομένα ασφαλή για μελλοντική χρήση.

2. Γλώσσες Προγραμματισμού

«Πρόγραμμα» ορίζεται να είναι ένα σύνολο από οδηγίες και εντολές σε μορφή κατανοητή από τον υπολογιστή. Αποτελούν το λειτουργικό σύστημα και ανήκουν στο Software του Η/Υ. Τα περισσότερα είναι αποθηκευμένα σε κάποιο μαγνητικό ή οπτικό μέσο –συνήθως σε δίσκο- και φορτώνονται στον υπολογιστή μόλις αυτός τεθεί σε λειτουργία. Τα προγράμματα που χρησιμοποιούνται για την επίλυση κάποιων προβλημάτων λέγονται προγράμματα εφαρμογής.

«Γλώσσα προγραμματισμού» είναι ένα αυστηρά καθορισμένο σύνολο κανόνων και εντολών που αποτελεί τη βάση για την ανάπτυξη των προγραμμάτων. Ο προγραμματισμός απαιτεί κατανόηση της δομής του υπολογιστή, των προγραμμάτων για τον υπολογιστή και των γλωσσών προγραμματισμού στις οποίες γράφονται τα προγράμματα. Μία γλώσσα προγραμματισμού προσδιορίζεται από το αλφάβητο, το λεξιλόγιο, τη γραμματική και τη σημασιολογία της. Αλφάβητο καλείται το σύνολο των στοιχείων που χρησιμοποιείται από τη γλώσσα προγραμματισμού. (χαρακτήρες που επιτρέπονται στις εντολές του προγράμματος)

Το λεξιλόγιο αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου (λέξεις που χρησιμοποιούνται για την έκφραση των εντολών).

Η Γραμματική αποτελείται από το τυπικό και το συντακτικό.

Τυπικό είναι το σύνολο των κανόνων που ορίζει το αν μία λέξη είναι αποδεκτή.

Συντακτικό είναι το σύνολο των κανόνων που καθορίζει τη νομιμότητα της διάταξης και της σύνδεσης των λέξεων μιας γλώσσας.

Σημασιολογία είναι το σύνολο των κανόνων που καθορίζουν το νόημα των λέξεων.

Η διαδικασία προγραμματισμού των Η/Υ περιλαμβάνει τα εξής στάδια:

- Ανάλυση: Κατανόηση του δεδομένου προβλήματος και κατάστρωση της διαδικασίας επίλυσής του.
- Σχεδιασμός του αλγορίθμου επίλυσης: Ο σχεδιασμός μπορεί να εκφραστεί με φυσική γλώσσα, με βήματα, διαγράμματα ροής, ψευδοκώδικα, κλπ.
- Συγγραφή του προγράμματος: Χρήση ειδικών προγραμμάτων (editors), που παρέχουν λειτουργίες αντιγραφής, μεταφοράς, αναζήτησης κειμένου, αποθήκευσης ή εκτύπωσης του προγράμματος. Το αποτέλεσμα είναι ο πηγαίος κώδικας (source code) του προγράμματος, το οποίο δεν είναι άμεσα κατανοητό από τον Η/Υ.
- Μεταγλώττιση (compilation) του πηγαίου κώδικα σε γλώσσα μηχανής. Αν ο πηγαίος κώδικας δεν είναι συντακτικά σωστός, ο μεταγλωττιστής (compiler) εμφανίζει μηνύματα λάθους. Αν είναι συντακτικά σωστός παράγει τον εκτελέσιμο κώδικα.
- Διασύνδεση μονάδων προγράμματος από διαφορετικά αρχεία.
- Εκτέλεση του προγράμματος.
- Έλεγχος του προγράμματος με διάφορα δεδομένα, ώστε να εξακριβωθεί η ορθή ή λανθασμένη λειτουργία του και διόρθωση των λογικών λαθών του προγράμματος, δηλαδή λαθών που οδηγούν σε μη αναμενόμενα αποτελέσματα με τη βοήθεια ειδικών προγραμμάτων που λέγονται εκσφαλματωτές (debuggers).

Οι γλώσσες προγραμματισμού χωρίζονται σε :

- Γλώσσες μηχανής

Αρχικά δημιουργήθηκαν οι γλώσσες μηχανής (machine language) και αργότερα οι γλώσσες assembly. Η assembly διευκολύνει την εργασία του προγραμματιστή, αφού είναι μια συμβολική παράσταση της γλώσσας μηχανής. Κάθε επεξεργαστής έχει τη δική του γλώσσα μηχανής, με αποτέλεσμα κάθε Η/Υ με διαφορετικό επεξεργαστή να έχει διαφορετική γλώσσα μηχανής.

Για να μπορέσει ο υπολογιστής να αρχίσει μια διαδικασία έπρεπε να του δοθούν οι κατάλληλες οδηγίες. Οδηγίες υπό την μορφή 0 και 1, σύμβολα κατανοητά για τον ηλεκτρονικό υπολογιστή αλλά μη κατανοητά για τον άνθρωπο.

- Γλώσσες χαμηλού επιπέδου (Συμβολικές)

Οι γλώσσες χαμηλού επιπέδου είναι δύσκολα κατανοητές. Μέσω ενός ειδικού προγράμματος, του assembler (συμβολομεταφραστή), οι εντολές μεταφράζονται από τον ηλεκτρονικό υπολογιστή σε μια ακολουθία δυαδικών ψηφίων 0 και 1 και έπειτα μπορούν να εκτελεστούν.

- Γλώσσες υψηλού επιπέδου.

Οι γλώσσες υψηλού επιπέδου (high level languages)-στις οποίες ανήκουν οι C++, η Java και η Pascal- είναι όσες χρησιμοποιούν όρους ίδιους με τις πράξεις που εκτελούν. Είναι εύκολα κατανοητές από τον άνθρωπο αφού οι λέξεις που χρησιμοποιούν είναι στα Αγγλικά και είναι κατανοητές από όλους. Τα προγράμματα που γράφονται σε γλώσσες υψηλού επιπέδου είναι ανεξάρτητα από τον Η/Υ και μπορούν να εκτελεστούν αυτούσια σε κάθε μηχάνημα. Οι παραπάνω γλώσσες χρησιμοποιούν εντολές που θυμίζουν την αγγλική γλώσσα και μετατρέπονται σε εντολές γλώσσας μηχανής όταν εκτελούνται στους υπολογιστές. Τα προγράμματα που γράφουμε είναι απλά αρχεία κειμένου (text files) που περιέχουν εντολές προγραμματισμού και μπορούμε να τα δημιουργήσουμε με έναν επεξεργαστή κειμένου (text editor). Το πρόγραμμα που γράφουμε αποκαλείται πηγαίο πρόγραμμα (source program) και είναι κατανοητό από εμάς αλλά όχι και από τον υπολογιστή, ο οποίος αναγνωρίζει το ίδιο πρόγραμμα που είναι μεταφρασμένο σε γλώσσα μηχανής, μεταγλωττισμένο ή αντικείμενο μηχανής (object program). Ένα πρόγραμμα σε γλώσσα υψηλού επιπέδου για να εκτελεστεί, μεταφράζεται σε γλώσσα μηχανής με τη βοήθεια ενός ειδικού προγράμματος που λέγεται «Μεταφραστής». Υπάρχουν δύο τύποι προγραμμάτων-μεταφραστών: Compiler (μεταγλωττιστής) και Interpreter (ερμηνευτής).

Υπάρχουν πολλές γλώσσες προγραμματισμού, κάθε μια από αυτές συγκεντρώνει πλεονεκτήματα και μειονεκτήματα. Παρακάτω διατυπώνονται τα βασικότερα πλεονεκτήματα.

- Να είναι εύκολη, κατανοητή και γρήγορη.
- Να μπορεί να χρησιμοποιηθεί σε όλες τις εφαρμογές που υπάρχουν-ακόμα και σε αυτές που μπορούν να προκύψουν.
- Να μπορεί να τρέχει σε όλους τους Η/Υ.
- Καμία γλώσσα δε μπορεί να είναι κατάλληλη για όλες τις εφαρμογές. Για αυτό και υπάρχουν αρκετές γλώσσες προγραμματισμού.
- Άλλες είναι κατάλληλες για εμπορικές εφαρμογές, άλλες για επιστημονικές και άλλες για πιο ειδικευμένες εφαρμογές, όπως για τη μουσική κτλ.

3. Συμβολοσειρές

Ορισμός συμβολοσειράς : Με τον όρο συμβολοσειρά μπορούμε να χαρακτηρίσουμε έναν μονοδιάστατο πίνακα του οποίου τα στοιχεία είναι χαρακτήρες (strings). Η ακολουθία των στοιχείων αυτών ονομάζετε συμβολοσειρά.

Για παράδειγμα η σειρά χαρακτήρων « ABIDE» αποτελεί μια συμβολοσειρά, αν ονομάζαμε την συμβολοσειρά αυτή S τότε S1 είναι ο χαρακτήρας A και S2 ο χαρακτήρας B. Γενικότερα μια συμβολοσειρά είναι μια ακολουθία χαρακτήρων τυπικά μεγάλου μήκους. Το σημαντικότερο όμως χαρακτηριστικό των συμβολοσειρών είναι ότι χρησιμοποιούνται σαν αντικείμενα των συστημάτων επεξεργασίας κειμένων δηλαδή με την βοήθεια και την δημιουργία συμβολοσειρών μπορούμε να πετύχουμε μια σειρά από επεξεργασίες σε κείμενα.

ΤΥΠΟΙ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

- ΔΥΑΔΙΚΕΣ ΣΥΜΒΟΛΟΣΕΙΡΕΣ:

  Ένας ιδιαίτερος τύπος συμβολοσειρών είναι η δυαδική συμβολοσειρά, η οποία είναι μια ακολουθία δυαδικών ψηφίων 0 και 1. Τέτοιου είδους συμβολοσειρές είναι ιδιαίτερα γνωστές γιατί χρησιμοποιούνται κατά κόρον σε διάφορα συστήματα όπως για παράδειγμα στην εμφάνιση και αποθήκευση γραφικών και εικόνων στις οθόνες και στη μνήμη. Ιδιαίτερο χαρακτηριστικό της δυαδικής συμβολοσειράς είναι ότι μπορεί να αποθηκεύσει σε διαδοχικά bits αντί σε ολόκληρες οχτάδες δηλαδή bytes που απαιτούνται για τις συμβολοσειρές χαρακτήρων, τέλος οι δυαδικές συμβολοσειρές χρησιμοποιούν μόνο δυο διαφορετικά σύμβολα.

- ΣΥΜΒΟΛΟΣΕΙΡΕΣ ΧΑΡΑΚΤΗΡΩΝ:

  Σε αντίθεση με τις δυαδικές συμβολοσειρές οι συμβολοσειρές χαρακτήρων δημιουργούνται από σύμβολα ενός μεγάλου αλφάβητου (γράμματα ελληνικά και αγγλικά, αριθμοί, ειδικοί χαρακτήρες). Εξαιτίας αυτού του γεγονότος το

μέγεθος του αλφάβητου επηρεάζει δραστικά την σχεδίαση των σχετικών αλγόριθμων, με αποτέλεσμα να έχουμε μια ποικιλία αλγόριθμων για κάθε διαφορετικό τύπο συμβολοσειράς.

ΑΛΦΑΡΙΘΜΗΤΙΚΟΙ ΠΙΝΑΚΕΣ

Όπως αναφέραμε και πιο πάνω οι συμβολοσειρές μπορούν να είναι στοιχεία ενός πίνακα, και όταν αυτό είναι εφικτό τότε μιλάμε για έναν αλφαριθμητικό πίνακα. Στις διάφορες υλοποιήσεις αλφαριθμητικών πινάκων και κατά συνέπεια στις διάφορες γλώσσες προγραμματισμού μπορούμε να διακρίνουμε τις εξής δυο περιπτώσεις αποθήκευσης στοιχείων σε ένα αλφαριθμητικό πίνακα Κάθε στοιχείο του πίνακα είναι συμβολοσειρά σταθερού μεγέθους. Στην περίπτωση αυτή κάθε στοιχείο του πίνακα έχει σταθερό μέγεθος και λειτουργεί το ίδιο όπως και στους αριθμητικούς πίνακες. Είναι προφανές ότι η αποθήκευση ενός μεγαλύτερου σε μήκος στοιχείου από το προκαθορισμένο μέγεθος, δεν είναι επιτρεπτή αλλά δεν ισχύει το ίδιο και για την αποθήκευση ενός μικρότερου σε μήκους στοιχείου, του οποίου η αποθήκευση μπορεί να είναι εφικτή αφήνοντας απλά κάποιες θέσεις κενές. Βέβαια σε αυτή την περίπτωση έχουμε κάποιο είδος σπάταλης μνήμης και θα μπορούσαμε να πούμε ότι είναι ένα βασικό μειονέκτημα στην υλοποίηση πινάκων συμβολοσειρών σταθερού μεγέθους.

Κάθε στοιχείο του πίνακα είναι συμβολοσειρά μεταβλητού μεγέθους :

Σε αυτή τη περίπτωση η αποθήκευση ενός στοιχείου γίνετε διαφορετικά και μπορεί να υπάρχει και η δυνατότητα ποικιλίας. Πιο συγκεκριμένα για την αποθήκευση ενός πίνακα διατίθεται μνήμη το πολύ 64Kbytes και κάθε συμβολοσειρά αποθηκεύετε σε κάποιες συγκεκριμένες θέσεις μνήμης. Επίσης κάθε στοιχείο του πίνακα έχει ένα σταθερό μέγεθος ίσο με 3. Πιο αναλυτικά το συγκεκριμένο σταθερό μέγεθος αποτελείτε από τρία bytes, στο πρώτο byte ορίζετε το μήκος της συμβολοσειράς, ενώ στα αλλά δυο αποθηκεύετε η διεύθυνση αρχής στην μνήμη της συμβολοσειράς.

Με τη μέθοδο συμβολοσειρών μεταβλητού μεγέθους μπορούμε να έχουμε πίνακες με στοιχεία συμβολοσειρών μεταβλητού μήκους. το μόνο τίμημα που συναντάμε είναι η δημιουργία ενός επιπλέον πίνακα, ο ποιος περιέχει τα μήκη και τις διευθύνσεις αρχής των συμβολοσειρών. τέλος απαιτείτε ένας ακόμα μηχανισμός χειρισμού της μνήμης και αυτό γιατί όταν εκχωρηθεί μία άλλη συμβολοσειρά σε ένα στοιχείο του πίνακα α και έχει διαφορετικά μέγεθος από αυτή που αντικαθιστά, τότε αυτό που γίνετε είναι να δημιουργούνται κενές θέσεις μνήμης για την καινούργια συμβολοσειρά ενώ οι θέσεις που καταλάμβανε η παλιά θεωρούνται άχρηστα δεδομένα, έτσι με την βοήθεια του συγκεκριμένου μηχανισμού έχουμε πρόνοια για την ανάκτηση των άχρηστων δεδομένων.

ΑΝΑΖΗΤΗΣΗ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

Ένα πολύ σημαντικό κεφαλαίο στις συμβολοσειρές είναι η αναζήτηση συμβολοσειρών. Με τον όρο αναζήτηση συμβολοσειρών εννοούμε το ταίριασμα μιας δεδομένης συμβολοσειράς με μιας δεύτερης την ονομάζουμε και πρότυπο. Πιο συγκεκριμένα η εύρεση μια συμβολοσειράς (πρότυπο) μέσα σε μια άλλη συμβολοσειρά (κείμενο) λέγεται ταύτιση πρότυπου η ταίριασμα. Είναι δηλαδή η λειτουργία κατά την οποία αναζητάμε μια λέξη μέσα σε ένα κείμενο. Ο πιο απλοϊκός τρόπος αναζήτησης ενός πρότυπου μέσα σε ένα κείμενο είναι ο διαδοχικός έλεγχος κάθε χαρακτήρα του κειμένου. Για παράδειγμα αν ο πρώτος χαρακτήρας του κειμένου ταιριάζει με αυτόν του πρότυπου τότε συνεχίζει στον επόμενο κτλ. Η μέθοδος αυτή είναι η πιο απλοϊκή γι΄ αυτό τον λόγο ονομάζετε και ωμή βία (BRUTE FORCE). Ο παρακάτω αλγόριθμος υλοποιεί την συγκεκριμένη μέθοδο αναζήτησης.

Ο ΑΛΓΟΡΙΣΘΜΟΣ BRUTE FORCE

```
Δεδομένα // a, N, p, M//
found <ψευδής :index? 0
 i←1: j←1
 αρχή _επανάληψης
        αν a(i)=p(i) τότε
          i←i+1:j←j+1
          αλλιώς
          i←i-j+2:j←1
       τέλος _αν
μεχρις_οτου j >M η ι >N
αν j>M τότε
     found ←αληθής :index ←i-m
τέλος _αν
αποτελέσματα //found,  index  //
τέλος brute force
```

ΣΥΜΒΟΛΙΣΜΟΙ:

όπου N: μέγεθος του πίνακα α (περιέχει χαρακτήρες βασικού κειμένου)

M: μέγεθος του πίνακα p (περιέχει χαρακτήρες προτύπου)

ΔΕΙΚΤΕΣ ΣΑΡΩΣΗΣ:

συμβολίζουμε με i το κείμενο με J το πρότυπο

Επεξήγηση αλγορίθμου:

Στο συγκεκριμένο αλγόριθμο ο μονοδιάστατος πίνακας a μεγέθους N περιέχει τους χαρακτήρες του βασικού κειμένου και αντίστοιχα ο πίνακας p μεγέθους M περιέχει χαρακτήρες του πρότυπου δηλαδή για παράδειγμα την λέξη που θέλουμε να βρούμε μέσα στον πίνακα. Οι δείκτες i και j χρησιμοποιούνται για την σάρωση του κειμένου και του πρότυπου αντίστοιχα. Στον συγκεκριμένο αλγόριθμο όταν οι πρώτοι χαρακτήρες του πρότυπου ταυτίζονται με κάποιους

αντίστοιχους του κειμένου, τότε οι δυο δείκτες αυξάνονται. Αν όμως κάποια στιγμή διαπιστωθεί μη ταύτιση, τότε ο δείκτης i παίρνει την επόμενη τιμή από αυτή που είχε ξεκινήσει και ο δείκτης j γίνεται 1. ο αλγόριθμος τερματίζει όταν ο δείκτης j καταφέρει να γίνει μεγαλύτερος από το M που σημαίνει ότι όλοι οι χαρακτήρες του πρότυπου κατάφεραν να ταυτιστούν.

Παράδειγμα:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Τ	Α		Δ	Ε	Δ	Ο	Μ	Ε	Ν	Α		Κ	Α	Ι		Ο	Ι		Δ	Ο	Μ	Ε	Σ		Τ	Ο	Υ	Σ

Υποθέτουμε ότι θέλουμε να εμφανίζετε η λέξη δομές, δηλαδή αναζητούμε μέσα σε ένα κείμενο την λέξη δομές επομένως το πρότυπο μας είναι ΔΟΜΕΣ. Ο δείκτης i αυξάνετε μέχρι την θέση 4 όπου διαπιστώνετε ταύτιση με τον πρώτο χαρακτήρα του πρότυπου, στην συνεχεία ο δείκτης j γίνετε 2 και ο i προχωράει στην θέση 5 όπου διαπιστώνετε μη ταύτιση εφόσον ο χαρακτήρας κειμένου δεν είναι ίδιος με του πρότυπου. Όποτε ο i παραμένει 5 και ο j ξανά μειώνεται και γίνεται 1. Στις θέσεις από 6 έως 9 υπάρχει ταύτιση επομένως στην θέση 9 που διαπιστώνετε και η τελευταία ταύτιση ο δείκτης i προχωράει και γίνετε 10 και ο j γίνετε 5. στην θέση 10 όμως δεν υπάρχει ταύτιση επομένως ο i γίνετε 5 (i ←j-i +2:j←1) και ο j ξαναγυρνάει στην τιμή 1.

ΑΛΓΟΡΙθΜΟΣ KMP

Ο αλγορίθμους KMP ο οποίος επινοήθηκε από τους Knuth, Pratt και Morris λειτουργεί με μια διαφορετικά από τον προηγούμενο αλγόριθμο που είδαμε. Πιο συγκεκριμένα στην αναζήτηση χαρακτήρων όταν διαπιστωθεί μια μη ταύτιση ο δείκτης i δεν οπισθοχωρεί στον προηγούμενο χαρακτήρα του κειμένου, αντίθετος ξεκινάει από τον τρέχοντα. Για παράδειγμα στην προηγούμενη συμβολοσειρά είχαμε ταύτιση στις θέσεις απόν 6 μέχρι 9 αλλά στην δέκατη θέση είχε διαπιστωθεί μη ταύτιση με το πρότυπο κειμένου, με τον συγκεκριμένο αλγόριθμο ο δείκτης i δεν χρειάζεται να οπισθοχωρήσει στην έβδομη θέση ξανά γιατί έτσι και αλλιώς αποκλείετε να υπάρχει ταύτιση κειμένου και πρότυπου από την θέση 7 μέχρι και την 10. Ουσιαστικά αυτό που κάνει ο αλγόριθμος KMP είναι ότι μετά από μια μη ταύτιση δεν αλλάζει το δείκτη i αλλά μόνο τον j, βέβαια για την συγκεκριμένη λειτουργία χρειάζεται μια προ επεξεργασία κειμένου προκείμενου να σχηματιστεί ένα διάνυσμα αλμάτων next το οποίο θα μας παρέχει και την αύξηση του δείκτη j. Πιο συγκεκριμένα δημιουργείτε ένας πίνακας next και βασίζεται στην σύγκριση του πρότυπου κειμένου με τον εαυτό του.

Αλγόριθμος KMP_SEARCH

Δεδομένα // a, N, p, M//

!δημιουργία πίνακα next

i←1: j←0:next (1) ←

αρχη_επαναληψης

Αν j = 0 η p(j) τότε

i←i+1:j←j+1

αν p(i)=/ p(j) τότε

next (i) ←j

αλλιώς

```
     next (i) ←next(j)
τέλος _αν
    αλλιώς
j←next (j)
τέλος αν
μέχρις οτου i>m
!αναζήτηση
found ←ψευδης :index ←0
i←1:j←1
αρχη_επαναληψης
   αν j=0 η a(i)=p(i) τότε
i←i+1:j←j+1
αλλιώς
    j←next(j)
τελος_αν
μεχρις_οτου j>M η i> N
Αν j>M τότε
       Found < αληθής :index ←i-m
Τέλος αν
Αποτελέσματα // found index//
Τέλος KMP_SEARCH
```

Ο συγκεκριμένος αλγόριθμος δεν είναι πολύ καλύτερος από τον αλγόριθμο brute_force στις συνήθεις περιπτώσεις αναζήτησης κειμένου. Ωστόσο υπερτερεί με ένα βασικό πλεονέκτημα, ο δείκτης i δεν οπισθοχωρεί ποτέ. Έτσι όταν εφαρμόζετε σε μεγάλα αρχεία κειμένου αποθηκευμένα στον δίσκο, δεν απαιτεί επιπλέον προσπελάσεις δίσκου

ΣΥΝΑΡΤΗΣΗ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ

Μια ακόμα τεχνική για την αναζήτηση συμβολοσειρών και η οποία βασίζετε στην μέθοδο κατακερματισμού είναι η συνάρτηση κατακερματισμού. Με αυτή την μέθοδο αρχικά υπολογίζετε μια τιμή για το πρότυπο κειμένου που ψάχνουμε και διαφορετικές τιμές για κάθε πεδίο Μ χαρακτήρων του κανονικού μας κειμένου. Με αυτόν τον τρόπο κάθε τιμή συγκρίνετε με την προηγούμενη και η τιμή πεδίου-κειμένου Μ που θα είναι ίσο με την τιμή πρότυπου θα μας δείχνει μια πιθανή ταύτιση.

Πιο αναλυτικά για παράδειγμα αν είχαμε το κείμενο:

Τ	Α		Δ	Ε	Δ	Ο	Μ	Ε	Ν	Α		Κ	Α	Ι		Ο	Ι		Δ	Ο	Μ	Ε	Σ		Τ	Ο	Υ	Σ

Και ψάχναμε την λέξη δομές, η συνάρτηση κατακερματισμού θα υπολόγιζε μια τιμή για την λέξη πρότυπο την οποία θα ονομάσουμε hp και θα χώριζε το κείμενο σε διαδοχικούς βρόχους για τους οποίους θα υπολόγιζε επίσης διαφορετικές τιμές για τον καθένα, αυτές τις τιμές θα τις ονομάσουμε hk. Επομένως όταν ξεκινήσει η διαδικασία αναζήτησης ελέγχονται οι τιμές hp και hk ως προς την ισότητα για κάθε βρόχο ξεχωριστά. Αν οι τιμές ανάμεσα στο πρώτο πεδίο κειμένου και στην τιμή του πρότυπου δεν είναι ίσες τότε υπολογίζετε με νέα τιμή για το κείμενο η οποία εξαρτάται από τους Μ χαρακτήρες του κειμένου από το δεύτερο μέχρι και Μ+1. Βασικό πλεονέκτημα είναι ότι όταν δεν βρεθεί ισότητα-ταύτιση τιμών ο υπολογισμός δεν χρίζεται να γίνει από την αρχή για τους Μ χαρακτήρες απλά αφαιρείται το πρώτο πεδίο Μ χαρακτήρων και προστίθεται για υπολογισμό το Μ+1.

Τέλος αν οι τιμές he και hp είναι ίσες τότε ενδέχεται να υπάρχει ταύτιση, προς τούτο εκτελείται ο βρόχος ελέγχου και αυτό γίνετε γιατί πολλές φορές η ισότητα των τιμών hp και hk δεν εξασφαλίζει την ισότητα.

ΣΥΝΑΡΤΗΣΗ MOD

Στην προηγούμενη μέθοδο κατακερματισμού είδαμε ότι δημιουργούνταν η ανάγκη ενός δευτερογενούς ελέγχου, αν κατάφερνε η συνάρτηση κατακερματισμού να παράγει διαφορετικές τιμές για κάθε συμβολοσειρά τότε δεν θα υπήρχε η ανάγκη του δευτερογενή ελέγχου ταυτίσεις. Πάνω σε αυτή την ιδέα οι Rabin και Karp επινόησαν έναν αλγόριθμο στον οποίο ως συνάρτηση κατακερματισμού χρησιμοποιείται η mod η οποία μπορεί και παρέχει διαφορετικές τιμές για κάθε διαφορετικό συνδυασμό Μ χαρακτήρων μιας συμβολοσειράς

Ιδιότητες του τελεστή mod :

(a+b) mod q=[(a mod q)+(b mod q)]mod q

για παράδειγμα (8+12)mod 5=20 mod 5=0

8mod 5=3 και 12 mod 5=2, 2+3=5 και 5 mod 5=0

(a. b) mod q=[(a mod q). (b mode q)]mod q

για παράδειγμα (8. 12) mod 5=96 mod 5=1

8 mod 5=3 και 12 mod 5=2, 2. 3=6 και 6 mod 5=1

Στον αλγόριθμο αυτό πρέπει οι εκάστοτε Μ χαρακτήρες του κειμένου που αρχίζουν με τον i-στο χαρακτήρα και τελειώνουν στον i +M-1, θεωρούνται ως ένας αριθμός του 256-δικου συστήματος αρίθμησης. η συνάρτηση μετασχηματισμού μετατρέπει τον ορισμό αυτό στο δεκαδικό σύστημα με την γνωστή σχέση.

X=aibm-1 + ai+1bm-2+…+ai+m-2d+ai+m-1

Όπου b=256 η βάση του συστήματος

Οι Μ χαρακτήρες του κειμένου από τον i+1 μέχρι τον i +M μετατρέπονται σε:

χ=ai+1bm-1+ai+2bm-2+…+ai+m-1d+ai+m

πολλαπλασιάζοντας την (1) επί b έχουμε

bx = aibm+ai+1bm-1+…+ai+m-2d2+ai+m-1b

από τις δυο και τρία λαμβάνουμε

χ=bx-aibm +ai+m

Η σχέση αυτή παρέχει την τιμή χ των επόμενων Μ χαρακτήρων σε σχέση με την τιμή των πρώτων.

Η συνάρτηση μετασχηματισμού για το χ είναι

h=x mod q

όπου Χ q ένας μεγάλος πρώτος αριθμός

αντίστοιχα

h = x mod q

αντικαθιστώντας στην 6 το χ από την 4, λαμβάνοντας υπόψη την 5 και παίρνοντας συνεχώς τα υπόλοιπα με q κατασκευάζεται μια εξίσωση που παρέχει την τιμή h σε σχέση με την τιμή h. οι αναλυτικές σχέσεις φαίνονται στον αλγόριθμο που ακολουθεί.

ΣΥΜΒΟΛΟΣΕΙΡΕΣ ΣΤΙΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Ο χειρισμός συμβολοσειρών ποικίλει στις διάφορες γλώσσες προγραμματισμού. Μπορεί οι πρώτες γλώσσες προγραμματισμού να είχαν περιορισμένες δυνατότητες αλλά οι νεότερες γλώσσες προγραμματισμού διαθέτουν συναρτήσεις με τις οποίες επιτυγχάνονται διάφορες λειτουργίες επί των συμβολοσειρών. Όπως για παράδειγμα μπορούμε να έχουμε συνάρτηση που επιτυγχάνει την αποκοπή ενός δεξιού η αριστερού τμήματος της συμβολοσειράς η ακόμα και ενός ενδιάμεσου, επίσης μπορούμε να έχουμε συνάρτηση που δίνει την δυνατότητα παροχής του μήκους της συμβολοσειράς, την συνένωση μιας συμβολοσειράς η ακόμα και την ανεύρεση μιας υποσυμβολοσειράς.

4. Η γλώσσα προγραμματισμού PASCAL

Η ιστορία της Pascal

Η επιθυμία για μια νέα γλώσσα με σκοπό τη διδασκαλία προγραμματισμού, ξεκινάει από τη βαθιά μου απογοήτευση με τις χρησιμοποιούμενες ως τώρα γλώσσες. Τα χαρακτηριστικά και οι δομές των οποίων, πολύ συχνά δε μπορούν να εξηγηθούν λογικά και πειστικά. Αυτές πολύ συχνά αντιπροσωπεύουν μια προσβολή στη νοημοσύνη των ανθρώπων, εκπαιδευμένων σε συστηματική θεμελίωση. Παράλληλα με την απογοήτευση μου αυτή συμβαδίζει και η πεποίθηση μου ότι η γλώσσα, που ο μαθητής διδάσκεται, πρέπει να εκφράζει τις ιδέες του και να επηρεάζει βαθύτατα τις συνήθειες σκέψεων και εφευρετικότητας του. Η δυσαρμονία που κυριαρχεί σε αυτές τις γλώσσες αντανακλά στο στυλ προγραμματισμού που αυτός υιοθετεί. . » εκφράζει το 1973 ο Niklaus Wirth, καθηγητής του Πολυτεχνείου της Ζυρίχης ETH (Eidgenossische Technische Hochschule) και δημιουργός της γλώσσας προγραμματισμού της Pascal. Οι γλώσσες προγραμματισμού που υπαινίσσεται ο Niklaus Wirth δεν είναι άλλες παρά οι Fortran, Cobol, Algol και εν μέρει η PL/1.

Η Fortran, η επικρατέστερη γλώσσα για επιστημονικές εφαρμογές, ήταν τότε η πληρέστερη (σε στυλ) στις πραγματικές διαδικασίες της μηχανής. Η ταχύτητα και η αποδοτικότητα της Fortran, είναι ακόμη και σήμερα χαρακτηριστικές για τέτοιες εφαρμογές.

Η Cobol (αρχή δεκαετίας του 60) χρησιμοποιούσε προτάσεις της καθομιλουμένης στην Αγγλική γλώσσα και ενώ πρόσφερε δυνατότητες επεξεργασίας και δημιουργίας αρχείων υστερούσε σε δυνατότητες δομημένου προγραμματισμού και επιστημονικών εφαρμογών.

Η Algol (αρχή δεκαετίας του 60) ήταν η πρώτη γλώσσα που διευκόλυνε διατυπώνοντας βήμα προς βήμα την αλγοριθμική λύση ενός προβλήματος. Η αδυναμία της σε δομές στοιχείων για δημιουργία και επεξεργασία αρχείων καθιστούσε άμεση την εξάρτηση της από τη μηχανή πάνω στην οποία

γραφόταν το πρόγραμμα. Αυτό αποτέλεσε και το βασικότερο μειονέκτημα της. Όλα αυτά τα κενά που άφηναν οι παραπάνω γλώσσες προγραμματισμού αποτέλεσαν και το πάτημα για την εξέλιξη και τη δημιουργία νέων γλωσσών προγραμματισμού. Στα μέσα με τέλος της δεκαετίας του 60 έκανε την εμφάνιση της η PL/1, η Basic και η Algol 60. Η PL/1 συγκέντρωνε όλα τα χαρακτηριστικά και τις δυνατότητες των προγενέστερων γλωσσών προγραμματισμού. Το βασικό της μειονέκτημα, ήταν ότι έθετε πολλές απαιτήσεις στον Compiler της γλώσσας με αποτέλεσμα να τον καθιστά απρόσιτο για τις περισσότερες μεσαίες ή μικρές μηχανές. Αποτέλεσε όμως λιθαράκι για τη δημιουργία νέων γλωσσών προγραμματισμού. Ακολούθησε η Basic, η οποία απευθυνόταν σε μη επαγγελματίες. Η απλή δομή της την καθιέρωσε ως την πιο δημοφιλή γλώσσα στον κόσμο των μικροϋπολογιστών στη δεκαετία του 80. Στα μέσα της δεκαετίας του 60 η Διεθνής Ομοσπονδία για Επεξεργασία πληροφοριών γνωστή σαν IFIP, άρχισε την επανασχεδίαση της Algol 60 η οποία περατώθηκε μερικά χρόνια αργότερα με το όνομα Algol 68. Σε αυτή την ομάδα των σοφών άνηκε και ο Wirth, ο οποίος ονειρευόταν μια γλώσσα προγραμματισμού με δυνατά σημεία και παράλληλα να μπορεί να χρησιμοποιηθεί σε μικρές μηχανές. Με αυτό το όραμα δημιουργήθηκε η Pascal, η οποία είναι μια από τις πιο δημοφιλής γλώσσες προγραμματισμού. Αναπτύχθηκε στα τέλη της δεκαετίας του 1960 και πήρε το όνομα της από το Γάλλο μαθηματικό του 17ου αιώνα, Blaise Pascal (1623-1662), ο οποίος κατασκεύασε μια από τις πρώτες μηχανές άθροισης, βάζοντας με αυτό τον τρόπο το δικό του λιθαράκι στην ιστορία των υπολογιστών.

Η γλώσσα προγραμματισμού της Pascal είναι μια γλώσσα υψηλού επιπέδου, εύκολα κατανοητή από τον άνθρωπο και κατάλληλη για την εκπαίδευση και δημιουργία δομημένων προγραμμάτων. Είναι γλώσσα για σειριακό προγραμματισμό και έγινε γρήγορα δημοφιλής για τους παρακάτω λόγους:

- Είναι απλή και εύκολη στην εκμάθηση.
- Περιλαμβάνει ένα ικανοποιητικό σύνολο εντολών ελέγχου.
- Διαχειρίζεται πολλούς τύπους δεδομένων.

- Είναι γλώσσα γενικού σκοπού και εφαρμόσιμη σε πολλές κατηγορίες προβλημάτων.
- Για τα περισσότερα υπολογιστικά συστήματα υπάρχουν διαθέσιμοι μεταφραστές της Pascal.
- Είναι καλοσχεδιασμένη εισαγωγική γλώσσα προγραμματισμού, ιδανική για τη διδασκαλία των βασικών αρχών της επιστήμης των υπολογιστών και θεωριών του προγραμματισμού.
- Τα προγράμματα της Pascal μπορούν να γραφούν με έναν πολύ οργανωμένο και ισχυρά δομημένο τρόπο, δίνοντας έτσι τη δυνατότητα για καλό έλεγχο και εύκολη τροποποίηση.

Η άνοδος της δημοτικότητας της, οφείλεται κυρίως στους παρακάτω λόγους:

- Στην ανάπτυξη των μικροϋπολογιστών.
- Στην ίδρυση νέων εταιρειών.
- Στα νέα στελέχη που προέρχονται κυρίως από πανεπιστήμια.
- Ο δομημένος κώδικας και η πλήρης κάλυψη των αρχών δομημένου προγραμματισμού, που κάθε πρόγραμμα απαιτεί, αποτελεί το βασικό αλλά και το πιο ισχυρό της πλεονέκτημα.

Το μεγαλύτερο πλεονέκτημα της γλώσσας αυτής, είναι ότι δίνει τη δυνατότητα σε κάποιον να διδαχθεί και να εκμεταλλευτεί τεχνικές προγραμματισμού. Η Pascal μπορούσε να χρησιμοποιηθεί με ένα συστηματικό τρόπο για την εκμάθηση του προγραμματισμού και μπορούσε να έχει τέτοια μορφή ώστε να αναπαρασταθεί σε κάποιον ηλεκτρονικό υπολογιστή. Το πρώτο βιβλίο εκδόθηκε το 1974 από τους K. Jensen- N. Wirth: «Pascal User Manual and report»Το 1983 έκανε την εμφάνιση της και η Turbo Pascal, προϊόν της εταιρείας Borland, η οποία παρήγαγε καθαρό εκτελέσιμο κώδικα μηχανής σε αντίθεση με την προγενέστερη που παρήγαγε ενδιάμεσο ψευδοκώδικα.

Η Pascal είναι ευρέως γνωστή ως μια από τις πιο κατάλληλες γλώσσες προγραμματισμού για τη διδασκαλία των τεχνικών του δομημένου προγραμματισμού (structured programming techniques).

Στόχοι της γλώσσας αυτής είναι :

- Να εκφράζονται οι Αλγόριθμοι του οποιοδήποτε προς μηχανογράφηση προβλήματος με ένα φυσικό τρόπο διατύπωσης.
- Να υπάρχουν διάφοροι χρήσιμοι τύποι στοιχείων, δηλαδή ευελιξία στην εσωτερική παράσταση των στοιχείων στη μνήμη, που κατ' επέκταση σημαίνει αποδοτικότερη αξιοποίηση της μνήμης.
- Να είναι όσο το δυνατόν σαφής, συναφής και λογική στη διατύπωση της, καθώς επίσης να είναι δυνατή η αποδοτική μετάφραση της και η εκτέλεση της σε οποιαδήποτε μηχανή.

Αυτή η τελευταία πινελιά συντέλεσε στο να γίνει μια από τις πιο δημοφιλής γλώσσες προγραμματισμού.

<u>Διάλεκτοι</u>

Λόγω της μεγάλης εξάπλωσης της εμφανίστηκαν με το πέρασμα των χρόνων οι παρακάτω εκδόσεις της Pascal.

- Small Pascal, Ο ίδιος ο Wirth τη παρουσίασε το 1981 για διδακτικούς σκοπούς.
- P-ΚΩΔΙΚΑΣ (1976)
- Ucsd Pascal
- Turbo Pascal
- M-S Pascal
- Pascal 1000
- Pascal / Z

Εμείς θα ασχοληθούμε με την Turbo Pascal, που είναι μια από τις πιο διαδεδομένες διαλέκτους για μικρό-υπολογιστές.

Κανόνες

Απαραίτητη είναι η παρουσία της επικεφαλίδας. Δηλώνει την ονομασία του προγράμματος. Αυτή είναι μια και μοναδική λέξη και μπορεί να είναι οποιαδήποτε, πχ program, test. . .

- Μία εντολή μπορεί να διασπαστεί σε περισσότερες από μια γραμμές.
- Πολλές εντολές μπορεί να υπάρχουν σε μια γραμμή.
- Ανάμεσα στα ονόματα δε μπορούν να υπάρχουν κενά.
- Μία εντολή αρχίζει με * και τελειώνει με *.
- Το ερωτηματικό ; (ημιπερίοδος) χρησιμοποιείται για να χωρίζει τις εντολές μεταξύ τους.
- Η αρχή ενός προγράμματος δηλώνεται με την λέξη begin.
- Το τέλος ενός προγράμματος δηλώνεται όταν η εκτέλεση φθάσει στο τελευταίο end και ακολουθείται τελεία.

Ανάμεσα στις εντολές Begin και end μπορούν να χρησιμοποιηθούν ενδιάμεσα και οι παρακάτω εντολές, if, then, else, case of, repeat, until, while, do, for, to, down to, go to, with.

Για την επιλογή από τον προγραμματιστή ενός ονόματος στην Pascal (ως μεταβλητής, σταθεράς, προγράμματος, συνάρτησης, κλπ.) υπάρχουν οι εξής κανόνες:

- Δεν πρέπει το όνομα να είναι δεσμευμένη λέξη της Pascal.
- Επιτρέπονται μόνο αλφαριθμητικοί χαρακτήρες.
- Επιτρέπονται μόνο λατινικά γράμματα.
- Ο πρώτος χαρακτήρας πρέπει να είναι λατινικό γράμμα ή _ (όχι ψηφίο).

4.1. Αλφάβητο

Αποτελείται από τα 26 γράμματα του αγγλικού αλφαβήτου a-z, από τα δέκα ψηφία 0-9 και από τα ειδικά σύμβολά όπως τα παρακάτω :

Σύμβολα αριθμητικών πράξεων: + - * / div και mod (πηλίκο και υπόλοιπο ατελούς διαίρεσης, αντίστοιχα)

Σύμβολα σχέσης: > < <= >= <> (Διάφορο) in (μπαίνει ανάμεσα σε εντολές)

Σύμβολα λογικών πράξεων: or and not (ή το ένα ή το άλλο)

Σύμβολα σχολίων : { } (* *)

Ειδικοί χαρακτήρες: () . , : ; . . := (Τελεστής ανάθεσης τιμής)

4.2. Λεξιλόγιο

Το λεξιλόγιο της Pascal περιλαμβάνει 8 κατηγορίες:

- Ονόματα

  Τα ονόματα ορίζονται από τον προγραμματιστή και επιλέγονται με τέτοιο τρόπο ώστε να υποδουλώνουν τη σημασία της ποσότητας που προσδιορίζουν.

  Αποτελούνται από γράμματα τα οποία μπορούν να συνοδεύονται από αριθμούς π. χ. value1, επιτρέπεται η χρήση του χαρακτήρα _ αλλά όχι της –, πχ. Mathima_1 και όχι mathima -1. Το μήκος είναι θεωρητικά απεριόριστο, ενώ δεν επιτρέπεται η χρήση ονομάτων που έχουν ιδιαίτερη σημασία για την Pascal.

- Αριθμοί

  Η Pascal υποστηρίζει δύο τύπους αριθμών: τους ακέραιους και τους πραγματικούς.

  Στους ακέραιους περιλαμβάνονται τα ψηφία, τα οποία ακολουθούνται από τα σύμβολα +, - , όπως +45, -45850 ενώ δεν επιτρέπεται η χρήση των αριθμών που περιέχουν σύμβολα όπως 43, 600.

  Στους πραγματικούς περιλαμβάνονται αριθμοί θετικοί ή αρνητικοί, οι οποίοι βρίσκονται έξω από τα όρια των ακέραιων ή περιέχουν δεκαδικά ψηφία. Μπορούν να περιέχουν υποδιαστολή, αρκεί να ακολουθούν δεκαδικά ψηφία, όπως 23. 56 ενώ η τιμή 23. δεν περιέχει δεκαδικά ψηφία. Επίσης μπορεί να είναι αριθμοί οι οποίοι είναι εκφρασμένοι σε εκθετική μορφή, όπως π. χ. ο αριθμός 38E27 συμβολίζει τον 38x10 στην 27η.

- Χαρακτήρες

  Είναι χαρακτήρες από το σύνολο των χαρακτήρων που υποστηρίζει το υπολογιστικό σύστημα μέσα σε ‘ ‘. Σημειώσεις: Αν θελήσουμε να δηλώσουμε τον χαρακτήρα απόστροφο, θα πρέπει να τον δηλώσουμε μέσα σε ‘’ δηλαδή : ’’’’. Ο χαρακτήρας ‘Μ’ διαφέρει από τον χαρακτήρα

‘m’. Ο κενός χαρακτήρας συμβολίζεται με ένα κενό ανάμεσα σε αποστρόφους. Δηλαδή: ‘ ‘ .

§ Αλυσίδες Χαρακτήρων
Είναι μια ακολουθία χαρακτήρων μεταξύ αποστρόφων. Δηλαδή: ‘programma _fisikis‘1’’.

§ Τελεστές

Ø Αριθμητικοί τελεστές

Χρησιμοποιούνται για πράξεις μεταξύ συνόλων ή για αριθμητικές λειτουργίες μεταξύ αριθμητικών δεδομένων. Είναι οι εξής: + - * / div (πηλίκο διαίρεσης) mod (υπόλοιπο ατελούς διαίρεσης)

Ø Τελεστές σύγκρισης ή Σχεσιακοί Τελεστές:

Χρησιμοποιούνται για σύγκριση δύο αριθμητικών ή αλφαριθμητικών τιμών. Είναι οι εξής: > < <= >= <> (Διάφορο) in (μπαίνει ανάμεσα σε εντολές)

Ø Λογικοί Τελεστές

Χρησιμοποιούνται για την εκτέλεση λογικών ή Boolean πράξεων. Είναι οι εξής: or (διάζευξη), and (λογική πρόθεση), not (αρνητική πρόθεση).

Ø Διαχωριστές

Είναι χαρακτήρες με συγκεκριμένη λειτουργία στην Pascal όπως () . , : ; . . :=

§ Τελεστής ανάθεσης τιμής:

Ο τελεστής αυτός είναι ο « := » και χρησιμοποιείται για την ανάθεση της τιμής μιας έκφρασης σε μια μεταβλητή.

Σύνταξη: Όνομα μεταβλητής := Έκφραση

Παράδειγμα:

Με την εντολή x:=5; δίνεται στη μεταβλητή x η τιμή 5. Ο τελεστής αυτός είναι δυαδικός, δηλαδή έχει δύο τελεστές, έναν αριστερά του και έναν δεξιά του. Ο αριστερά τελεστής είναι υποχρεωτικά το όνομα μιας μεταβλητής, ενώ ο δεξιά τελεστής μπορεί να είναι το όνομα μιας άλλης μεταβλητής, μια σταθερά ή μια έκφραση. Οι δύο τελεστές πρέπει συνήθως να είναι του ίδιου τύπου (π. χ. και οι δύο ακέραιοι - integer).

Σημείωση: Στα μαθηματικά δεν ισχύει x=x+1, στον προγραμματισμό όμως ισχύει το x:=x+1.

Αριστερά από το := είναι πάντα το άγνωστο μέρος το οποίο θα πάρει τιμή, ενώ δεξιά το γνωστό μέρος. Όταν βλέπουμε μια μεταβλητή να είναι και από τη δεξιά μεριά του := και από την αριστερή, δίνουμε αρχική τιμή πιο πριν σε αυτή την μεταβλητή.

- Λέξεις κλειδιά

  Εννοούμε λέξεις με ειδική σημασία. Αυτές είναι: and, array, begin, case, const, div, do, down to, else, end, file, for, function, goto, if, in, label, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, then, to, type, until, var, while, with.

- Σχόλια

  Σημαντικό στοιχείο για ένα πρόγραμμα είναι η ύπαρξη σχολίων, τα οποία βελτιώνουν την αναγνωσιμότητα του πηγαίου κώδικα και κάνουν τη συντήρησή του πιο εύκολη. Τα σχόλια περικλείονται σε άγκιστρα, π. χ. {auto einai to proto sxolio}, ή σε παρενθέσεις με αστερίσκο, π. χ. (*auto einai to deutero sxolio *), και μπαίνουν οπουδήποτε στο πρόγραμμα, χωρίς περιορισμούς σχετικά με το περιεχόμενό τους.

Μεταβλητές και Σταθερές

Η Pascal είναι non case-sensitive γλώσσα (μη ευαίσθητη σε πεζά - κεφαλαία), δηλαδή τα πεζά και τα κεφαλαία έχουν την ίδια ακριβώς σημασία. Για

παράδειγμα, τα ονόματα sum, Sum και SUM αναφέρονται στην ίδια ακριβώς μεταβλητή.
Οι σταθερές διακρίνονται σε αριθμητικές (π. χ. 1, 2, 1. 6) και σε αλφαριθμητικές, οι οποίες περικλείονται σε αποστρόφους (π. χ. 'a', 'hello', 'kalimera', '12345', '$@#42a'). Για τις αλφαριθμητικές σταθερές δεν υπάρχουν περιορισμοί σχετικά με το περιεχόμενό τους (μπορεί να είναι οποιοδήποτε σύμβολο ή γράμμα σε οποιαδήποτε γλώσσα). Προτείνεται η χρησιμοποίηση του λατινικού αλφαβήτου γιατί σε περίπτωση λάθους κρίνεται δύσκολη η ανίχνευση του λόγω ομοιοτήτων με το ελληνικό.

Η Pascal διαθέτει ενσωματωμένους τρεις βασικούς τύπους δεδομένων: τα αριθμητικά δεδομένα, τα δεδομένα χαρακτήρα και τα λογικά δεδομένα.

Στους αριθμητικούς τύπους περιλαμβάνονται:

§ Πραγματικοί αριθμοί

Στην Pascal οι πραγματικοί αριθμοί δηλώνονται ως real (μια μεταβλητή αυτού του τύπου καταλαμβάνει 6 bytes στη μνήμη), single (4 bytes), double (8 bytes) και extended (10 bytes). Όσο μεγαλύτερο χώρο καταλαμβάνει ένας πραγματικός αριθμός στη μνήμη, τόσο μεγαλύτερος θα μπορεί να είναι και μεγαλύτερη ακρίβεια να έχει.

§ Ακέραιοι αριθμοί

Στην Pascal οι ακέραιοι αριθμοί δηλώνονται ως integer (μια μεταβλητή αυτού του τύπου καταλαμβάνει 16 bits, εκ των οποίων το ένα bit χρησιμοποιείται ως πρόσημο, άρα οι τιμές που μπορεί να πάρει είναι από -215 = -32768 μέχρι 215 - 1 = 32767), shortint 98 bits με πρόσημο και με τιμές από -128 μέχρι 127), longint (32 bits με πρόσημο και με τιμές από -2. 147. 483. 648 μέχρι 2. 147. 483. 647), byte (8 bits χωρίς πρόσημο και με τιμές από 0 μέχρι 255) και word (16 bits χωρίς πρόσημο και με τιμές από 0 μέχρι 65535).

Τα δεδομένα τύπου χαρακτήρα διακρίνονται σε δεδομένα ενός χαρακτήρα char και σε δεδομένα πολλών χαρακτήρων string, που μπορεί να είναι από 1 μέχρι 255 χαρακτήρες.

Τα λογικά δεδομένα δηλώνονται στην Pascal ως Boolean και έχουν το χαρακτηριστικό ότι μπορούν να πάρουν δύο τιμές, τις TRUE (αληθής) και FALSE (ψευδής). Οι τύποι integer, char και Boolean λέγονται και τακτικοί τύποι, γιατί υπάρχει διάταξη μεταξύ των μελών τους (καθένα από τα μέλη έχει προηγούμενο και επόμενο). Με βάση αυτούς τους τύπους, μπορούν να σχηματιστούν και σύνθετοι τύποι, δηλαδή ομάδες δεδομένων, όπως οι πίνακες (array) και οι εγγραφές (records). Επιπλέον, ο προγραμματιστής έχει τη δυνατότητα να ορίσει και δικούς του τύπους δεδομένων, χρησιμοποιώντας στο τμήμα δηλώσεων τη δεσμευμένη λέξη type. Για παράδειγμα, με τη δήλωση type month_number = 1. . 12; ορίζεται ένας νέος τύπος δεδομένων με όνομα month_number. Οι μεταβλητές του τύπου αυτού θα παίρνουν ακέραιες τιμές από 1 μέχρι 12. Ο συμβολισμός. . . δηλώνει διάστημα από - έως. Ένας τύπος στην Pascal, και σε πολλές άλλες δημοφιλείς γλώσσες προγραμματισμού, καθορίζει μια μεταβλητή με τέτοιο τρόπο που κατ' αρχήν προσδιορίζει το εύρος των τιμών που είναι ικανή να αποθηκεύσει, και επίσης καθορίζει ένα σύνολο από λειτουργίες που είναι επιτρεπτό να εφαρμοσθούν πάνω στις μεταβλητές αυτού του τύπου.

Integer	Όλοι οι αριθμοί στο διάστημα από -32768 έως 32767
Byte	Όλοι οι ακέραιοι από 0 έως 255
Real	Όλοι οι πραγματικοί αριθμοί από 1E-38 έως 1E+38
Boolean	Μπορεί να έχει μόνο τιμές TRUE και FALSE
Char	Όλοι οι χαρακτήρες του κώδικα ASCII
Shortint	Όλοι οι ακέραιοι από -128 έως 127
Word	Όλοι οι ακέραιοι από 0 έως 65535
Longint	Όλοι οι ακέραιοι από -2147483648 έως 2147483647

Single	Όλοι οι πραγματικοί με 7 σημαντικά ψηφία
Double	Όλοι οι πραγματικοί με 15 σημαντικά ψηφία
Extended	Όλοι οι πραγματικοί με 19 σημαντικά ψηφία
Comp	Όλοι οι ακέραιοι από -10E18 έως 10E18

ΔΗΛΩΣΗ ΜΕΤΑΒΛΗΤΩΝ

Η μεταβλητή δηλώνεται στην αρχή της εκτέλεσης και δηλώνεται με το χαρακτηρισμό var.

Μόνο μια var μπορεί να υπάρχει στο πρόγραμμα.

Πχ var mathites : integer;

Omada 1, omada 2: char;

ΔΗΛΩΣΗ ΧΑΡΑΚΤΗΡΩΝ

Ένα όνομα, που έχει δηλωθεί char, σε μια δήλωση var μπορεί να πάρει τιμές που αποτελούνται από ένα χαρακτήρα.

4.3. Δομές Επιλογής και Επανάληψης

Στην Pascal υπάρχουν τρεις βασικές προγραμματιστικές δομές:

Η δομή της ακολουθίας των εντολών, στην οποία οι εντολές εκτελούνται με τη σειρά που αναγράφονται και κατεύθυνση από πάνω προς τα κάτω.

Η δομή της επιλογής ή απόφασης, στην οποία ο έλεγχος της ροής του προγράμματος αποφασίζεται από την τιμή μιας λογικής έκφρασης. Αν η τιμή της λογικής έκφρασης είναι αληθής (TRUE) εκτελείται το τμήμα Α του προγράμματος, αν είναι ψευδής (FALSE) εκτελείται το τμήμα Β του προγράμματος.

Η δομή της επανάληψης, στην οποία ένα τμήμα του προγράμματος εκτελείται κατ' επανάληψη ανάλογα με την τιμή μιας λογικής έκφρασης.

ΕΝΤΟΛΕΣ ΕΙΣΟΔΟΥ-ΕΞΟΔΟΥ

Για εισαγωγή δεδομένων στο πρόγραμμα από το χρήστη (μέσω του πληκτρολογίου) και εμφάνιση των αποτελεσμάτων στην οθόνη, χρησιμοποιούμε τις διαδικασίες εισόδου και εξόδου (procedures) read και write (ή readln και writeln όταν θέλουμε να διαβάζει μαζί με την τιμή που θα δώσουμε και το enter, ή να αλλάζει γραμμή μετά την εκτέλεση της writeln, αντίστοιχα).

Η «writeln» είναι μια ειδική λέξη που επιτρέπει να εμφανιστούν κάποια δεδομένα στην οθόνη του υπολογιστή μας. Τα δεδομένα μέσα στην παρένθεση είναι τα δεδομένα που θα εκτυπωθούν. Οποιαδήποτε πληροφορία ανάμεσα στις αποστρόφους θα εκτυπωθεί ως απλό κείμενο. Η ειδική λέξη «writeln» δεν είναι δεσμευμένη λέξη αλλά είναι καθορισμένη από το σύστημα να εκτυπώνει μια γραμμή από δεδομένα στην οθόνη.

Η «write» είναι μια άλλη ειδική λέξη η οποία υποχρεώνει το κείμενο να εκτυπωθεί όπως και με την «writeln», αλλά ο κέρσορας δεν αλλάζει γραμμή στην έξοδο. Η «writeln» γράφει το προς εκτύπωση κείμενο και αλλάζει γραμμή, η «write» όχι.

Παραδείγματα

read(a);	Διαβάζει τη μεταβλητή a. Ο χρήστης πρέπει να δώσει από το πληκτρολόγιο του Η/Υ μια τιμή για τη μεταβλητή a. Αν η μεταβλητή έχει δηλωθεί integer θα πρέπει ο χρήστης να δώσει ακέραια τιμή για το a.
read(a, b);	Ο χρήστης θα πρέπει να δώσει από το πληκτρολόγιο του Η/Υ δυο τιμές χωρισμένες με ένα κενό.
write('KALHMERA');	Εμφανίζει στην οθόνη τη λέξη KALHMERA και ο δρομέας βρίσκεται ακριβώς δίπλα από τη λέξη.
writeln('KALHMERA');	Εμφανίζει στην οθόνη τη λέξη KALHMERA και ο δρομέας πηγαίνει στην ακριβώς από κάτω γραμμή από τη λέξη.
write('A = ', a);	Εμφανίζει στην οθόνη τη λέξη A = και αμέσως μετά την τιμή της μεταβλητής a. Το σχόλιο από τη μεταβλητή χωρίζεται με ένα κόμμα.
write(a:5:2);	Εμφανίζει στην οθόνη τη τιμή του πραγματικού αριθμού a σε πέντε θέσεις, από τις οποίες οι δύο είναι τα δεκαδικά ψηφία και η μία η υποδιαστολή. Αν οι θέσεις που μένουν δεν επαρκούν για το ακέραιο μέρος, θα χρησιμοποιηθούν περισσότερες χωρίς καμία παρέμβαση από τον χρήστη.
writeln(1, 3*3, 4+2, 6-1);	Εμφανίζει στην οθόνη τις τιμές 1 9 6 και 5 χωρίς κενά μεταξύ τους και ο δρομέας πηγαίνει στην από κάτω γραμμή.

writeln(1, ' ', 2, ' '3*2);	Εμφανίζει στην οθόνη τις τιμές 1 2 και 6 με ένα κενό μεταξύ τους και ο δρομέας πηγαίνει στην από κάτω γραμμή.
writeln(a+b);	Εμφανίζει στην οθόνη την τιμή του αθροίσματος a+b και ο δρομέας πηγαίνει στην από κάτω γραμμή.
writeln('a+b');	Εμφανίζει στην οθόνη τη λέξη (σχόλιο) a+b και ο δρομέας πηγαίνει στην από κάτω γραμμή.

ΕΝΤΟΛΗ ΚΑΤΑΧΩΡΗΣΗΣ

Αυτή η εντολή έχει τη μορφή :

μεταβλητή : = μάθημα

ΕΝΤΟΛΗ Begin-End

Οι εντολές begin και end περικλείουν μια ενότητα. Καλούνται παρενθέσεις εντολών. Μία ενότητα θεωρείται εντολή. Εντολές μιας ενότητας μπορούν να θεωρηθούν και αυτές ενότητες. Δηλώνουν την αρχή μιας ενότητας και το τέλος της.

Begin

Εντολή;

Εντολή;

…….

End;

Η ΕΝΤΟΛΗ – IF

Απλή επιλογή

Σκοπός της είναι να εκτελέσει μια πρόταση, αν ισχύει κάποια συνθήκη ή επιλογή ανάμεσα σε δύο προτάσεις.

Η εντολή χρησιμοποιεί τρεις δεσμευμένες (IF, THEN, ELSE).

Μετά το IF ακολουθεί η συνθήκη, η οποία τελειώνει πριν από τη δεσμευμένη λέξη THEN.

Μετά το THEN και πριν από τη λέξη ELSE, γράφεται η εντολή1 ή οι εντολές της ομάδας 1.

Η εντολή2 ή οι εντολές της ομάδας 2 γράφονται αμέσως μετά τη βοηθητική λέξη ELSE.

```
IF <συνθήκη> THEN
εντολή1
ELSE
εντολή2;
```

Σημείωση: Αν υπάρχουν περισσότερες από μια εντολές πρέπει να τις βάλουμε ανάμεσα σε BEGIN. . . END

```
IF <συνθήκη> THEN
  BEGIN
ομάδα εντολών 1
   END
ELSE
 BEGIN
ομάδα εντολών 2
   END;
```

Η δομή επιλογής εκτελείται ως εξής:

Αρχικά ελέγχεται η συνθήκη. Στη συνέχεια υπάρχουν δύο δυνατές περιπτώσεις. Αν η συνθήκη ικανοποιείται (η απάντηση είναι ναι), εκτελείται μόνο η εντολή1 ή η ομάδα εντολών 1. Η εντολή2 ή ομάδα εντολών 2 αγνοείται και ο έλεγχος του αλγόριθμου βγαίνει από τη δομή επιλογής. Σε αντίθετη περίπτωση, αν δηλαδή η συνθήκη δεν ικανοποιείται (η απάντηση είναι όχι), δε λαμβάνεται υπόψη η εντολή1 ή η ομάδα εντολών 1. Εκτελείται μόνο η εντολή2 ή η ομάδα εντολών 2 και η εκτέλεση της δομής επιλογής τελειώνει.

Περιορισμένη επιλογή

Υπάρχει περίπτωση σε μια εντολή επιλογής, στη περίπτωση που ισχύει η συνθήκη, να εκτελείται μια εντολή ή μια σειρά εντολών, ενώ στην περίπτωση που δεν ισχύει η συνθήκη να μη θέλουμε να εκτελεστεί κάποια εντολή ή εντολές. Στη περίπτωση αυτή χρησιμοποιούμε τη δομή περιορισμένης επιλογής.

```
IF <συνθήκη> THEN
εντολή 1;

IF <συνθήκη> THEN
  BEGIN
ομάδα εντολών 1
  END;
```

Στη δομή περιορισμένης επιλογής, η δεσμευμένη λέξη ELSE, καθώς και η εντολή2 ή η ομάδα εντολών 2 παραλείπονται.

Η ΕΝΤΟΛΗ – CASE

Σκοπός αυτής της εντολής είναι η επιλογή μιας πρότασης, ανάμεσα από μια ομάδα προτάσεων, σύμφωνα με τη τιμή μιας έκφρασης. Σε πολλές περιπτώσεις, υπάρχουν περισσότερες εναλλακτικές επιλογές, που εξαρτώνται από την τιμή μιας μεταβλητής. Σε μια εντολή case δίνονται ένα πλήθος από εναλλακτικές εντολές. Κατά την εκτέλεση επιλέγεται μια από αυτές και εκτελείται. Η εντολή case, έχει την παρακάτω μορφή:

CASE <έκφραση> OF

<τιμή 1>: <εντολή 1>;

<τιμή 2>: <εντολή 2>;

. . .

<τιμή n>: <εντολή n>;

END;

Υπολογίζεται η έκφραση και συγκρίνεται με κάθε τιμή της λίστας των πιθανών τιμών της έκφρασης. Εκτελείται εκείνη η εντολή, στην οποία η τιμή συμπίπτει με την τιμή της έκφρασης, και ο έλεγχος του προγράμματος μεταφέρεται στην εντολή που ακολουθεί το end. Η έκφραση και οι πιθανές τιμές της πρέπει απαραίτητα να είναι τακτικού τύπου.

Στην εντολή case μπορεί να υπάρχει πριν το end μια εναλλακτική εντολή που θα εκτελεστεί όταν καμία τιμή δεν συμπίπτει με την τιμή της έκφρασης. Η εντολή αυτή συντάσσεται με else:

CASE <έκφραση> OF

<τιμή 1>: <εντολή 1>;

<τιμή 2>: <εντολή 2>;

. . .

<τιμή k>: <εντολή k>

ELSE <εντολή n>;

END;

Σημείωση: Αν υπάρχουν περισσότερες από μια εντολές πρέπει να τις βάλουμε ανάμεσα σε BEGIN. . . END

ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ

Η εντολή FOR

Η εντολή for χρησιμοποιείται όταν γνωρίζουμε το πλήθος των εντολών, και μπορεί να έχει μια από τις ακόλουθες μορφές:

FOR < μτ >:=< ατ > TO < ττ > DO <εντολή1>;

FOR < μτ >:=< ατ > DOWNTO < ττ > DO <εντολή1>;

FOR < μτ >:=< ατ > TO < ττ > DO

Begin

ομάδα εντολών;

End;

Εδώ χρησιμοποιούνται οι δεσμευμένες λέξεις FOR, TO (DOWNTO), DO και οι μεταβλητές-σταθερές μτ (μετρητής), ατ (αρχική τιμή), ττ (τελική τιμή). Οποιαδήποτε απλή μεταβλητή με τύπο integer, byte ή char μπορεί να

χρησιμοποιηθεί ως δείκτης επανάληψης με την προϋπόθεση ότι πρέπει να υπάρχει μεταβλητή που να είναι δηλωμένη ως var. Αυτή η μορφή της επαναληπτικής δομής έχει το χαρακτηριστικό ότι δεν απαιτείται στην ομάδα εντολών να υπάρχουν εντολές μεταβολής της τιμής του μετρητή, η οποία μεταβάλλεται από την ίδια την εντολή σε κάθε επανάληψη κατά 1 στην περίπτωση του TO ή κατά -1 όταν υπάρχει το DOWNTO.

Η εντολή WHILE – DO

Σε κάποια προγράμματα όπου δεν έχει οριστεί ο αριθμός των επαναλήψεων χρησιμοποιείται η εντολή while. Σκοπός της εντολής αυτής είναι η επαναληπτική εκτέλεση μιας πρότασης, όσο ισχύει μια δεδομένη συνθήκη. Η σύνταξη της εντολής μπορεί να έχει μια από τις ακόλουθες μορφές:

WHILE <συνθήκη> DO

<εντολή1>;

WHILE <συνθήκη> DO

Begin

ομάδα εντολών;

End;

Εδώ χρησιμοποιούνται οι δεσμευμένες λέξεις WHILE, DO. Μετά τη λέξη WHILE ακολουθεί η συνθήκη συνέχειας. Αμέσως μετά την λέξη DO γράφεται η εντολή ή η ομάδα εντολών. Οι συνθήκες που χρησιμοποιούνται εδώ είναι ίδιες με τις συνθήκες που χρησιμοποιούνται στην εντολή επιλογής. Στην αρχή της εκτέλεσης ελέγχεται η συνθήκη. Αν αυτή ικανοποιείται (true), τότε εκτελείται η εντολή ή η ομάδα εντολών. Στη συνέχεια, ελέγχεται ξανά η συνθήκη. Δεν είναι σίγουρο ότι αυτή εξακολουθεί να είναι αληθής διότι το αποτέλεσμά της μπορεί να έχει επηρεαστεί από την εκτέλεση της εντολής ή της ομάδας εντολών. Αν η συνθήκη ικανοποιείται ξανά, εκτελείται πάλι η εντολή ή η ομάδα εντολών. Η

διαδικασία συνεχίζεται, όσο η συνθήκη παραμένει αληθής. Όταν η συνθήκη συνέχειας γίνει ψευδής, η εντολή ή η ομάδα εντολών δεν εκτελείται και τότε λήγει η εκτέλεση της δομής.

Η εντολή REPEAT – UNTIL

Η εντολή αυτή καθορίζει το πόσες φορές θα επαναληφθεί η εντολή.

Η σύνταξη της εντολής μπορεί να έχει μια από τις ακόλουθες μορφές:

REPEAT

<εντολή1>;

UNTIL <συνθήκη>;

REPEAT

ομάδα εντολών;

UNTIL <συνθήκη>;

Εδώ χρησιμοποιούνται οι δεσμευμένες λέξεις REPEAT, UNTIL. Μετά τη λέξη REPEAT ακολουθούν οι εντολές οι οποίες τελειώνουν πριν από τη λέξη UNTIL. Και στη περίπτωση αυτή, οι συνθήκες που χρησιμοποιούνται είναι ίδιες με τις συνθήκες που χρησιμοποιούνται στην εντολή επιλογής. Η εντολή τελειώνει με το UNTIL <συνθήκη>. Σε αυτή τη μορφή της εντολής επανάληψης ο έλεγχος γίνεται μετά την εκτέλεση της ομάδας εντολών. Αυτό σημαίνει ουσιαστικά ότι η ομάδα εντολών εκτελείται τουλάχιστον μία φορά. Μετά την εκτέλεση της ομάδας εντολών ελέγχεται η συνθήκη. Αν αυτή δεν ικανοποιείται, τότε εκτελείται ξανά η ομάδα εντολών, όσο η συνθήκη παραμένει ψευδής. Όταν η συνθήκη γίνει αληθής, λήγει η εκτέλεση της επαναληπτικής δομής.

4.4. Διαδικασίες και Συναρτήσεις

Τις περισσότερες φορές ένα μεγάλο πρόγραμμα μπορεί να γίνει με τη δημιουργία πολλών μικρότερων υποπρογραμμάτων. Αυτό μας προσφέρει αρκετά πλεονεκτήματα όπως:

- Η συντήρηση των προγραμμάτων γίνεται ευκολότερη.
- Η δυνατότητα χρήσης των υποπρογραμμάτων που εκτελούν συνηθισμένες εργασίες μπορούν να μεταφερθούν και σε άλλα προγράμματα - εφαρμογές.
- Η δομή του προγράμματος γίνεται εύκολα κατανοητή.
- Η δυνατότητα ανάπτυξης μικρότερων προγραμμάτων από ανεξάρτητους προγραμματιστές, με σκοπό τη βελτίωση του χρόνου ανάπτυξης του λογισμικού.

Ένα υποπρόγραμμα είναι ένα σύνολο εντολών της Pascal που εκτελεί συγκεκριμένο έργο και γράφεται με τους ίδιους κανόνες που γράφεται ένα πρόγραμμα. Η Pascal μας δίνει τη δυνατότητα να έχουμε δύο είδη υποπρογραμμάτων: τις διαδικασίες (procedures) και τις συναρτήσεις (functions). Η δομή τους ελάχιστα διαφέρει, αλλά χρησιμοποιούνται σε διαφορετικές περιπτώσεις. Τα υποπρογράμματα δεν είναι αυτοτελή προγράμματα γι' αυτό και δεν μπορούν να εκτελεστούν από μόνα τους. Δηλώνονται μετά το τμήμα δηλώσεων του προγράμματος, μετά το VAR και πριν από το BEGIN του κυρίως προγράμματος.

Δομή Διαδικασιών

Programexample(input, output);

->Επικεφαλίδα Διαδικασίας

{…….} Σχόλια

Procedure message;

Begin

Write(‘this is a procedure’)

End;

{…….}

Begin

Write(‘this is the first program’);

End.

Δομή Συνάρτησης

Programexample(input, output);

->Επικεφαλίδα Συνάρτησης

{…….} Σχόλια

function message;

Begin

Write(‘this is a function’)

End;

{…….}

```
Begin

Write(‘this is a function);

End.
```

Τα υποπρογράμματα επικοινωνούν με το κυρίως πρόγραμμα μέσω παραμέτρων και περιλαμβάνουν ένα σύνολο εντολών οι οποίες εκτελούν μια συγκεκριμένη λειτουργία. Η διαφορά μεταξύ τους είναι ότι οι συναρτήσεις υπολογίζουν και επιστρέφουν στο σημείο το οποίο καλούνται μια τιμή, ενώ οι διαδικασίες εκτελούν κάποιες λειτουργίες μεταβάλλοντας κάποιες μεταβλητές χωρίς να επιστρέφουν τιμή στο σημείο στο οποίο καλούνται.

4.5. Πίνακες

Χρησιμοποιούμε πίνακες, όταν θέλουμε να επεξεργαστούμε ένα πλήθος ομοειδών στοιχείων. Τα στοιχεία ενός πίνακα είναι πάντα του ίδιου τύπου, π. χ. ακέραιοι αριθμοί, πραγματικοί αριθμοί, συμβολοσειρές κλπ. , και αναφέρονται με κοινό όνομα (το όνομα του πίνακα) και τη θέση τους στο πίνακα. Οι διαστάσεις του πίνακα στην Pascal πρέπει να είναι γνωστές εκ των προτέρων (να δηλώνονται δηλαδή στο τμήμα δηλώσεων του προγράμματος). Τα στοιχεία του πίνακα αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη.

Μια μεταβλητή πίνακα δηλώνεται ως εξής:

VAR <όνομα πίνακα>: ARRAY[<διαστάσεις>] OF <τύπος στοιχείων>;

Συγκεκριμένα για το μονοδιάστατο πίνακα Α με 10 στοιχεία:

VAR A: ARRAY[1. . 10] OF INTEGER;

Το όνομα του πίνακα είναι Α (επιλέγεται από τον προγραμματιστή), τα στοιχεία του πίνακα είναι ακέραιοι αριθμοί και καθένα από αυτά αναφέρεται ως A[1], A[2], A[3], . . . , A[10]

Για πίνακα δύο διαστάσεων Α με 3 γραμμές και 4 στήλες:

VAR A: ARRAY[1. . 3, 1. . 4] OF REAL;

Το όνομα του πίνακα είναι Α (επιλέγεται από τον προγραμματιστή), τα στοιχεία του πίνακα είναι πραγματικοί αριθμοί και καθένα από αυτά αναφέρεται ως A[I, J], όπου το I έχει τιμές 1, 2, ή 3 και το J έχει τιμές 1, 2, 3 ή 4.

A [1, 1] A [1, 2] A[1, 3] A[1, 4]

A [2, 1] A [2, 2] A [2, 3] A [2, 4]

A [3, 1] A [3, 2] A [3, 3] A [3, 4]

Βασικές λειτουργίες που μπορεί να γίνουν στα στοιχεία ενός πίνακα είναι:

- Ανάγνωση των στοιχείων του πίνακα από το πληκτρολόγιο ή απόδοση τιμής στα στοιχεία με κάποιο τύπο.
- Εκτύπωση των στοιχείων του πίνακα ή εμφάνισή τους στην οθόνη.
- Προσπέλαση του πίνακα για αναζήτηση των στοιχείων που ικανοποιούν κάποια συνθήκη.
- Επεξεργασία των στοιχείων του πίνακα (για εύρεση αθροίσματος, μέσου όρου, μέγιστου και ελάχιστου στοιχείου, κλπ.).
- Ταξινόμηση των στοιχείων του πίνακα, με στόχο την ευκολότερη και ταχύτερη αναζήτηση.

Επειδή οι λειτουργίες αυτές εκτελούνται σε όλα τα στοιχεία του πίνακα και δεδομένου ότι οι διαστάσεις του πίνακα είναι γνωστές (άρα και ο αριθμός των στοιχείων) για την επεξεργασία του πίνακα ενδείκνυται η χρήση της εντολής FOR.

Π. χ. για να διαβάσουμε το μονοδιάστατο πίνακα Α με 10 στοιχεία που ορίσαμε πιο πάνω χρησιμοποιούμε:

```
FOR I: =1 TO 10 DO
   READLN (A [I]);
```

Αν θέλουμε να είναι πιο ολοκληρωμένο και να ξέρουμε πιο στοιχείο του πίνακα διαβάζουμε κάθε φορά χρησιμοποιούμε:

```
FOR I: =1 TO 10 DO
BEGIN
   WRITE ('DOSE STOIXEIO A[ ', I, ' ]');
   READLN (A[I]);
END;
```

Για να διαβάσουμε το πίνακα δύο διαστάσεων Α με 3 γραμμές και 4 στήλες που ορίσαμε πιο πάνω χρησιμοποιούμε:

```
FOR I: =1 TO 3 DO
  FOR J:=1 TO 4 DO
     READLN (A [I,  J]);
```

Αν θέλουμε να είναι πιο ολοκληρωμένο και να ξέρουμε πιο στοιχείο του πίνακα διαβάζουμε κάθε φορά χρησιμοποιούμε:

```
FOR I: =1 TO 3 DO
  FOR J:=1 TO 4 DO
  BEGIN
     WRITE ('DOSE STOIXEIO A [', I, ',  ', J, ']');
     READLN (A [I, J]);
  END;
```

4.6. Εγγραφές

Η εγγραφή (record) είναι ένα σύνολο στοιχείων, τα οποία δεν είναι (υποχρεωτικά) του ίδιου τύπου μεταξύ τους. Τα στοιχεία του record λέγονται πεδία (fields).

Η δήλωση ενός record είναι η ακόλουθη:

TYPE <ΟΝΟΜΑ ΤΥΠΟΥ> = RECORD

<ΟΝΟΜΑ ΠΕΔΙΟΥ 1>: <ΤΥΠΟΣ ΠΕΔΙΟΥ 1>;

<ΟΝΟΜΑ ΠΕΔΙΟΥ 2>: <ΤΥΠΟΣ ΠΕΔΙΟΥ 2>;

. . .

<ΟΝΟΜΑ ΠΕΔΙΟΥ N>: <ΤΥΠΟΣ ΠΕΔΙΟΥ N>;

END;

Παράδειγμα:

Ας υποθέσουμε ότι θέλουμε να επεξεργαστούμε τα στοιχεία των σπουδαστών του Τμήματος Επιχειρηματικού Σχεδιασμού και Πληροφοριακών Συστημάτων. Τα στοιχεία που θέλουμε να έχουμε είναι: Επώνυμο, Όνομα, Αριθμός Μητρώου, Χρονολογία Γέννησης, Φύλο και Βαθμολογία ενός Μαθήματος.

SURNAME	NAME	AM	BIRTHDAY	SEX	GRADE
ΑΛΕΞΑΚΗ	ΥΠΑΤΙΑ	100	1984	ΚΟΡΙΤΣΙ	8, 5
ΑΙΓΙΝΗΤΟΥ	ΜΑΡΙΑ	124	1985	ΚΟΡΙΤΣΙ	9
ΓΟΥΡΓΟΥΡΙΝΗ	ΕΛΕΝΗ	156	1986	ΚΟΡΙΤΣΙ	9, 5
ΠΑΠΑΔΟΠΟΥΛΟΣ	ΣΤΑΜΑΤΗΣ	567	1983	ΑΓΟΡΙ	7, 5

```
TYPE STUDENT = RECORD
  SUR,  NAME: STRING [20];
  AM,  BIRTH: INTEGER;
  SEX: CHAR;
  GRADE: REAL;
END;
VAR STUD: STUDENT;
```

Στη παραπάνω δήλωση τύπου (type) ορίζουμε τον τύπο STUDENT με στοιχεία: το επώνυμο (SUR) και το όνομα (NAME) του σπουδαστή ως συμβολοσειρές μήκους 20 χαρακτήρων, τον αριθμό μητρώου (AM) και την χρονολογία γέννησης (BIRTH) ως ακέραιους αριθμούς, το φύλο (SEX) ως χαρακτήρα και τη βαθμολογία (GRADE) ως πραγματικό αριθμό. Στη VAR δηλώνουμε μια μεταβλητή STUD η οποία είναι τύπου STUDENT. Το όνομα STUDENT δεν μπορεί να χρησιμοποιηθεί στις εντολές του προγράμματος, αφού έχει δηλωθεί ως τύπος δεδομένων και όχι ως μεταβλητή.

Η αναφορά των πεδίων ενός RECORD γίνεται με τον τελεστή.

Η εντολή STUD. SUR := 'ΑΛΕΞΑΚΗ'; δίνει στο πεδίο SUR του RECORD STUD την τιμή ΑΛΕΞΑΚΗ.

Η εντολή STUD. NAME := 'ΥΠΑΤΙΑ'; δίνει στο πεδίο NAME του RECORD STUD την τιμή ΥΠΑΤΙΑ.

Η εντολή STUD. AM := 100; δίνει στο πεδίο AM του RECORD STUD την τιμή 100.

Η εντολή STUD. BIRTH := 1984; δίνει στο πεδίο BIRTH του RECORD STUD την τιμή 1984.

Η εντολή STUD. SEX := 'F'; δίνει στο πεδίο SEX του RECORD STUD την τιμή F (Female -> Κορίτσι).

Η εντολή STUD. GRADE := 8. 5; δίνει στο πεδίο GRADE του RECORD STUD την τιμή 8. 5.

Πεδία του RECORD μπορεί να είναι και πίνακες. Στο πιο πάνω παράδειγμα θα μπορούσαμε να είχαμε τους βαθμούς (GRADE) 5 μαθημάτων οπότε η δήλωση του RECORD θα γινόταν ως εξής:

```
TYPE STUDENT = RECORD
  SUR,  NAME: STRING [20];
  AM,  BIRTH: INTEGER;
  SEX: CHAR;
  GRADE: ARRAY [1. . 5] OF REAL;
END;
VAR STUD: STUDENT;
```

Η εντολή STUD. GRADE[1] := 8, 5 δίνει στο πεδίο GRADE[1] του RECORD STUD την τιμή 8. 5. (Βαθμό για το 1ο μάθημα).

Επίσης στοιχεία ενός πίνακα μπορεί να είναι RECORDS. Στο πρώτο παράδειγμα που χρησιμοποιήσαμε θα μπορούσαμε να είχαμε τα στοιχεία για 50 μαθητές οπότε η δήλωση του RECORD θα γινόταν ως εξής:

```
TYPE STUDENT = RECORD
  SUR,  NAME: STRING [20];
  AM,  BIRTH :INTEGER;
  SEX: CHAR;
  GRADE: REAL;
END;
VAR STUD: ARRAY[1. . 50] OF STUDENT;
```

Η εντολή STUD[1]. SUR := 'ΑΛΕΞΑΚΗ'; δίνει στο πεδίο SUR του RECORD STUD[1] την τιμή ΑΛΕΞΑΚΗ. (Για το 1ο Σπουδαστή)

Η εντολή STUD[1]. NAME := 'ΥΠΑΤΙΑ'; δίνει στο πεδίο NAME του RECORD STUD[1] την τιμή ΥΠΑΤΙΑ. (Για το 1ο Σπουδαστή)

Η εντολή STUD[1]. AM := 100; δίνει στο πεδίο AM του RECORD STUD[1] την τιμή 100. (Για το 1ο Σπουδαστή)

Η εντολή STUD. BIRTH := 1984; δίνει στο πεδίο BIRTH του RECORD STUD[1] την τιμή 1984. (Για το 1ο Σπουδαστή)

Η εντολή STUD[1]. SEX := 'F'; δίνει στο πεδίο SEX του RECORD STUD[1] την τιμή F (Female -> κορίτσι). (Για το 1ο Σπουδαστή)

Η εντολή STUD[1]. GRADE := 8. 5; δίνει στο πεδίο GRADE του RECORD STUD[1] την τιμή 8. 5. (Για το 1ο Σπουδαστή)

Επίσης θα μπορούσαμε να είχαμε συνδυασμό ή συνδυασμούς των πιο πάνω περιπτώσεων. π. χ. να είχαμε τα στοιχεία για 50 μαθητές και τους βαθμούς (GRADE) 5 μαθημάτων οπότε η δήλωση του RECORD θα γινόταν ως εξής:

```
TYPE STUDENT = RECORD
  SUR,  NAME: STRING [20];
  AM,  BIRTH: INTEGER;
  SEX: CHAR;
  GRADE: ARRAY [1. . . 5] OF REAL;
END;
VAR STUD: ARRAY [1. . . 50] OF STUDENT;
```

Η εντολή STUD[1]. GRADE[1] := 8. 5; δίνει στο πεδίο GRADE[1] του RECORD STUD[1] την τιμή 8. 5. (Το πρώτο μάθημα για το 1ο Σπουδαστή)

4.7. Οι Συμβολοσειρές στην Pascal

Μια συμβολοσειρά (string) είναι μια ακολουθία μηδέν ή περισσοτέρων χαρακτήρων ASCII που περιέχονται σε μια γραμμή προγράμματος και περικλείονται με αποστρόφους. Μια συμβολοσειρά χωρίς καθόλου χαρακτήρες ανάμεσα στις αποστρόφους καλείται μηδενική συμβολοσειρά, τότε πρέπει να την γράψετε δύο φορές. Αν μέσα σε μια συμβολοσειρά υπάρχει ένας χαρακτήρας ελέγχου, τότε αυτός πρέπει να τοποθετηθεί έξω από αποστρόφους. Αν ο χαρακτήρας ελέγχου βρίσκεται στο μέσο, μιας συμβολοσειράς, τότε η συμβολοσειρά πρέπει να κλειστεί προσωρινά και μετά να ξανανοίξει.
Αλυσίδα χαρακτήρων (string) είναι μια ακολουθία χαρακτήρων μεταξύ αποστρόφων.

Π.χ. ’algorithm’ , ‘Pascal Vers’, ‘5’, ‘236’, ‘5+2’

Οι αλυσίδες χαρακτήρων (*character strings*) επιτρέπουν στην Pascal την εύκολη χρήση κειμένου. Ορίζονται ως πίνακες τύπου packed array [0..N] of char

Παράδειγμα:

```
var
  myname : packed array [0..32] of char;
```

Με μεταβλητές που έχουν οριστεί ως αλυσίδες χαρακτήρων μπορούμε:

- να διαβάσουμε από το χρήστη και να τυπώσουμε τα περιεχόμενά τους με τις εντολές ReadLn και WriteLn.
- να αναθέσουμε μια μεταβλητή σε μια άλλη με χρήση του :=
- να συγκρίνουμε δύο μεταβλητές μεταξύ τους με τους τελεστές =, <, >, <=, >=. Το αποτέλεσμα της σύγκρισης εξαρτάται από την απόλυτη αλφαβητική σειρά των περιεχομένων των δύο μεταβλητών σύμφωνα με τον πίνακα χαρακτήρων του συστήματος.

Παράδειγμα:

```
program test;
var
   a, b : packed array [0..10] of char;

begin
     readln(a);
     b := a;
     writeln(b);
     writeln(b = a)
end.
```

Πρόσβαση στους χαρακτήρες

Σε μεταβλητές που έχουν οριστεί ως αλυσίδες χαρακτήρων μπορούμε ακόμα να επεξεργαστούμε τα στοιχεία τους γράμμα-γράμμα όπως σε πίνακες μιας διάστασης.

Παράδειγμα:

```
program test;
var
   a : packed array [0..10] of char;
   i : integer;

begin
     readln(a);
     for i := 0 to 10 do
         write(a[i]);
     writeln
end.
```

4.8. Τα προγράμματα της Pascal

Η ολοκληρωμένη δομή ενός προγράμματος Pascal αποτελείται από τρία τμήματα:

- Την επικεφαλίδα του προγράμματος.
- Το τμήμα δηλώσεων (declaration part).
- Το τμήμα των προτάσεων (program body).

Η επικεφαλίδα είναι μια λέξη κλειδί, η οποία χαρακτηρίζει όλο το πρόγραμμα. Τα δύο βασικά standard ονόματα αρχείων Input και Output στη λίστα ονομάτων των εξωτερικών αρχείων, αναφέρονται στις εξ' ορισμού περιφερειακές μονάδες εισόδου και εξόδου δεδομένων (όπως τερματικό, εκτυπωτής κ. α)

Παράδειγμα

Θέλουμε να ονομάσουμε ένα νέο πρόγραμμα και του δίνουμε το όνομα myprogram. Αυτό θα γραφτεί ως εξής:

Myprogram(input, output);

Ορίζουμε έτσι το όνομα του προγράμματος το οποίο είναι μια και μοναδική πρόταση. Μπορεί να είναι μια οποιαδήποτε λέξη π. χ. Test ή program κ. ο. κ, η οποία μας προσδίδει το θέμα το οποίο εξετάζουμε. Στις περισσότερες υλοποιήσεις της Pascal, δεν απαιτείται η λίστα των ονομάτων των αρχείων και σε πολλές άλλες, όπως στην Turbo Pascal, ολόκληρη η επικεφαλίδα είναι προαιρετική.

Το τμήμα δηλώσεων περιέχει όλα τα στοιχεία που πρόκειται να χρησιμοποιηθούν στο πρόγραμμα. Αποτελείται από τις περιοχές των:

- επιγραφών (π. χ. LABEL επιγραφή 1;)
- των σταθερών (π. χ. CONST LIMIT = 50;)
- των τύπων (π. χ. TYPE MONTH = 1. . . 12 ;)

- των μεταβλητών (π. χ. VAR X: INTEGER ;) και
- των υποπρογραμμάτων -τους ορισμούς των συναρτήσεων (function) και των διαδικασιών (procedure).

Κάθε δήλωση είναι μοναδική και πρέπει να ορισθεί πριν χρησιμοποιηθεί για πρώτη φορά.

Το τμήμα των προτάσεων περιέχει το σώμα του προγράμματος που αρχίζει με τη δεσμευμένη λέξη BEGIN και τελειώνει με τη δεσμευμένη λέξη END. , περικλείοντας ανάμεσά τους τις εκτελέσιμες εντολές του προγράμματος. Η Pascal είναι σειριακή γλώσσα προγραμματισμού. Αυτό σημαίνει ότι οι προτάσεις του προγράμματος εκτελούνται σειριακά η μία μετά την άλλη. Στην Pascal δεν υπάρχει ο περιορισμός μιας εντολής σε κάθε γραμμή. Μπορούμε να γράψουμε μια εντολή σε περισσότερες γραμμές ή και πολλές εντολές σε μια γραμμή. Για να διαχωρίζονται οι εντολές μεταξύ τους χρησιμοποιείται το σύμβολο του ελληνικού ερωτηματικού (;), το οποίο τοποθετούμε στο τέλος κάθε εντολής. Οι δεσμευμένες λέξεις BEGIN και END χρησιμοποιούνται επίσης ως διαχωριστές εντολών, για αυτό δεν είναι απαραίτητο το ερωτηματικό μετά από το BEGIN και πριν από το END. Στο τέλος του προγράμματος, μετά το END βάζουμε τελεία για να δηλώσουμε το τέλος του προγράμματος.

Ένα πρόγραμμα Pascal έχει την ακόλουθη γενική μορφή:

PROGRAM <όνομα προγράμματος>;

LABEL<επιγραφή1, επιγραφή2…>;

CONST<τμήμα δήλωσης σταθερών του προγράμματος>;

TYPE<τμήμα δήλωσης τύπων του προγράμματος>;

VAR<τμήμα δήλωσης μεταβλητών του προγράμματος>;

<τμήμα δήλωσης συναρτήσεων και διαδικασιών>;

BEGIN {έναρξη σώματος κυρίως προγράμματος}

<εντολές κυρίως προγράμματος>;

END.

4.9. Αλγόριθμοι εύρεσης συμβολοσειράς σε κείμενο

4.9.1. Αλγόριθμος Brute Force

Η σπουδαιότερη λειτουργία στις συμβολοσειρές είναι η λειτουργία κατά την οποία αναζητάμε μια λέξη σε μια πρόταση, το λεγόμενο ταίριασμα ή ταύτιση προτύπου. Ένας απλός τρόπος αναζήτησης μιας λέξης σε ένα κείμενο είναι ο διαδοχικός έλεγχος κάθε χαρακτήρα του κειμένου. Ελέγχεται αν είναι ίδιος με τον πρώτο χαρακτήρα του κειμένου,. Αν ναι, προχωρά και συγκρίνει τον δεύτερο έως ότου βρει όλους τους χαρακτήρες. Η μέθοδος αυτή καλείται απλοϊκή (naive) ή ωμή βία (Brute force)

Υλοποιείται από τον παρακάτω αλγόριθμο:

```
Program BruteForce(input, output);

var i, j, k, l, m, n :integer;

   text: array[1. . 17] of char;

   pat: array[1. . 4] of char;

   x, y:char;

   search: integer;

   label telos;

   begin

   text:='I want to go home';

   pat:='home';

        search:=0;

        m:=4;

         if m=0 then begin
```

```
 search:=1; goto telos; end;
 n:=17;
 j:=1; i:=1;
 while (i<=n) do begin
      if text[i] <> pat[j] then begin
        i:=i-j+1; j:=1; end
      else begin
        j:=j+1;
        if j>m then begin
          search:=i-j+2;
          goto telos;
        end
      end;
      i:=i+1;
      end;
    telos:
if search<>0 then begin
 writeln('Pattern found in location ', search);
end
else begin
 writeln('Pattern not found. ');
end;
```

```
    readln;
end.
```

Στον αλγόριθμο αυτό έχουμε 2 μονοδιάστατους πίνακες. Τον πίνακα text που περιέχει 17 χαρακτήρες και στοιχεία του αποτελούν οι λέξεις της πρότασης ‘I want to go home’ που αποτελεί και το κείμενο που έχουμε ως πρότυπο. Δεύτερος πίνακας είναι ο pat, που αποτελείται από 4 χαρακτήρες-όσοι και οι χαρακτήρες της λέξης που ψάχνουμε. -

Στο παρακάνω παράδειγμα αναζητούμε τη λέξη ‘home’ στη πρόταση ‘ I want to go home’.

```
program BruteForce(input, output);
```

Επικεφαλίδα προγράμματος. Δηλώνουμε το όνομα του προγράμματος.

```
var i, j, k, l, m, n :integer; ->Περιοχή μεταβλητών
```

```
text: array[1. . 17] of char; -> Περιοχή Τύπων. Δήλωση του πίνακα text, που αποτελείται από 17 χαρακτήρες.
```

```
pat: array[1. . 4] of char; -> Περιοχή Τύπων. Δήλωση του πίνακα pat, που αποτελείται από 4 χαρακτήρες.
```

```
x, y:char; ->Περιοχή μεταβλητών
```

```
search: integer; ->Περιοχή μεταβλητών
```

```
label telos; -> Περιοχή επιγραφών
```

Ακολουθεί έπειτα το τμήμα προτάσεων που περιέχει τις εκτελέσιμες προτάσεις του προγράμματος. Αρχίζει με την εντολή begin και τελειώνει με την εντολή end.

Σε αυτό το σημείο δηλώνουμε την πρόταση που έχουμε ορίσει ως κείμενο: ‘I want to go home’ και από κάτω την λέξη την οποία ψάχνουμε. ‘home’.

```
begin
text:='I want to go home';
```

pat:='home';

Ακολουθούν οι εντολές σύμφωνα με τις οποίες ψάχνουμε να βρούμε τη λέξη που θέλουμε.

Οι δείκτες i και j χρησιμοποιούνται για τη σάρωση του κειμένου και του προτύπου.

```
search:=0;
m:=4;
 if m=0 then begin
  search:=1; goto telos; end;
 n:=17;
 j:=1; i:=1;
```

Όταν ο πρώτος χαρακτήρας του προτύπου ταυτίζεται με εκείνου του κειμένου, τότε και οι δυο δείκτες αυξάνονται. Αν κάποια στιγμή παρατηρηθεί μη ταύτιση, τότε ο δείκτης i παίρνει την επόμενη τιμή από αυτή που είχε ξεκινήσει-δηλαδή προχωρά στο επόμενο γράμμα για να ψάξει- και ο δείκτης j παίρνει τη τιμή 1. Ο αλγόριθμος τερματίζει όταν ο δείκτης j γίνει μεγαλύτερος του m που σημαίνει ότι βρέθηκε η λέξη που ψάχναμε.

Ο δείκτης i αυξάνεται μέχρι τη θέση 14 όπου και συναντά το πρώτο γράμμα της λέξης που ψάχνει.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I		w	a	n	t		t	o		g	o		h	o	m	e

```
 while (i<=n) do begin
     if text[i] <> pat[j] then begin
       i:=i-j+1; j:=1; end
```

Έως ότου να φθάσει στη θέση 14 ο i αυξανόταν συνεχώς κατά 1 ενώ ο j έπαιρνε συνεχώς τη τιμή 1 λόγω του ότι δε συναντούσε κάποιο γράμμα της λέξης που αναζητούμε. Όταν ο i παίρνει τη τιμή 14 ο j παίρνει τη τιμή 2. Έπειτα ο i παίρνει τις τιμές 15, 16, 17 και ο j αντίστοιχα τις τιμές 3, 4, 5. Όταν ο j πάρει τη τιμή 5 η εφαρμογή τερματίζεται διότι j>m.

```
        else begin
          j:=j+1;
          if j>m then begin
            search:=i-j+2;
            goto telos;
          end
        end;
        i:=i+1;
        end;
      telos:
```

```
    if search<>0 then begin
```

Με την εντολή writeln, ζητάμε να μας γράψει εφόσον έχει ψάξει όλη την πρόταση, αν βρήκε ή όχι τη λέξη που του ζητήσαμε. Αν βρέθηκε, εφόσον τρέξουμε το πρόγραμμα. Θα μας πει

'Pattern found in location…,

Ενώ αν του ζητήσαμε να βρει μια λέξη που δεν υπάρχει πχ ‘ball’ θα μας πει 'Pattern not found.

```
      writeln('Pattern found in location ', search);
```

```
end

else begin

 writeln('Pattern not found. ');

end;
```

Τέλος με την εντολή Readln ζητάμε από το πρόγραμμα να διαβάσει όλα τα παραπάνω. Σε περίπτωση που την παραλείψουμε το πρόγραμμα θα τρέξει απλά δε θα μας εμφανιστεί το αποτέλεσμα.

```
readln;
```

Σημαντικό είναι να τερματίσουμε το πρόγραμμα με την εντολή end και να ακολουθήσει η τελεία «. » για να δηλώσουμε, ότι το πρόγραμμα τερματίζει εδώ.

```
end.
```

Αν είχαμε ένα διαφορετικό παράδειγμα, όπως πχ να αναζητούμε τη λέξη ‘domes’ στο κείμενο:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 171819 20 21 22 23 24

t	a		d	e	d	o	m	e	n	a		k	a	i		o	i		d	o	m	e	s

το i αυξάνεται μέχρι το 4, όπου διαπιστώνεται ταύτιση με το πρώτο γράμμα της λέξης που ψάχνουμε. Στη συνέχεια το i γίνεται 5 και το j 2. Αλλά ο χαρακτήρας στη θέση 5 δε ταιριάζει με αυτόν που ψάχνουμε, οπότε το i παραμένει 5 και ο j γίνεται πάλι 1. Στις θέσεις 6 έως 9 υπάρχει ταύτιση οπότε το i αυξάνεται ως το 9 και το j είναι 5. Επειδή όμως στη θέση 11 δεν υπάρχει ταύτιση το j ξαναγίνεται 1 και το i γίνεται 7. Στις θέσεις 20-24 υπάρχει ταύτιση και το i αυξάνεται στο 24 και το j είναι 6. Η αναζήτηση θα τελειώσει αφού το j έχει πάρει τιμή μεγαλύτερη του m

Σημείωση

Σε αυτή την περίπτωση θα τρέξω τον παραπάνω αλγόριθμο με τη διαφορά ότι θα αλλάξω τα στοιχεία του πίνακα, δηλαδή Text από [1. . 17] σε [1. . 24] όσα είναι και τα γράμματα του κειμένου και θα αλλάξω και τα στοιχεία του δεύτερου πίνακα που περιέχει τη λέξη που αναζητούμε, δηλαδή από pat[1. . 4] σε pat[1. . 5] .

4.9.2. Αλγόριθμος KMP

Οι Knuth, Pratt και Morris επινόησαν τον αλγόριθμο KMP, σύμφωνα με τον οποίο όταν διαπιστωθεί μια μη ταύτιση, η αναζήτηση δε συνεχίζεται από τον επόμενο χαρακτήρα, αλλά από τον τρέχοντα. Η διαφορά από τον προηγούμενο αλγόριθμο είναι ότι ο δείκτης i δεν οπισθοχωρεί ποτέ.

Υλοποιείται από τον παρακάτω αλγόριθμο:

```
program kmp(input, output);

var i, j, k, l, m, n :integer;

    text: array[1. . 17] of char;

    pat: array[1. . 4] of char;

    x, y:char;

    search: integer;

    next: array[1. . 17] of integer;

    found: boolean;

    procedure check_pattern;

    var k, l: integer;

    begin

     m:=4;

     l:=1;

     k:=0; next[1]:=0;

     repeat begin

      if (k=0) or (pat[l]=pat[k]) then begin
```

```
      l:=l+1; k:=k+1;
      if pat[k]=pat[l] then next[l]:=next[k]
      else                 next[l]:=k;
      end
  else k:=next[k];
  end
 until (l>m);
end;
begin
text:='I want to go home';
pat:='want';
found:=FALSE; search:=0;
m:=4;
if m=0 then begin
search:=1; found:=TRUE; end;
check_pattern;
n:=17;
j:=1; i:=1;
while not found and (i<=n) do begin
      if (j=0) or (pat[j]=text[i]) then begin
         i:=i+1; j:=j+1;
         if j>m then begin
```

```
            search:=i-j+1;
            found:=TRUE;
            end;
          end
       else j:=next[j];
  end;
  if search<>0 then begin
   writeln('Pattern found in location ', search);
  end
  else begin
   writeln('Pattern not found. ');
  end;
  readln;
  End.
```

4.9.3. Αλγόριθμος Not So Naive

Ο Not So Naive αλγόριθμος εκτελεί αναζήτηση αλφαριθμητικού – προτύπου (pattern) σε κείμενο (text) και για τούτο εφαρμόζει το μηχανισμό του κυλιόμενου παράθυρου. Ο μηχανισμός δουλεύει ως εξής: το κείμενο σαρώνεται με τη βοήθεια ενός παραθύρου του οποίου το μέγεθος είναι m, όσο και το μέγεθος του προτύπου. Η αριστερή άκρη αυτού του παραθύρου καθώς και του κειμένου στοιχίζονται και έπειτα συγκρίνονται οι χαρακτήρες του κειμένου που βρίσκονται μέσα στο παράθυρο με τους χαρακτήρες του προτύπου. Μετά από επιτυχημένο ταίριασμα του προτύπου ή μιας αποτυχίας ταιριάσματος το παράθυρο κυλά προς τα δεξιά. Η ίδια διαδικασία επαναλαμβάνεται ωσότου η δεξιά άκρη του παραθύρου ξεπεράσει τη δεξιά άκρη του κειμένου.

```
program Not_So_Naive;
```

x είναι ο πίνακας που κρατά το string του προτύπου (pattern) που έχει μήκος m, ενώ y είναι ο πίνακας που κρατά το string του κειμένου (text) που έχει μήκος n. Τα αλφαριθμητικά διαβάζονται με την εντολή readln και το μήκος τους υπολογίζεται από την εντολή length(string). j είναι η θέση του κειμένου από την οποία ξεκινά το κυλιόμενο παράθυρο ώστε να αρχίσει η σύγκριση των γραμμάτων.

```
  var m, n, j, k, ell : integer;
      index : byte;
      x, a: String[50];
      y, b:String[255];
  begin
  Writeln('Not So Naive Algorithm');
  Write('Enter the pattern: ');
  Readln(x);
  Write('Enter the text: ');
  Readln(y);
  m:=length(x);
  n:=length(y);
```

Στη φάση της προ-επεξεργασίας που ακολουθεί, ο αλγόριθμος διαχωρίζει δύο περιπτώσεις για το string του προτύπου (pattern): μία στην οποία τα πρώτα δύο γράμματα είναι ίδια και μία στην οποία αυτό δεν συμβαίνει. Αυτός ο διαχωρισμός γίνεται με τον έλεγχο x[1] = x[2] και σε περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το παράθυρο εντοπισμού του προτύπου θα κυλά κατά δύο θέσεις δεξιά κάθε φορά, το οποίο οφείλεται στην εκχώρηση της τιμής 2 στη μεταβλητή k.

Αυτό το επιθυμούμε, διότι αν οι χαρακτήρες x[2] και y[j+1] είναι ανόμοιοι, σίγουρα θα είναι επίσης ανόμοιοι οι x[1] και y[j+1]. Η δεύτερη σύγκριση δεν θα γίνει όμως λόγω της τιμής 2 που έχει εκχωρηθεί στη μεταβλητή ell και όπως θα δούμε παρακάτω, ο δείκτης j στον πίνακα χαρακτήρων που κρατά το κείμενο θα μετατοπίζεται κατά 2 θέσεις και όχι 1.

```
if x[1] = x[2] then begin
   k := 2;
   ell := 1;
end
else begin
   k := 1;
   ell := 2;
end;
```

Στη φάση της αναζήτησης του προτύπου μέσα στο string, ο έλεγχος j<=n-m+1 εξασφαλίζει ότι όταν το κείμενο θα φτάνει στο τέλος του, η σύγκριση των χαρακτήρων του προτύπου και του κειμένου θα συμβαίνει μόνο μέχρις ότου αυτά να έχουν στοιχιστεί ως προς το τέλος τους, χωρίς το πρότυπο να έχει μεγαλύτερο μήκος από τους χαρακτήρες που απομένουν στο κείμενο. Έτσι, εξασφαλίζεται ότι το παράθυρο θα είναι αρκετά μεγάλο ώστε να χωρά το πρότυπο και όταν αυτό δεν θα συμβαίνει, ο βρόχος επανάληψης τερματίζεται. Ο έλεγχος x[2] <> y[j+1] βγαίνει αληθής όταν τα αντίστοιχα στοιχεία του προτύπου και του κειμένου δεν ταυτίζονται. Σε αυτή την περίπτωση εκτελούνται οι εντολές του τμήματος if, τουτέστιν ο δείκτης j αυξάνεται κατά k θέσεις. Όπως προαναφέραμε, αν οι 2 πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το k είναι 2, ώστε να αποφευχθεί αργότερα ο σίγουρα ανεπιτυχής έλεγχος του x[1] με το y[j+1].

```
j:=1;
```

```
  while j<=n-m+1 do begin
    if x[2] <> y[j+1] then j:=j+k
```

Αν ο έλεγχος βγει ψευδής, εκτελούνται οι εντολές του τμήματος else. Η εντολή copy την πρώτη φορά παίρνει το απόσπασμα του string x που ξεκινά από τη θέση 3 και έχει μήκος m-2 και το αντιγράφει στο string a, ενώ τη δεύτερη φορά παίρνει το απόσπασμα του string y που ξεκινά από τη θέση j+2 και έχει μήκος m-2 και το αντιγράφει στο string b.

```
    else begin
      a:=copy(x, 3, m-2);
      b:=copy(y, j+2, m-2);
```

Συγκρίνονται τα a και b αν είναι ίδια (όχι ίσα, αφού δεν είναι αριθμοί αλλά αλφαριθμητικά) και για να θεωρηθεί ότι βρέθηκε το πρότυπο, πρέπει επίσης οι πρώτοι χαρακτήρες του προτύπου και του παραθύρου να είναι ίδιοι. Αν αυτές οι δύο προϋποθέσεις πληρούνται, έχουμε μία επιτυχημένη απόπειρα εύρεσης του προτύπου μέσα στο κείμενο και σαν αποτέλεσμα, εμφανίζεται αντίστοιχο μήνυμα στο χρήστη. Τέλος, η μεταβλητή j αυξάνεται κατά ell, ώστε να συνεχιστεί η αναζήτηση στο κείμενο, διότι υπάρχει πιθανότητα το πρότυπο να εμφανίζεται περισσότερες από μία φορές μέσα στο κείμενο.

```
      if ( (a=b) and (x[1]=y[j]) ) then
        writeln('Pattern found at position ', j);
      j:=j+ell
    end;
  end;
end.
```

4.9.4. Αλγόριθμος Quick Search

Ο αλγόριθμος Quick Search αναζητά ένα αλφαριθμητικό πρότυπο (pattern) μέσα σε κείμενο (text), χρησιμοποιώντας bad character shift, ενώ η σειρά με την οποία εκτελούνται οι συγκρίσεις των χαρακτήρων των αλφαριθμητικών είναι αδιάφορη.

```
program Quick_Search;
```

p είναι το string με το πρότυπο, ενώ t είναι το string με το κείμενο. plen και tlen τα μήκη των αντίστοιχων αλφαριθμητικών

```
  var plen, tlen : integer;
      index : byte;
      p, a: String[50];
      t, b:String[255];
```

Η συνάρτηση QS επιτελεί το ρόλο της αναζήτησης του προτύπου στο κείμενο και δηλώνεται προτού χρησιμοποιηθεί.

```
  procedure QS(x:String;m:integer;y:String;n:integer);
  var i, j:integer;
      qsBc:array[1. . 255] of integer;
  begin
```

Κατά τη φάση της προεπεξεργασίας, ο πίνακας qsBc γεμίζει ως εξής: αν ένα γράμμα του αλφαβήτου δεν υπάρχει στο πρότυπο, αντιστοιχίζεται σε αυτό τιμή για shift η m+1.

```
      for i:=1 to 254 do qsBc[i]:=m+1;
```

Όταν ένα γράμμα του αλφαβήτου υπάρχει στο πρότυπο, αντιστοιχίζεται η μικρότερη τιμή του i για την οποία ισχύει ότι x[m-1-i]=c και 0<=i<=m-1 όπου x το πρότυπο και c το γράμμα. Ουσιαστικά, βρίσκεται η δεξιότερη εμφάνιση κάθε

χαρακτήρα x[i] του προτύπου, πχ αν x[i]='a', επειδή ο κωδικός του σε ASCII είναι 97, το qsBc[x[i]] ισοδυναμεί με qsBc[97]. Και αφού η θέση του χαρακτήρα στο πρότυπο είναι I, στη θέση x[i] του qsBc μπαίνει η τιμή m-i. . Η συνάρτηση ord (ordinal) επιστρέφει την ASCII τιμή ενός χαρακτήρα

```
for i:=1 to m do begin
qsBc[Ord(x[i])]:=m-i+1;
end;
```

Ο δείκτης j που δείχνει στο κείμενο αρχικά είναι 1 και ακολουθεί η φάση της αναζήτησης, δηλαδή συγκρίσεις του προτύπου με το κείμενο. Ο βρόγχος while επαναλαμβάνεται τόσες φορές όσες το μέγεθος του κυλιόμενου παραθύρου είναι μεγάλο σε μήκος όσο και το πρότυπο.

```
j:=1;
while j<=(n-m+1) do begin
```

Συγκρίνονται τα αλφαριθμητικά και αν ταυτιστούν εμφανίζεται ανάλογο μήνυμα στο χρήστη.

```
a:=copy(x, 1, m);
b:=copy(y, j, m);
if (a=b) then Writeln('Pattern found at position ', j);
```

Το j μετακινείται δεξιά τόσο όσο η τιμή στη θέση y[j+m] στον πίνακα qsBc, πχ αν ο χαρακτήρας υπάρχει μία φορά στην πρώτη θέση στο πρότυπο, θα γίνουν m μετατοπίσεις ενώ, όταν ο χαρακτήρας y[j+m] δεν υπάρχει στο πρότυπο, γίνονται m+1 μετατοπίσεις

```
j:=j+qsBc[Ord(y[j+m])];
end;
end;
```

```
  begin
  Writeln('Not So Naive Algorithm');
  Write('Enter the pattern: ');
  Readln(p);
  Write('Enter the text: ');
  Readln(t);
  plen:=length(p);
  tlen:=length(t);
```

Καλείται η συνάρτηση QS, η οποία επιτελεί την αναζήτηση του προτύπου μέσα στο κείμενο.

```
  QS(p, plen, t, tlen);
  readln(a);
end.
```

5. Η γλώσσα προγραμματισμού C++

Η γλώσσα προγραμματισμού C αναπτύχθηκε στην AT&T με σκοπό τη δημιουργία ενός λειτουργικού συστήματος για τη σειρά υπολογιστών PDP-11, που τελικά έγινε το λειτουργικό σύστημα Unix. Η C αναπτύχθηκε με πρωταρχικό σκοπό την αποδοτικότητα. Ο Bjarne Stroustrup, επίσης της AT&T, ανέπτυξε την C++ με πρωταρχικό σκοπό την προσθήκη αντικειμενοστραφών δομών στη γλώσσα. Επειδή η αντικειμενοστραφής τεχνολογία ήταν καινούργια και όλες οι αντικειμενοστραφείς υλοποιήσεις που υπήρχαν ήταν αρκετά αργές και μη αποδοτικές, ένας δευτερεύων σκοπός της C++ ήταν να διατηρήσει την αποδοτικότητα της C. Η C++ μπορεί να θεωρηθεί μια διαδικαστική γλώσσα με κάποιες επιπλέον δομές, μερικές από τις οποίες προστέθηκαν για αντικειμενοστραφή προγραμματισμό, ενώ άλλες για την βελτίωση του συντακτικού της γλώσσας. Ένα καλογραμμένο πρόγραμμα πρέπει να έχει στοιχεία τόσο αντικειμενοστραφή όσο και κλασικού διαδικαστικού προγραμματισμού. Η C++ είναι ουσιαστικά μια επεκτάσιμη γλώσσα αφού μπορούμε να ορίσουμε νέους τύπους με τέτοιο τρόπο ώστε να λειτουργούν σαν τους προκαθορισμένους τύπους, που είναι τμήμα της γλώσσας. Η C++ σχεδιάστηκε για την ανάπτυξη μεγάλων προγραμμάτων.

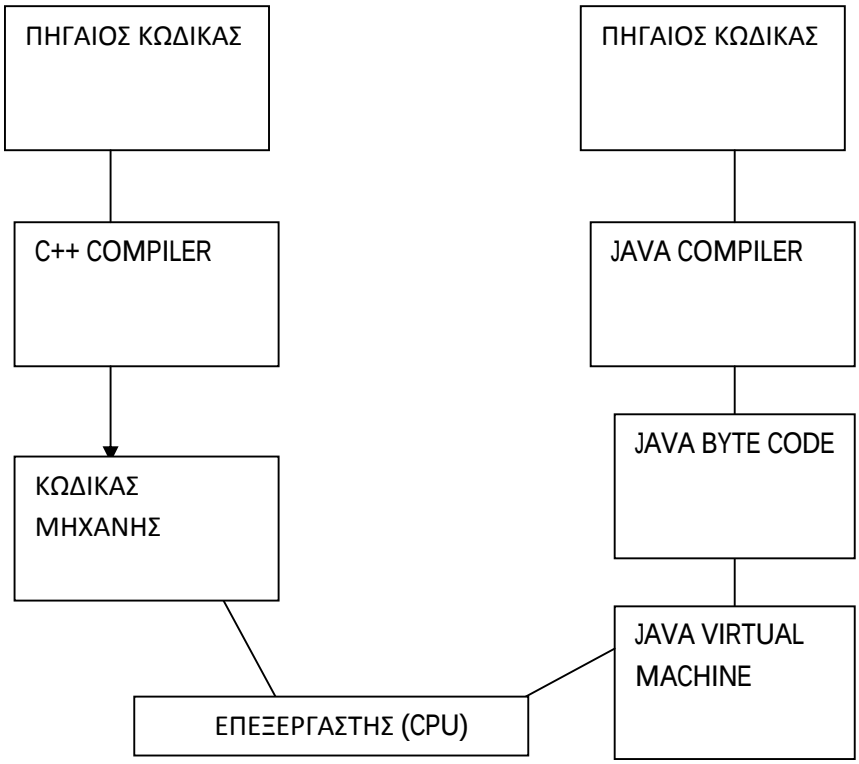
Η γλώσσα προγραμματισμού C ορίστηκε αρχικά στο κλασσικό σύγγραμμα των Kernigham και Ritchie "The C Programming Language", και ήταν το πρότυπο που χρησιμοποιούσαν όλοι οι προγραμματιστές στη C. Το πρότυπο ANSI για τη C τελικά εγκρίθηκε τον Δεκέμβριο του 1989 και έγινε το επίσημο πρότυπο για τον προγραμματισμό στη C. Το πρότυπο ANSI εισήγαγε αρκετά νέα στοιχεία, που δεν υπήρχαν στην αρχική έκδοση των Kernigham και Ritchie, και άλλαξε κάποια άλλα, έτσι ώστε τα δύο πρότυπα δεν είναι τελείως συμβατά. Η μελέτη αυτού του εγχειριδίου προϋποθέτει την γνώση της γλώσσας προγραμματισμού C, και εξηγεί εν συντομία τα βασικά στοιχεία της γλώσσας. Ωστόσο, επειδή πολλοί προγραμματιστές μπορεί να είναι εξοικειωμένοι με την ορολογία των Kernigham και Ritchie, κάποια τμήματα θα εξηγούν τις αλλαγές που έγιναν από την ANSI-C.

Ο καλύτερος τρόπος για να μάθει κανείς τη C++ είναι να την χρησιμοποιήσει. Ο πιο καλός τρόπος για να μάθει κανείς τη γλώσσα C++ είναι να μάθει τις νέες δομές χρησιμοποιώντας τις σε κάθε νέο πρόγραμμα. Θα ήταν λάθος να προσπαθήσει κανείς να χρησιμοποιήσει όλες τις νέες δομές της γλώσσας στο πρώτο του πρόγραμμα στη C++. Θα ήταν πιο σωστό σε κάθε πρόγραμμα να γίνεται η προσθήκη λίγων νέων στοιχείων ώστε να εξοικειωθείτε μαζί τους.

Μεταγλώττιση (compile) & Σύνδεση (link)

Μια συνηθισμένη γλώσσα προγραμματισμού (π. χ. C, C++, PASCAL) μεταγλωττίζει τον πηγαίο κώδικα του προγράμματος σε εκτελέσιμη μορφή που να καταλαβαίνει ο επεξεργαστής. Η μορφή αυτή είναι η γλώσσα μηχανής, και διαφέρει για κάθε επεξεργαστή, αρχιτεκτονική και λειτουργικό σύστημα. Η εκτελέσιμη μορφή ενός προγράμματος για μια συγκεκριμένη αρχιτεκτονική, δεν είναι δυνατό να χρησιμοποιηθεί σε διαφορετική αρχιτεκτονική, δηλαδή δε μπορούμε να χρησιμοποιήσουμε το εκτελέσιμο πρόγραμμα για Windows σε έναν υπολογιστή Macintosh. Έτσι, είναι απαραίτητη η εκ νέου μεταγλώττιση του πηγαίου κώδικα (source code) για κάθε αρχιτεκτονική στη οποία θέλουμε να τρέξουμε το πρόγραμμα.

Στο παρακάτω σχήμα απεικονίζεται η διαδικασία της μεταγλώττισης ενός προγράμματος


ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ
ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ
C++ COMPILER
JAVA COMPILER
ΚΩΔΙΚΑΣ
ΜΗΧΑΝΗΣ
JAVA BYTE CODE
JAVA VIRTUAL
MACHINE
ΕΠΕΞΕΡΓΑΣΤΗΣ (CPU)

5.1. Αντικειμενοστραφής προγραμματισμός (Object-Oriented Programming)

Οι αντικειμενοστραφείς γλώσσες προγραμματισμού (Java, και C++) δίνουν έμφαση στα δεδομένα παρά στον κώδικα. Το πρόγραμμα αναπτύσσεται γύρω από τα δεδομένα (data-centric) τα οποία ορίζουν από μόνα τους τον τρόπο με τον οποίο μπορούμε να τα διαχειριστούμε. Ο φυσικός και ο τεχνητός κόσμος που ζούμε είναι πιο κοντά στη φιλοσοφία του Αντικειμενοστραφή προγραμματισμού παρά του Δομημένου προγραμματισμού. Ένα απλό παράδειγμα που μπορούμε να χρησιμοποιήσουμε για την κατανόηση της φιλοσοφίας του αντικειμενοστραφούς προγραμματισμού είναι το αυτοκίνητο. Κάθε αυτοκίνητο είναι ένα αντικείμενο που ανήκει σε μια κλάση που ορίζει τα βασικά χαρακτηριστικά του αυτοκινήτου. Αυτά μπορεί να διαφέρουν ανάμεσα στους κατασκευαστές αλλά όλα θα παρέχουν τα βασικά χαρακτηριστικά που ορίζει η κλάση “αυτοκίνητο” για τη χρήση του. Δηλαδή τιμόνι, γκάζι, φρένο, συμπλέκτης και ταχύτητες. Αυτά είναι τα δεδομένα. Κάθε ένα από αυτά ορίζει τον τρόπο χρήσης του. Το τιμόνι στρίβει αριστερά/δεξιά, τα πεντάλ πιέζονται ή αφήνονται και οι ταχύτητες αλλάζουν διακριτά έχοντας μηχανισμό ασφαλείας – δε μπορούμε να αλλάξουμε ταχύτητα σε όπισθεν ενώ το αυτοκίνητο κινείται με ταχύτητα.

Η υλοποίηση καθενός από αυτούς τους μηχανισμούς διαφέρει σε κάθε κατασκευαστή, αλλά η χρήση τους είναι η ίδια για όλους. Δηλαδή η χρήση του τιμονιού και των ταχυτήτων γίνεται με τον ίδιο τρόπο ανεξαρτήτως κατηγορίας, κατασκευαστή και μοντέλου του αυτοκινήτου. Επίσης, ο μηχανισμός με τον οποίο γίνεται η χρήση των δεδομένων αυτών είναι κρυμμένος από τον χρήστη (δηλαδή τον οδηγό). Ο χρήστης δεν ενδιαφέρεται για τον τρόπο μετάδοσης της κίνησης από το τιμόνι στους τροχούς ώστε το αυτοκίνητο να στρίψει. Επίσης, η χρήση ενός αυτοκινήτου δεν αλλάζει με την επέκτασή του ή αλλαγή ορισμένων χαρακτηριστικών του (όπως π. χ. νέα μηχανή, λάστιχα, κλπ). Αλλάζει η συμπεριφορά του αλλά όχι η χρήση του. Με αυτόν τον τρόπο, περιγράψαμε με

ένα απλό παράδειγμα τα πιο σημαντικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού:

• Encapsulation ()

Η διαδικασίες κρύβονται από το χρήστη και τα ίδια τα δεδομένα προσδιορίζουν τους

τρόπους διαχείρισης τους.

• Polymorphism (πολυμορφισμός)

Αντικείμενα που ανήκουν σε παρόμοιες κλάσεις μπορούν να έχουν κοινό τρόπο προσπέλασης, με αποτέλεσμα ο χρήστης να μπορεί να τα χειριστεί με τον ίδιο τρόπο χωρίς να χρειάζεται να μάθει νέες διαδικασίες.

• Inheritance (κληρονομικότητα)

Μπορούμε να δημιουργήσουμε ένα νέο αντικείμενο παίρνοντας ως βάση ένα άλλο ήδη υπάρχον. Το νέο αντικείμενο θα έχει τα χαρακτηριστικά του παλιού ενώ θα μπορεί να τα τροποποιήσει, να τα επεκτείνει και να προσθέσει καινούρια για να καλύψει συγκεκριμένες ανάγκες. Όπως και κάθε γλώσσα προγραμματισμού, έτσι και η C++ ακολουθεί κάποιους κανόνες στη σύνταξη και υλοποίηση ενός προγράμματος. Οι κανόνες αυτοί αφορούν τη δήλωση μεταβλητών, τους τύπους δεδομένων, τους ορισμούς κλάσεων κλπ.

5.2. Αλφάβητο

Τελεστές (operators)

Μπορούμε να εκτελέσουμε διάφορες πράξεις μεταξύ των μεταβλητών με τη χρήση των τελεστών. Οι τελεστές είναι σύμβολα που αντιστοιχούν σε αριθμητικές ή λογικές πράξεις μεταξύ των αντικειμένων. Υπάρχουν 4 είδη τελεστών στη C++

Αριθμητικοί τελεστές

Σύμβολο	Είδος
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο Διαίρεσης
++	Αυξητικός τελεστής
--	Αφαιρετικός τελεστής

Ειδικά για τους αυξητικούς και αφαιρετικούς τελεστές, πρέπει να αναφέρουμε ότι αυτοί αυξάνουν ή αφαιρούν την τιμή της μεταβλητής κατά μία μονάδα (κατά συνέπεια χρησιμοποιούνται μόνο σε ακέραιες μεταβλητές, όχι σε δεκαδικές).

Σχεσιακοί Τελεστές

Οι σχεσιακοί τελεστές αναφέρονται στην σχέση μεταξύ δύο αντικειμένων.

Σύμβολο	Είδος
==	Ισότητα
!=	Ανισότητα

>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο από ή ίσο με
<=	Μικρότερο από ή ίσο με

Λογικοί Τελεστές

Οι λογικοί τελεστές αναφέρονται στους τρόπους συνδυασμών αληθών και ψευδών προτάσεων.

Σύμβολο	Είδος
&&	Short-circuit AND
||	Short-circuit OR
!	NOT (άρνηση)

Αν με τον υπολογισμό της πρώτης παράστασης ο συνδυασμός βγαίνει αληθής ή ψευδής τότε αποφεύγεται ο υπολογισμός της δεύτερης.

Για παράδειγμα, αν έχουμε την παράσταση

(alpha == true) && (beta == 1)

το πρόγραμμα θα επιστρέψει αμέσως ψευδή τιμή (false) αν η πρώτη είναι ψευδής, και θα προχωρήσει στον υπολογισμό της δεύτερης μόνο αν η πρώτη είναι αληθής. Στην πραγματικότητα είναι θέμα εξοικονόμησης χρόνου.

5.3. Λεξιλόγιο

Μεταβλητές - Σταθερές

Η έννοια της μεταβλητής είναι ίσως η πιο θεμελιώδης στον προγραμματισμό γενικά. Η μεταβλητή δεν είναι άλλο από μια ονομασμένη θέση μνήμης που μπορεί να περιέχει δεδομένα οποιουδήποτε είδους. Το περιεχόμενο της μεταβλητής μπορεί να μεταβάλλεται -εξ' ου και το όνομά της- με δεδομένα παρόμοιου είδους. Στην C++ μια μεταβλητή είναι μια θέση για αποθήκευση πληροφοριών πιο αναλυτικά, μια μεταβλητή μπορεί να θεωρηθεί μια θέση στην μνήμη του υπολογιστή μας, στην οποία μπορούμε να αποθηκεύσουμε μια τιμή και από την οποία θα μπορούμε αργότερα να ανακτήσουμε την τιμή αυτή.

Οι μεταβλητές ορίζονται ως εξής:

type var: όπου type το είδος της μεταβλητής και var το όνομά της. Ως είδος μπορούμε να διαλέξουμε έναν από τους διαθέσιμους τύπους δεδομένων της C++ ή το όνομα μιας κλάσης αντικειμένων (σε επόμενη παράγραφο).

Εύρος μεταβλητών

Μια μεταβλητή θεωρείται έγκυρη, δηλαδή μια αναφορά σε αυτήν έχει νόημα μόνο μέσα στο πλαίσιο στο οποίο ορίζεται. Το πλαίσιο μπορεί να είναι μια κλάση, μια συνάρτηση ή μέθοδος, το πεδίο ισχύος ενός loop for/while/do ή των εντολών if/else/switch.

Παράδειγμα:

```
type var1;

if (συνθήκη) {

    // Επεξεργασία της var1 (σωστό).

    // Επεξεργασία της var2 (λάθος).

type var2;
```

```
  // Επεξεργασία της var2 (σωστό)
}
   // Επεξεργασία της var1 (σωστό)
  // Επεξεργασία της var2 (λάθος)
```

Τύποι δεδομένων

Οι βασικοί τύποι δεδομένων στη C++ είναι σχεδόν οι ίδιοι με αυτούς της C.

Ακολουθεί πίνακας με τους διαθέσιμους τύπους δεδομένων της C++ και το μέγεθός τους σε bytes, για ένα συνηθισμένο PC (32-bit αρχιτεκτονική):

Όνομα	Μέγεθος (σε bytes)	Όρια
char	1	-128 έως 127
short	2	-32, 768 έως 32, 767
int/long	4	-2, 147, 483, 648 έως -2, 147, 483, 647.
long long	8	-9, 223, 372, 036, 854, 775, 808 έως -9, 223, 372, 036, 854, 775, 807
float	4	1. 4 * 10-45 έως 3. 4 * 1038
double	8	4. 9 * 10-324 έως 1. 8 * 10308
bool	1	true / false
wchar	2	-
string	ΜΕΤΑΒΛΗΤΟ	-

ΕΠΕΞΗΓΗΣΗ ΜΕΤΑΒΛΗΤΩΝ

Οι τύποι δεδομένων char, short, int, long, long long προορίζονται για χρήση με ακέραιους αριθμούς, ενώ οι τύποι float, double για χρήση αριθμών κινητής υποδιαστολής (δηλ. δεκαδικών αριθμών). Ο τύπος bool μπορεί να λαμβάνει μόνο δύο τιμές true και false. Ο τύπος wchar έχει μέγεθος 2 bytes γιατί έχει σχεδιαστεί να περιέχει χαρακτήρες Unicode (UTF-16 για την ακρίβεια). unsigned μεταβλητές

Στη C++ όπως και στη C, αλλά αντίθετα με τη Java, μπορούμε να διαμορφώσουμε το εύρος των μεταβλητών char, short, int, long ώστε να περιέχουν μόνο θετικούς αριθμούς. Στην πραγματικότητα ορίζουμε το bit του πρόσημο (sign) να χρησιμοποιείται και αυτό για αρίθμηση, διπλασιάζοντας έτσι το εύρος των θετικών αριθμών του κάθε τύπου. Οι τύποι αυτοί διαχωρίζονται με τους απλούς με τη λέξη unsigned πριν από το όνομα του τύπου.

Σταθερές (constants)

Αν και οι μεταβλητές έχουν σχεδιαστεί να μεταβάλλονται, υπάρχουν περιπτώσεις που χρειαζόμαστε κάποιες σταθερές, όπως στα Μαθηματικά. Μπορούμε να δηλώσουμε μια μεταβλητή ως σταθερά (δηλαδή που να μη μεταβάλλεται) με τη λέξη const. Για παράδειγμα:

```
const double pi = 3. 1415;
```

5.4. Συναρτήσεις

Οι συναρτήσεις είτε επιστρέφουν μία τιμή, είτε επιστρέφουν void, που σημαίνει τίποτε. Μια συνάρτηση που προσθέτει δυο αριθμούς ακέραιους μπορεί να επιστρέψει το άθροισμα, και έτσι θα πρέπει να δηλώνετε ότι επιστέφει ακέραια τιμή. Μια συνάρτηση που απλώς προβάλει ένα μήνυμα, δεν έχει τίποτε να επιστρέψει και θα πρέπει να δηλωθεί ως void. Οι συναρτήσεις αποτελούνται από μια κεφαλίδα και το σώμα τους. Η κεφαλίδα αποτελείτε με την σειρά της, από τον τύπο επιστροφής, το όνομα της συνάρτησης και τις παραμέτρους προς την συνάρτηση αυτή. οι παράμετροι προς μια συνάρτηση επιτρέπουν να διοχετεύονται τιμές στην συνάρτηση. Έτσι, αν η συνάρτηση πρόσθετε δυο αριθμούς, οι αριθμοί θα διοχετευόταν ως παράμετροι στην συνάρτηση. Ακολουθεί μια τυπική κεφαλίδα συνάρτησης :

Int sum (int a, int b).

5.5. Πίνακες

Αν και έχουμε ήδη αναφέρει τους σημαντικότερους τύπους δεδομένων στη C++, αφήσαμε σκόπιμα την αναφορά στους πίνακες. Ένας λόγος είναι ότι στη C++, οι πίνακες υλοποιούνται είτε ως παραδοσιακοί πίνακες (όπως στη C) είτε ως αντικείμενα. Οι πίνακες γενικά χρησιμοποιούνται για την οργάνωση και καταχώρηση όμοιων αντικειμένων. Μπορούν να είναι μίας ή πολλών διαστάσεων και ο τρόπος προσπέλασης των στοιχείων είναι ο ίδιος με αυτόν της C. Στη C++ μπορούμε να ορίσουμε ένα μονοδιάστατο πίνακα πολύ απλά ως εξής: type table[size];

Ως type θεωρούμε τον τύπο δεδομένων των αντικειμένων του πίνακα και μπορεί να είναι είτε ένας από τους απλούς τύπους (bool, char, short, int, long, float, double, char) είτε το όνομα μιας κλάσης αντικειμένων. Το size απεικονίζει το μέγεθος του πίνακα table. Όπως και στη C, μπορούμε να προσπελάσουμε τα στοιχεία του πίνακα με την σύνταξη table[i], όπου i η θέση του στοιχείου που μας ενδιαφέρει. Εδώ πρέπει να σημειωθεί ότι σε αντιστοιχία με τη C η C++ πραγματοποιεί την αρίθμηση των πινάκων από το μηδέν (0) ως το size-1. Δηλαδή αν έχουμε έναν πίνακα A με 10 στοιχεία το πρώτο στοιχείο είναι το A[0] και το τελευταίο το A[9]. Σημειώνεται ότι ενώ στη Java οι πίνακες είναι πραγματικά αντικείμενα και το μέγεθος ενός πίνακα A δίνεται από τη μεταβλητή A. length, στη C++ δεν υπάρχει τέτοια δυνατότητα και το μέγεθος το λαμβάνουμε με τη χρήση της συνάρτησης sizeof(). Η ίδια όμως επιστρέφει το μέγεθος σε bytes και όχι σε στοιχεία του πίνακα.

5.6. Οι Συμβολοσειρές στη C++

Μεταβλητή String

Τα strings τα έχουμε ήδη αναφέρει και τα χρησιμοποιήσαμε μερικές φορές στην εντολή cout. Αντίθετα με τα strings σε άλλες γλώσσες προγραμματισμού (C, PASCAL) που είναι απλώς πίνακες χαρακτήρων, στη C++ τα strings είναι κανονικά αντικείμενα, που υλοποιούνται με την κλάση string (στο namespace std). Φυσικά, για λόγους συμβατότητας με τη C, υπάρχει πλήρης υποστήριξη των strings υπό την μορφή πινάκων χαρακτήρων (char * ή char []).

Πέρα από την απευθείας χρήση τους που είδαμε στην cout, μπορούμε να δημιουργήσουμε αντικείμενα string, με τον ίδιο τρόπο όπως και με κάθε άλλο αντικείμενο, δηλαδή στατικά ή δυναμικά (με τη χρήση της new).

```
std::string str(“Hello”);

std::string str2 = “ there”;

std::string *str3 = new std::string(“Hello there”);

cout << str << str2 << endl;

cout << *str2 << endl;
```

Η κλάση string παρέχει ορισμένες μεθόδους, οι οποίες είναι αρκετά χρήσιμες για επεξεργασία του κειμένου μέσα στο string. Παραθέτουμε τις σημαντικότερες από αυτές στον ακόλουθο πίνακα:

Όνομα	Λειτουργία
bool empty()	Επιστρέφει true αν το αντικείμενο string είναι κενό.
int length()	Επιστρέφει το μήκος (σε χαρακτήρες) του string.

reference operator[](int index)	Επιστρέφει ένα δείκτη αναφοράς (reference) στον χαρακτήρα που βρίσκεται στη θέση index του string.
int compare(string &str)	Συγκρίνει δύο αντικείμενα string. Αν το καλόν αντικείμενο (δηλ. αυτό που καλεί την compare()) είναι μικρότερο από το str, τότε επιστρέφει αρνητικό αποτέλεσμα, μηδέν αν έχουν το ίδιο περιεχόμενο, ή θετικό αποτέλεσμα αν το καλόν string είναι μεγαλύτερο από το str.
int find(string &str)	Αναζητά το str μέσα στο καλόν αντικείμενο string. Αν βρεθεί επιστρέφει τη θέση της πρώτης εμφάνισής του, αλλιώς -1.
int find_last_of(string str)	Αναζητά το str μέσα στο καλόν αντικείμενο string. Αν βρεθεί επιστρέφει τη θέση της τελευταίας εμφάνισής του, αλλιώς -1.

Ακολουθεί ένα παράδειγμα χρήσης των strings:

```
string str1(“Hello there,  from C++!”);
string str2 = “One two three four”;
string str3 = “C++ strings are cool!”;
string *str4 = new string(str3);
int index,  result;
cout << “str1 is “ << str1. length() << “ characters long. ”);
for (int i=0; i < str1. length(); i++)
      cout << str1[i] << “|“;
cout << endl;
if (str3 == *str4)
     cout << “str3 == str4” << endl;
else
     cout << “str3 != str4” << endl;
if (str3 == str2)
    cout << “str3 == str2” << endl;
else
     cout << “str3 != str2” << endl
result = str3. compare(str1);
if (result < 0)
      cout << “str3 < str1” << endl;
else if (result == 0)
```

```
cout << “str3 == str1” << endl;
```

Το αποτέλεσμα του κώδικα αυτού θα είναι:

str1 is 22 characters long.

H|e|l|l|o| |t|h|e|r|e|, | |f|r|o|m| |C|+|+|!|

str3 == str4

str3 != str2

str3 < str1

'C++' exists in str1 in position 18

'C++' does not exist in str2

'C++' exists in str3 in position

Συμβολοσειρές

Η C++ δεν προβλέπει λειτουργίες άμεσου χειρισμού συμβολοσειρών, η έννοια της εκχώρησης μιας συμβολοσειράς ως τιμή σε μια μεταβλητή δεν υπάρχει, διότι δεν προβλέπεται τέτοιος τύπος μεταβλητών. Γι’ αυτό τον λόγο οι λειτουργίες χειρισμού συμβολοσειρών απορρέουν από την κατάλληλη επεξεργασία των μονοδιάστατων πινάκων.

Έτσι η δήλωση:

Static char pinax []={‘δ’ο’μ’ε’ς’};

Θα μπορούσε να θεωρηθεί κατάλληλη για τον χειρισμό της συμβολοσειράς «δομές» μέσω του μονοδιάστατου πίνακα pinax[]

Στον παρακάτω πίνακα φαίνονται μερικές συναρτήσεις επεξεργασίας συμβολοσειρών:

ΣΥΝΑΡΤΗΣΗ	ΛΕΙΤΟΥΡΓΙΑ
Stchr-char*strchr(const char*s, int c);	Επιστρέφει έναν δείκτη στην πρώτη εμφάνιση του χαρακτήρα c στην s. Αν ο χαρακτήρας δεν υπάρχει επιστρέφει NULL
Strstr-char*strstr(const char*s1, const char*s2);	Βρίσκει την πρώτη εμφάνιση μιας συμβολοσειράς σε μια άλλη. επιστρέφει έναν δείκτη στην s1 όπου αρχίζει η s2 διαφορετικά επιστρέφει null
Strcpy-char*strcpy(const*dest, const char*src);	Αντιγράφει τη συμβολοσειρά src στη dest επιστρέφει τη dest
Strcmp-int strcmp(const char*s1, const char*s2)	Συγκρίνει δυο συμβολοσειρές και επιστρέφει: < 0, αν η s1 είναι μικρότερη από την s2 ==0 αν η s1 είναι ίδια με τη s2 > αν η s1 είναι μεγαλύτερη από την s2
Strlen-size_tstrlen (const char*s);	Υπολογίζει το μήκος μιας συμβολοσειράς επιστρέφει τον αριθμό των χαρακτήρων στην s, χωρίς να μετρήσει τον τερματικό null χαρακτήρα
Strcat-char*strcat(char*dest, const char*src);	Προσαρτά την src συμβολοσειρά στη dest επιστρέφει dest
Strrev-char*strrev(char*s);	Αντιστρέφει τους χαρακτήρες στην s

	επιστρέφει έναν δείκτη στην αντιστραμμένη συμβολοσειρά
Strupr-char * strupr(char*s);	Μετατρέπει όλους τους χαρακτήρες στην s σε κεφαλαία. επιστρέφει έναν δείκτη s
Strlwr-char*strwr(char*s);	Μετατρέπει όλους τους χαρακτήρες στην s σε πεζά. επιστρέφει έναν δείκτη στην s

Τι είναι μια συμβολοσειρά στην c++; Μια συμβολοσειρά είναι ένας πίνακας χαρακτήρων που περιλαμβάνει το μηδενικό χαρακτήρα τερματισμού (/0). Για παράδειγμα, αποθηκεύσατε μια συμβολοσειρά στον πίνακα bb [] χαρακτήρα προς χαρακτήρα στο πρόγραμμα με τις παρακάτω δηλώσεις :

bb[0]='c'

bb[1]='a'

bb[2]='t'

bb[3]='/0';

(είναι σημαντικό να αναφέρουμε ότι τα στοιχεία ενός πίνακα αποθηκεύονται σε διαδοχικές και αυξανόμενες θέσεις μνήμης)

Δημιουργία και Επεξήγηση μιας Συμβολοσειράς στη C++

Με την εντολή cin μπορούμε να χειριστούμε δείκτες χαρακτήρων (char*), έτσι μπορούμε να δημιουργήσουμε έναν ενταμιευτή χαρακτήρων και να χρησιμοποιήσουμε την cin για να τον γεμίσουμε για παράδειγμα :

Char your name [50]

Cout << enter your name:";

Cin >>your name

Στο παραπάνω παράδειγμα έχουμε δημιουργήσει έναν πίνακα μεγέθους 50 χαρακτήρων και αν εισάγουμε το όνομα για παράδειγμα eleni θα συμπληρωθεί στη μεταβλητή your name και θα υπάρχει και ο τελευταίος κενός στην ουσία χαρακτήρας τερματισμού το /0,. Η cin τερματίζει αυτόματα την συμβολοσειρά με έναν κενό χαρακτήρα και πρέπει να έχει αρκετό χώρο στον ενταμιευτη για να επιστρέψει και να χωρέσει ολόκληρη την συμβολοσειρά. Γενικά ο κενός χαρακτήρας (/0) συμβολίζει το τέλος της συμβολοσειράς στις συναρτήσεις.

5.7. Τα προγράμματα της C++

Στο σημείο αυτό έχουμε ορίσει τα περισσότερα από τα βασικά εργαλεία για να γράψουμε ένα πλήρες πρόγραμμα σε C++. Δεν έχουμε αναφέρει όμως τη δομή ενός προγράμματος, τον τύπο αρχείων που θα χρησιμοποιηθούν ή πως θα τα μεταγλωττίσουμε και εκτελέσουμε.

Η ρουτίνα main()

Στη C και στη C++, κάθε πρόγραμμα ξεκινά την εκτέλεσή του από την ρουτίνα main(). Η ρουτίνα αυτή μπορεί να βρίσκεται σε οποιοδήποτε αρχείο πηγαίου κώδικα του προγράμματός μας, αλλά μπορεί να είναι μόνο μία για κάθε εκτελέσιμο πρόγραμμα. Σε αντίθεση με τη Java, δεν αποτελεί μέρος μιας κλάσης αλλά είναι αυτόνομη ρουτίνα. Το ακόλουθο είναι το πιο απλό παράδειγμα προγράμματος C++:

```
#include <iostream>

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

std::cout << “hello everyone” << std::endl;

}
```

Με την εντολή #include εισάγουμε την κεφαλίδα που δηλώνεται η χρήση των βασικών stream εισόδου και εξόδου της C++, των cin, cout και cerr αντίστοιχα (περισσότερα σε επόμενη ενότητα). Η κεφαλίδα αυτή είναι η iostream. Όσον αφορά την int αναφέρεται στο ότι η ρουτίνα main() επιστρέφει στο λειτουργικό σύστημα έναν κωδικό επιτυχίας ή αποτυχίας εκτέλεσης του προγράμματος. Σε όλα τα λειτουργικά συστήματα και για λόγους συμβατότητας, ο κωδικός αυτός είναι μηδέν (0) για επιτυχή εκτέλεση του προγράμματος και μη μηδενικό (5, 10, ή άλλο) σε περίπτωση ελεγχόμενου τερματισμού του προγράμματος π. χ. γιατί δεν είχε αρκετή μνήμη ή δικαιώματα εγγραφής σε κάποια αρχεία, ενώ ο κωδικός αυτός αποκτά μεγάλη τιμή όταν το πρόγραμμα τερματίσει απότομα λόγω σφάλματος (crash). Οι παράμετροι argc, argv παίζουν το ρόλο της

παραμέτρου args[] στη Java. Δηλαδή περιέχουν τις παραμέτρους με τις οποίες καλείται το πρόγραμμα από την γραμμή εντολών. Ο λόγος που έχουμε δύο παραμέτρους είναι ότι ένας πίνακας στη C/C++ δεν παρέχει κάποιο εύκολο τρόπο πληροφόρησης του μεγέθους του. Έτσι ενώ ο πίνακας argv[] περιέχει τις παραμέτρους σε strings της C (ακολουθίες χαρακτήρων char και όχι αντικείμενα string), δεν είναι δυνατό να γνωρίζουμε το πλήθος αυτών των παραμέτρων χωρίς την argc. Σε επόμενη παράγραφο ακολουθεί παράδειγμα χρήσης των argc, argv. Το cout το έχουμε ήδη δει αρκετές φορές σε προηγούμενα παραδείγματα, είναι το stream που αντιστοιχεί στη πρότυπη έξοδο (standard output) όπου τυπώνονται τα μηνύματα (την ίδια λειτουργία σε άλλες γλώσσες έχουν εντολές όπως print, printf, write, κλπ). Περισσότερα για τα streams στη C++ θα δούμε σε επόμενη ενότητα. Το endl συμβολίζει το χαρακτήρα νέας γραμμής (new line) και έχει ακριβώς την ίδια λειτουργία με την εκτύπωση του χαρακτήρα “\n” (για την ακρίβεια κάνει και flush το stream εξόδου). Θα παρατηρήσατε ίσως ότι το cout και το endl έχουν το πρόθεμα std με διπλές άνω και κάτω τελείες “::”. Το πρόθεμα αυτό συμβολίζει το namespace στο οποίο ανήκει το αντικείμενο stream cout και το endl, δηλαδή στην ουσία το πεδίο ισχύος τους. Αν ο κώδικάς μας άνηκε στο ίδιο namespace std, δε θα ήταν αναγκαία η χρήση του προθέματος “std::”, αλλά το συγκεκριμένο namespace είναι ήδη καθορισμένο και δεν συνιστάται η τροποποίηση ή προσθήκη άλλων κλάσεων ή αντικειμένων. Οποιαδήποτε χρήση αντικειμένου εκτός κάποιου namespace απαιτεί το πρόθεμα του ονόματος του namespace (στην προκειμένη περίπτωση το std) ακολουθούμενο από '::'.

5.8. Αλγόριθμοι εύρεσης συμβολοσειράς σε κείμενο

5.8.1. Αλγόριθμος Brute Force

Όπως είδαμε και στην πρώτη ενότητα υπάρχουν διάφοροι αλγόριθμοι αναζήτησης συμβολοσειρών, σε αυτό το κεφάλαιο θα αναπτύξουμε δυο παραδείγματα στους αλγορίθμους αναζήτησης συμβολοσειρών brute_force η αλλιώς ωμό αλγόριθμο και στον αλγόριθμο kmp_search. Ο αλγόριθμος brute_force στην χειρότερη περίπτωση απαιτεί χρόνο της τάξης O(M*N) αλλά στη μέση περίπτωση απαιτεί χρόνο της τάξης O(N). Ενώ σε αντίθεση ο αλγόριθμος KMP με την μέθοδο του πίνακα next γίνετε πιο γρήγορος και στην χειρότερη και μέση περίπτωση απαιτεί χρόνο της τάξης O(N).

Παράδειγμα ανάπτυξης αλγορίθμου Brute_force στην c++

```
1    #include <iostream. h>
2
3     main()
4  {
5    int i=0, j=0, found=0, index=0;
6    char a[100]="";
7   char p[30]="";
8    int N,  M;
9
10   cout << "Brutal Force Algorithm" << endl;
11  cout << "give the phrase" << endl;
12  cin. getline(a, 100);
13  cout << "the phrase you wrote is " << a << endl;
```

```
14
15       for( N=0; a[N]!='\0'; N++)
16           ;
17         cout << "Your phrase consists of " << N << " characters" <<
18                          endl;
19
20        cout << "give the word" << endl;
21        cin. getline(p, 30);
22       cout << "the word you wrote is " << p << endl;
23
24           for( M=0;p[M]!='\0';M++)
25         ;
26      cout << "Your phrase consists of " << M << " characters" <<
27           endl;
28
29       do
30      {
31
32     if( a[i] == p[j] )
33      {
34       i++;
35        j++;
```

```
36       cout << "mpika stin if. i="<< i << " kai j=" << j << endl;
37          }
38     else
39      {
40         i=i-j+1;
41         j=0;
42         cout << "mpika sto else kai i=" << i << " kai j=" << j << endl;
43         }
44       } while ( (j<=M-1) && (i<=N-1) );
45
46    if (j>M-1)
47    {
48       found=1;
49       index=i-M;
50     }
51
52     if(found)
53     {
54              cout << "the word is found in the phrase,  in the position " <<
55 index << " of the table,  that is on the character number " <<
56 index+1 << endl;
57     }
```

```
58    else
59    {
60       cout << "the word is not found inside the phrase" << endl;
61     }
62
63     system("PAUSE");
64    return 0;
65     }
```

Ανάλυση και Επεξήγηση του κώδικα:

1η γραμμή

Αρχικά στο πρόγραμμα μας θα συμπεριλάβουμε την βιβλιοθήκη <iostream> η οποία χρησιμοποιείται από την cout και η οποία εκτελείτε για να μπορούμε να έχουμε προβολή στην οθόνη. Όπως είναι γνωστό η επικάλεση αυτής της βιβλιοθήκης γίνετε στην αρχή δηλαδή στην γραμμή ένα.

5η – 6η γραμμή

Στην γραμμή πέντε και έξι ξεκινάμε και δηλώνουμε τους δείκτες μας καθώς και τους πίνακές μας. Είναι σημαντικό να αναφέρουμε και θα μας χρειαστεί πιο μετά για να αυτόν τον κώδικα ότι οι δείκτες I και j ξεκινάμε με την τιμή μηδέν και όχι με την τιμή ένα όπως όταν γράφουμε έναν πίνακα σε ψευδοκώδικα στο χαρτί.

Παράδειγμα Πίνακας σε ψευδοκώδικα

δ	ο	μ	ε	ς
Θεση1	θεση2	θεση3	θεση4	θεση5

Πίνακας στη γλώσσα προγραμματισμού c++

δ	ο	μ	ε	ς	/0
Θέση 0	Θέση 1	θεση2	θεση3	θεση4	τερματισμός

(στη δημιουργία πίνακα σε c++ μετράει ο δείκτης από το μηδέν και όχι από το ένα)

Στο πρόγραμμα μας φτιάχνουμε δυο πίνακες έναν για το κείμενο το οποίο θα γράφουμε και θα αναζητούμε μια λέξη και ο οποίος πίνακας θα έχει μέγεθος εκατό και δημιουργούμε και έναν δεύτερο πίνακα για την λέξη πρότυπο, δηλαδή την λέξη που θα αναζητούμε μέσα σε ένα κείμενο και ο οποίος πίνακας θα έχει μέγεθος τριάντα. Οι ακέραιοι μας θα είναι ο Μ και Ν όπου ο Μ συμβολίζει το μέγεθος του πίνακα για την πρότυπη λέξη την οποία και θα αναζητήσουμε μέσα στο κείμενο, και αντίστοιχα το Ν θα συμβολίζει το μέγεθος του πίνακα που θα αντιστοιχεί στο κείμενο μας.

10η – 13η γραμμή

Στις γραμμές από δέκα μέχρι δεκατρία χρησιμοποιήσαμε τις εντολές cout και cin με σκοπό να δημιουργήσουμε μια αλληλεπίδραση ανάμεσα σε χρήστη και πρόγραμμα.

15η γραμμή

Στην γραμμή δεκαπέντε χρησιμοποιήσαμε την πρόταση for η αλλιώς χρησιμοποιήσαμε έναν βρόχο for με σκοπό να περιορίσουμε το μέγεθος του πίνακα μόνο στους χαρακτήρες που πληκτρολογούνται κάθε φορά από τον χρήστη, διαφορετικά το πρόγραμμα μας θα αναζητούσε και θα μέτραγε κάθε φορά όλο το μέγεθος του πίνακα που είχαμε δηλώσει από την αρχή. Για να αποφύγουμε αυτή την δυσκολία, δημιουργήσαμε ένα βρόχο for ο οποίος περιορίζει το μέγεθος του πίνακα του κειμένου εκεί που τελειώνει το ίδιο το κείμενο κάθε φορά που δημιουργείτε. Ουσιαστικά του δίνουμε εντολή να αυξάνει μια θέση κάθε φορά που βρίσκει χαρακτήρα και να σταματήσει όταν συναντήσει το τερματικό στοιχείο του πίνακα /0.

For (αρχική_τιμή; Έλεγχος ενέργεια)

Πρόταση;

Η πρόταση αρχική_τιμή χρησιμοποιείτε για να δώσει την κατάσταση ενός μετρητή, η διαφορετικά, για να προετοιμάσει τον βρόχο. Η πρόταση έλεγχος είναι οποιαδήποτε έγκυρη έκφραση της c++ και εκτιμάται κάθε φορά που γίνετε διέλευση από τον βρόχο. Εάν η έλεγχος είναι “true”, εκτελείτε η ενεργεία (τυπικά αυξάνετε ο μετρητής)και κατόπιν εκτελείτε το σώμα του βρόχου for.

Παράδειγμα:

// απλό πρόγραμμα που προβάλει την λέξη hello δέκα φορές

for (int i=0; i<10; i++)

cout <<”hello!”;

Οι προτάσεις for είναι ισχυρές και ευέλικτες. οι τρεις ανεξάρτητες προτάσεις (αρχική_τιμή, ελεγχος, και ενεργεία)προσφέρονται σε έναν αριθμό παραλλαγών. Γενικότερα όμως ένας βρόχος for λειτούργει με την εξής σειρά :

εκτελεί τις λειτουργίες της απόδοσης αρχικής τιμής

εκτιμά την συνθήκη.

εάν η συνθήκη είναι αληθής, εκτελεί την πρόταση ενεργείας και τον βρόχο.

μετά από κάθε έλεγχο ο βρόχος επαναλαμβάνει τα βήματα μέχρι να βρεθεί σε “false”

20η -27η γραμμή

Στις γραμμές από είκοσι μέχρι και είκοσι εφτά χρησιμοποιούμε την ιδία διαδικασία και για την λέξη πρότυπο όπως προγραμματίσαμε και για το βασικό μας κείμενο.

29η γραμμή

Ενώ από την είκοσι εννιά γραμμή και μετά ξεκινάμε να προγραμματίζουμε την διαδικασία αναζήτησης μια συμβολοσειράς χρησιμοποιώντας τον αλγόριθμο brute_force. Οι διαδικασία εντολών απαιτεί τον προγραμματισμό δεικτών i και αντίστοιχα j. Μεταφράζοντας τον ψευδοκώδικα του αλγορίθμου σε γλωσσά προγραμματισμού c++ δημιουργείτε μια σημαντική αντιπαράθεση. Ενώ στον ψευδοκώδικα του αλγορίθμου έχουμε την έκφραση εάν ο δείκτης i του κειμένου είναι ίσος με τον δείκτη J του προτύπου τότε προχωρά στον επόμενο χαρακτήρα αλλιώς. . . Στην γλωσσά προγραμματισμού c++ δεν υπάρχει τέτοιου είδους πρόταση ώστε να μπορέσουμε να εκφράσουμε το «εάν» «αλλιώς». Για αυτόν τον λόγο χρησιμοποιούμε τις προτάσεις do δηλαδή κάνε και while δηλαδή ενώ. Πιο αναλυτικά στην γραμμή εικοσιεννέα ξεκινάμε την πρόταση μας με τον βρόχο do. . . while που θα είναι και το σώμα της πρότασης μας.

Πιο συγκεκριμένα ο βρόχος do … while εκτελεί το σώμα του βρόχου πριν ελέγξει την συνθήκη και εξασφαλίζει ότι το σώμα του βρόχου εκτελείτε πάνα, τουλάχιστον μια φόρα.

Η πρόταση do…while

Η σύνταξη για την πρόταση do…while έχει ως ακολούθως:

Do

Πρόταση

While (συνθήκη);

Η πρόταση εκτελείτε και κατόπιν εκτελείτε η συνθήκη. Εάν η συνθήκη είναι αληθής, επαναλαμβάνετε ο βρόχος. Διαφορετικά, τερματίζεται.

32η γραμμή

Μπαίνοντας στον κορμό της συνθήκης πιο συγκεκριμένα στην γραμμή τριάντα δυο, ξεκινάμε να προγραμματίζουμε τους δείκτες μας χρησιμοποιώντας την πρόταση if…. else. Πιο αναλυτικά και σε πιο ελεύθερη μετάφραση ξεκινάμε

δηλώνοντας εάν (if) ο δείκτης i στον πίνακα [a] είναι ίσος με τον δείκτη j στον πίνακα [p] τότε αύξησε κατά ένα και τους δυο δείκτες (i++, j++). Διαφορετικά (else) ο δείκτης i να πάρει τιμή ίση με i-j+1. Σημαντικό είναι να αναφέρουμε ότι στον ψευδοκώδικα του αλγόριθμου brute_force το i σε περίπτωση ψευδής συνθήκης παίρνει τιμή i-j+2. Αυτό γίνετε γιατί όπως αναφέραμε και πιο πάνω ένας πίνακας σε γλώσσα c++ ξεκινάμε να τον μετράμε από το μηδέν και από το ένα όπως κάνουμε σε έναν ψευδοκώδικα.

Η πρόταση if

Η πρόταση if μας δίνει την δυνατότητα να ελέγχουμε την αλήθεια μιας συνθήκης και να μεταβαίνουμε σε διαφορετικά μέρη του κώδικα μας, ανάλογα με το αποτέλεσμα μας.

Η απλούστερη μορφή μιας πρότασης if είναι :

```
If(έκφραση)
    Πρόταση;
```

Πχ:

```
If (big_number>small_number)
   Big_number=small_number
```

Εάν η έκφραση έχει τιμή μηδέν θεωρείται ψευδής και παρακάμπτεται η πρόταση. Εάν έχει μη μηδενική τιμή, θεωρείτε αληθής και εκτελείτε η πρόταση. Ο παραπάνω κώδικας συγκρίνει τις τιμές bignumber και smallnumber, εάν είναι μεγαλύτερη η big number η δεύτερη γραμμή ορίζει την τιμή της στην τιμή της smallnumber.

Μια πιο ένθετη μορφή της if απεικονίζεται κάπως έτσι:

```
If (έκφραση)
  Πρόταση;
Else
```

Προταση2;

Επόμενη πρόταση;

Εάν η έκφραση είναι true, εκτελείται η πρόταση 1. διαφορετικά, εκτελείτε η πρόταση 2. Μετά το πρόγραμμα συνεχίζει με την επόμενη πρόταση.

44η γραμμή

Στην γραμμή σαράντα τέσσερα του κώδικα μας βλέπουμε την συνεχεία την πρότασης που είχαμε ξεκινήσει και αναφέραμε και πιο πάνω do…while. Ουσιαστικά αυτό που θέλουμε να επιτύχουμε με τον συνδυασμό εκφράσεων if…else και do…. while είναι να ερμηνεύουμε την ιδέα του αλγορίθμου με την ακόλουθη λογική: Κάνε. . . εάν ο δείκτης i του πίνακα a είναι ίσος με τον δείκτη j του πίνακα p αύξησε κατά ένα τους δείκτες διαφορετικά ο i δείκτης να παίρνει τιμή ίση i-j+1. Κάνε όλα αυτά ενώ ο j παραμένει μικρότερος, ίσος του M-1 όπου M σημαίνει μέγεθος πίνακα πρότυπου και αντίστοιχα όσο το i παραμένει μικρότερο του N-1. Εάν ο δείκτης j φτάσει να γίνει μεγαλύτερος του M-1 και αντίστοιχα ο δείκτης i μεγαλύτερος του N-1 τότε θα έχουμε την found =1 ίση με ένα που σημαίνει ότι η συνθήκη μας θα είναι αληθής. Πιο συγκεκριμένα σημαίνει ότι η λέξη πρότυπο ταυτίστηκε με κάποια λέξη του κειμένου και βρέθηκε!

Διαφορετικά όταν το πρόγραμμα τρέξει και ψάξει όλους τους χαρακτήρες του κειμένου και δεν βρεθεί η λέξη πρότυπο θα βγάλει στη οθόνη ένα μήνυμα που θα επεξηγεί ότι η λέξη δεν βρέθηκε στο κείμενο.

Συμπεράσματα για τον αλγόριθμο Brute Force

Κλείνοντας το κομμάτι του αλγορίθμου BRUTE_FORCE θα μπορούσαμε αρχικά να συμπεράνουμε ότι δεν πήρε τυχαία αυτό το όνομα, στην ελληνική γλώσσα σημαίνει ωμή δύναμη. Ο BRUTE_FORCE είναι από τους πιο χαρακτηριστικούς αλγορίθμους αναζήτησης πάνω στις συμβολοσειρές και ο πιο κατανοητός, χωρίς βέβαια αυτό να σημαίνει ότι είναι και ο πιο γρήγορος. Υπάρχουν πολύ πιο γρήγορες μέθοδοι και αλγόριθμοι αναζήτησης όπως για παράδειγμα οι συναρτήσεις κατακερματισμού. Σίγουρα όμως είναι πολύ πιο πολύπλοκες και πιο δύσκολες να αναπτυχθούν σε μια γλώσσα προγραμματισμού. Η επιλογή ανάπτυξης του αλγορίθμου BRUTAL_FORCE έγινε κατά κύριο λόγο γιατί είναι η αρχή της ιδέας για το πώς κάποιος μπορεί να προγραμματίσει ένα απλό πρόγραμμα αναζήτησης μιας συμβολοσειράς σε ένα κείμενο σε διάφορες γλώσσες προγραμματισμού και κατά δεύτερο λόγο γιατί η δομή του και η φύση του μπορεί να καθοδηγήσει τον χρήστη η τον προγραμματιστή να μπει και να κατανοήσει πιο εύκολα τον κόσμο τον αλγορίθμων και του ψευδοκώδικα και στην συγκεκριμένη περίπτωση να ενταχθεί στον κόσμο των συμβολοσειρών…

5.11.2. Αλγόριθμος KMP

```
#include <iostream. h>

int main()

{

  char a[100];

  char p[30];

  int N, M;

  int i=0, j=-1, found=0, index=0;


  cout << "KMP Search Algorithm" << endl;
```

```
cout << "give the phrase" << endl;

cin. getline(a, 100);

cout << "the phrase you wrote is " << a << endl;


for(N=0;p[N]!='\0';N++)

    ;

cout << "give the word" << endl;

cin. getline(p, 30);

cout << "the word you wrote is " << p << endl;


for(M=0;p[M]!='\0';M++)

    ;

//Dhmioyrgia pinaka next

int next[M];

next[0]=-1;

do

{

  if( (j==-1)||(p[i]==p[j]) )

  {


    i++;

    j++;
```

```
    cout << "mpika stin if. i="<< i << " kai j=" << j << endl;


    if(p[i]!=p[j])
    {
    next[i]=j;
    cout << "mpika sto esoteriko if kai next[" << i << "]=" << next[i] << endl;
    }
    else
    {
    next[i]=next[j];
    cout << "mpika sto esoteriko else kai next[" << i << "]=" << next[i] << endl;
    }
  }
  else
  {
    cout << "mpika sto else" << endl;
    j=next[j];
  }
  cout << "teleiono to do,  to i einai "<< i << " kai to j einai " << j << endl;
  cout << "---------------------------" << endl;
 } while(i<=M-1);
```

```
  // Anazitisi toy protypoy mesa sti frash
  i=0;
  j=0;

  do
  {
     cout << "mpika sto deytero Do" << endl << "-----------------------------------" <<
endl;
     if((j==-1)||(a[i]==p[j]))
     {
       i++;
       j++;
       cout << "mpika sto if toy deyteroy do i=" << i << " j=" << j << endl;
     }
     else
     {
       j=next[j];
       cout << "mpika sto else toy deyteroy do i=" << i << " j=" << j << endl;
     }
  }
  while ( (j<=M-1)&&(i<=N-1) );
```

```
        if (j>M-1)
        {
          found=1;
          index=i-M;
        }

        if(found)
        {
        cout << "the word is found in the phrase,  in the position " << index << " of the table,  that is on the character number " << index+1 << endl;
        }
        else
        {
        cout << "the word is not found inside the phrase" << endl;
       }

     system("PAUSE");
     return 0;
}
```

Ο Αλγόριθμος Not So Naive σε C++

```
#include <iostream>
#include <string>
```

using namespace std;

Ο αλγόριθμος Not So Naive εκτελεί αναζήτηση ενός αλφαριθμητικού – προτύπου (pattern) μέσα σε κείμενο (text) και συνιστά μία ακόμα εφαρμογή του μηχανισμού του κυλιόμενου παράθυρου. Ο μηχανισμός δουλεύει ως εξής: το κείμενο σαρώνεται με τη βοήθεια ενός παραθύρου του οποίου το μέγεθος είναι m, όσο και το μέγεθος του προτύπου. Η αριστερή άκρη αυτού του παραθύρου καθώς και του κειμένου στοιχίζονται και έπειτα συγκρίνονται οι χαρακτήρες του κειμένου που βρίσκονται μέσα στο παράθυρο με τους χαρακτήρες του προτύπου. Μετά από επιτυχημένο ταίριασμα του προτύπου ή μιας αποτυχίας ταιριάσματος το παράθυρο κυλά προς τα δεξιά. Η ίδια διαδικασία επαναλαμβάνεται ωσότου η δεξιά άκρη του παραθύρου ξεπεράσει τη δεξιά άκρη του κειμένου.

int main ()

{

p είναι ο πίνακας χαρακτήρων μεγέθους 50 θέσεων, ο οποίος κρατά το string του προτύπου (pattern) και το μήκος του αφού υπολογιστεί, θα καταχωρηθεί στη μεταβλητή m.

t είναι ο πίνακας χαρακτήρων μεγέθους 250 θέσεων, ο οποίος κρατά το string του κειμένου (text) και το μήκος του αφού υπολογιστεί, θα καταχωρηθεί στη μεταβλητή n.

j είναι η θέση του κειμένου από την οποία ξεκινά το κυλιόμενο παράθυρο ώστε να αρχίσει η σύγκριση των γραμμάτων.

```
  char p[50]="";

  char t[250]="";

  int m, n, j, k, ell;
```

Εμφανίζεται κείμενο που λέει στο χρήστη ότι χρησιμοποιεί τον αλγόριθμο “Not So Naive” και τον προτρέπει να δώσει το κείμενο και το πρότυπο το οποίο αναζητείται μέσα στο κείμενο. Η συνάρτηση getline καταχωρεί σε πίνακα χαρακτήρων ό, τι πληκτρολογείτε ως είσοδος από το χρήστη.

```
cout << "Not So Naive Algorithm" << endl;

cout << "Enter the pattern at least two characters: " << endl;

cin. getline(p, 50);

cout << "Enter the text: " << endl;

cin. getline(t, 250);
```

Με τη δομή επανάληψης for υπολογίζεται το μήκος του προτύπου, καθώς και του κειμένου. Ουσιαστικά, δεν εκτελείται καμία εντολή εντός της δομής, απλά πετυχαίνουμε σε κάθε επανάληψη να αυξάνεται η μεταβλητή μετρητής, η οποία στο τέλος καταλήγει να έχει τιμή ίση με το μήκος του αλφαριθμητικό που είναι αποθηκευμένο σε πίνακα χαρακτήρων.

```
for(m=0;p[m]!='\0';m++) ;

for(n=0;t[n]!='\0';n++) ;
```

Στη φάση της προ-επεξεργασίας που ακολουθεί, ο αλγόριθμος διαχωρίζει δύο περιπτώσεις για το string του προτύπου (pattern): μία στην οποία τα πρώτα δύο γράμματα είναι ίδια και μία στην οποία αυτό δεν συμβαίνει. Αυτός ο διαχωρισμός γίνεται με τον έλεγχο p[0] = = p[1] και σε περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το παράθυρο εντοπισμού του προτύπου θα κυλά κατά δύο θέσεις δεξιά κάθε φορά, το οποίο οφείλεται στην εκχώρηση της τιμής 2 στη μεταβλητή k. Αυτό το επιθυμούμε, διότι αν οι χαρακτήρες p[1] και t[j+1] είναι ανόμοιοι, σίγουρα θα είναι επίσης ανόμοιοι οι p[0] και t[j+1]. Η δεύτερη σύγκριση δεν θα γίνει όμως λόγω της τιμής 2 που έχει εκχωρηθεί στη μεταβλητή ell και όπως θα δούμε παρακάτω, ο δείκτης j στον

πίνακα χαρακτήρων που κρατά το κείμενο θα μετατοπίζεται κατά 2 θέσεις και όχι 1.

```
if (p[0] == p[1]) {
   k = 2;
   ell = 1;
}
else {
   k = 1;
   ell = 2;
}
```

Στη φάση της αναζήτησης του προτύπου μέσα στο string, ο έλεγχος j <=n-m εξασφαλίζει ότι όταν το κείμενο θα φτάνει στο τέλος του, η σύγκριση των χαρακτήρων του προτύπου και του κειμένου θα συμβαίνει μόνο μέχρις ότου αυτά να έχουν στοιχιστεί ως προς το τέλος τους, χωρίς το πρότυπο να έχει μεγαλύτερο μήκος από τους χαρακτήρες που απομένουν στο κείμενο. Έτσι, εξασφαλίζεται ότι το παράθυρο θα είναι αρκετά μεγάλο ώστε να χωρά το πρότυπο και όταν αυτό δεν θα συμβαίνει, ο βρόχος επανάληψης τερματίζεται. Ο έλεγχος p[1] != t[j + 1] βγαίνει αληθής όταν τα αντίστοιχα στοιχεία του προτύπου και του κειμένου δεν ταυτίζονται. Σε αυτή την περίπτωση εκτελούνται οι εντολές του τμήματος if, τουτέστιν ο δείκτης j αυξάνεται κατά k θέσεις. Όπως προαναφέραμε, αν οι 2 πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το k είναι 2, ώστε να αποφευχθεί αργότερα ο σίγουρα ανεπιτυχής έλεγχος του p[0] με το t[j].

```
j = 0;
while (j <= n - m)
   if (p[1] != t[j + 1])
```

Αν τώρα οι χαρακτήρες στη δεύτερη θέση των δύο αλφαριθμητικών δεν ταυτίζονται, δηλαδή ο έλεγχος p[1] != t[j+1] βγαίνει αρνητικός, το παράθυρο μετατοπίζεται k θέσεις δεξιά πάνω στο κείμενο. Όπως αναφέραμε παραπάνω, το k είναι 1 στην περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου δεν είναι ίδιοι. Και τούτο διότι μπορεί ο χαρακτήρας του παραθύρου του κειμένου που δεν ταυτιζόταν με τον δεύτερο χαρακτήρα του προτύπου, να ταυτίζεται με τον πρώτο. Και επειδή αυτό δεν μπορεί να συμβαίνει όταν οι δύο πρώτοι χαρακτήρες του προτύπου είναι ίδιοι, σε αυτή την περίπτωση το k είναι 2.

```
    j += k;
else {
```

Αν ο έλεγχος βγει ψευδής, εκτελούνται οι εντολές του τμήματος else, οι οποίες είναι μια εμφωλευμένη δομή if και μία εντολή προσαύξησης της μεταβλητής j. Εφόσον λοιπόν τα στοιχεία στη δεύτερη θέση των strings είναι ίδια, γίνεται έλεγχος και στα δεξιά μέρη τους, στην αμέσως επόμενη θέση του πίνακα χαρακτήρων με τη συνάρτηση memcmp, η οποία συγκρίνει τους πρώτους m-2 χαρακτήρες του προτύπου και του κειμένου, ξεκινώντας από τη θέση 2 και j+2 αντίστοιχα. . Αν η σύγκριση δείξει ταύτιση, η συνάρτηση επιστρέφει τιμή 1. Για να θεωρηθεί όμως ότι βρέθηκε το πρότυπο, πρέπει επίσης οι πρώτοι χαρακτήρες του προτύπου και του παραθύρου να είναι ίδιοι. Αν αυτές οι δύο προϋποθέσεις πληρούνται, έχουμε μία επιτυχημένη απόπειρα εύρεσης του προτύπου μέσα στο κείμενο και εμφανίζεται αντίστοιχο μήνυμα στο χρήστη.

```
    if (memcmp(p + 2,  t + j + 2,  m - 2) = = 0 &&  p[0] = = t[j])
       cout << "Pattern is found at position " << j << endl;
```

Είτε ταιριάζει το πρότυπο στους χαρακτήρες του παραθύρου είτε όχι, σε αυτό το σημείο ο δείκτης j αυξάνει κατά ell. Όπως προαναφέραμε, το ell είναι 2 στην περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου δεν ταυτίζονται. Αυτό γίνεται ώστε να γλιτώσουμε τον σίγουρα αποτυχημένο έλεγχο του πρώτου χαρακτήρα του προτύπου με τον αμέσως επόμενο χαρακτήρα του παραθύρου, διότι αυτός ταυτιζόταν στο προηγούμενο βήμα με τον χαρακτήρα στη δεύτερη

θέση του προτύπου, ο οποίος είναι διαφορετικός από αυτόν στην πρώτη θέση. Αυτή η σύγκριση παρακάμπτεται, με κέρδος σε υπολογιστικούς πόρους. Αν ο πρώτος με τον δεύτερο χαρακτήρα του προτύπου ταυτίζονται, το ell είναι 1 και σε αυτή την περίπτωση ο δείκτης j αυξάνει κατά 1.

```
      j += ell;
    }
  system("PAUSE");
  return 0;
}
```

5.11.3. Αλγόριθμος Not So Naive

```
#include <iostream>
#include <string>
using namespace std;
```

Ο αλγόριθμος Not So Naive εκτελεί αναζήτηση ενός αλφαριθμητικού – προτύπου (pattern) μέσα σε κείμενο (text) και συνιστά μία ακόμα εφαρμογή του μηχανισμού του κυλιόμενου παράθυρου. Ο μηχανισμός δουλεύει ως εξής: το κείμενο σαρώνεται με τη βοήθεια ενός παραθύρου του οποίου το μέγεθος είναι m, όσο και το μέγεθος του προτύπου. Η αριστερή άκρη αυτού του παραθύρου καθώς και του κειμένου στοιχίζονται και έπειτα συγκρίνονται οι χαρακτήρες του κειμένου που βρίσκονται μέσα στο παράθυρο με τους χαρακτήρες του προτύπου. Μετά από επιτυχημένο ταίριασμα του προτύπου ή μιας αποτυχίας ταιριάσματος το παράθυρο κυλά προς τα δεξιά. Η ίδια διαδικασία επαναλαμβάνεται ωσότου η δεξιά άκρη του παραθύρου ξεπεράσει τη δεξιά άκρη του κειμένου.

```
int main ()
{
```

p είναι ο πίνακας χαρακτήρων μεγέθους 50 θέσεων, ο οποίος κρατά το string του προτύπου (pattern) και το μήκος του αφού υπολογιστεί, θα καταχωρηθεί στη μεταβλητή **m**.
t είναι ο πίνακας χαρακτήρων μεγέθους 250 θέσεων, ο οποίος κρατά το string του κειμένου (text) και το μήκος του αφού υπολογιστεί, θα καταχωρηθεί στη μεταβλητή **n**.
j είναι η θέση του κειμένου από την οποία ξεκινά το κυλιόμενο παράθυρο ώστε να αρχίσει η σύγκριση των γραμμάτων.

```
char p[50]="";
char t[250]="";
int m,n,j,k,ell;
```

Εμφανίζεται κείμενο που λέει στο χρήστη ότι χρησιμοποιεί τον αλγόριθμο “Not So Naive” και τον προτρέπει να δώσει το κείμενο και το πρότυπο το οποίο αναζητείται μέσα στο κείμενο. Η συνάρτηση getline καταχωρεί σε πίνακα χαρακτήρων ό,τι πληκτρολογείται ως είσοδος από το χρήστη.

```
cout << "Not So Naive Algorithm" << endl;
cout << "Enter the pattern at least two characters: " << endl;
cin.getline(p,50);
cout << "Enter the text: " << endl;
cin.getline(t,250);
```

Με τη δομή επανάληψης for υπολογίζεται το μήκος του προτύπου, καθώς και του κειμένου. Ουσιαστικά, δεν εκτελείται καμία εντολή εντός της δομής, απλά πετυχαίνουμε σε κάθε επανάληψη να αυξάνεται η μεταβλητή μετρητής, η οποία στο τέλος καταλήγει να έχει τιμή ίση με το μήκος του αλφαριθμητικό που είναι αποθηκευμένο σε πίνακα χαρακτήρων.

```
for(m=0;p[m]!='\0';m++) ;
for(n=0;t[n]!='\0';n++) ;
```

Στη φάση της **προ-επεξεργασίας** που ακολουθεί, ο αλγόριθμος διαχωρίζει δύο περιπτώσεις για το string του προτύπου (pattern): μία στην οποία τα πρώτα δύο γράμματα είναι ίδια και μία στην οποία αυτό δεν συμβαίνει. Αυτός ο διαχωρισμός γίνεται με τον έλεγχο p[0] = = p[1] και σε περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το παράθυρο εντοπισμού του προτύπου θα κυλά κατά δύο θέσεις δεξιά κάθε φορά, το οποίο οφείλεται στην εκχώρηση της τιμής 2 στη μεταβλητή k.
Αυτό το επιθυμούμε, διότι αν οι χαρακτήρες p[1] και t[j+1] είναι ανόμοιοι, σίγουρα θα είναι επίσης ανόμοιοι οι p[0] και t[j+1]. Η δεύτερη σύγκριση δεν θα γίνει όμως λόγω της τιμής 2 που έχει εκχωρηθεί στη μεταβλητή ell και όπως θα δούμε παρακάτω, ο δείκτης j στον πίνακα χαρακτήρων που κρατά το κείμενο θα μετατοπίζεται κατά 2 θέσεις και όχι 1.

```
if (p[0] == p[1]) {
   k = 2;
   ell = 1;
}
else {
   k = 1;
   ell = 2;
}
```

Στη φάση της **αναζήτησης** του προτύπου μέσα στο string, ο έλεγχος j <=n-m εξασφαλίζει ότι όταν το κείμενο θα φτάνει στο τέλος του, η σύγκριση των χαρακτήρων του προτύπου και του κειμένου θα συμβαίνει μόνο μέχρις ότου αυτά να έχουν στοιχιστεί ως προς το τέλος τους, χωρίς το πρότυπο να έχει μεγαλύτερο μήκος από τους χαρακτήρες που απομένουν στο κείμενο. Έτσι,

εξασφαλίζεται ότι το παράθυρο θα είναι αρκετά μεγάλο ώστε να χωρά το πρότυπο και όταν αυτό δεν θα συμβαίνει, ο βρόχος επανάληψης τερματίζεται. Ο έλεγχος p[1] != t[j + 1] βγαίνει αληθής όταν τα αντίστοιχα στοιχεία του προτύπου και του κειμένου δεν ταυτίζονται. Σε αυτή την περίπτωση εκτελούνται οι εντολές του τμήματος if, τουτέστιν ο δείκτης j αυξάνεται κατά k θέσεις. Όπως προαναφέραμε, αν οι 2 πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το k είναι 2, ώστε να αποφευχθεί αργότερα ο σίγουρα ανεπιτυχής έλεγχος του p[0] με το t[j].

```
j = 0;
while (j <= n - m)
   if (p[1] != t[j + 1])
```

Αν τώρα οι χαρακτήρες στη δεύτερη θέση των δύο αλφαριθμητικών δεν ταυτίζονται, δηλαδή ο έλεγχος p[1] != t[j+1] βγαίνει αρνητικός, το παράθυρο μετατοπίζεται k θέσεις δεξιά πάνω στο κείμενο. Όπως αναφέραμε παραπάνω, το k είναι 1 στην περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου δεν είναι ίδιοι. Και τούτο διότι μπορεί ο χαρακτήρας του παραθύρου του κειμένου που δεν ταυτιζόταν με τον δεύτερο χαρακτήρα του προτύπου, να ταυτίζεται με τον πρώτο. Και επειδή αυτό δεν μπορεί να συμβαίνει όταν οι δύο πρώτοι χαρακτήρες του προτύπου είναι ίδιοι, σε αυτή την περίπτωση το k είναι 2.

```
     j += k;
 else {
```

Αν ο έλεγχος βγει ψευδής, εκτελούνται οι εντολές του τμήματος else, οι οποίες είναι μια εμφωλευμένη δομή if και μία εντολή προσαύξησης της μεταβλητής j. Εφόσον λοιπόν τα στοιχεία στη δεύτερη θέση των strings είναι ίδια, γίνεται έλεγχος και στα δεξιά μέρη τους, στην αμέσως επόμενη θέση

του πίνακα χαρακτήρων με τη συνάρτηση memcmp, η οποία συγκρίνει τους πρώτους m-2 χαρακτήρες του προτύπου και του κειμένου, ξεκινώντας από τη θέση 2 και j+2 αντίστοιχα.. Αν η σύγκριση δείξει ταύτιση, η συνάρτηση επιστρέφει τιμή 1. Για να θεωρηθεί όμως ότι βρέθηκε το πρότυπο, πρέπει επίσης οι πρώτοι χαρακτήρες του προτύπου και του παραθύρου να είναι ίδιοι. Αν αυτές οι δύο προϋποθέσεις πληρούνται, έχουμε μία επιτυχημένη απόπειρα εύρεσης του προτύπου μέσα στο κείμενο και εμφανίζεται αντίστοιχο μήνυμα στο χρήστη.

```
if (memcmp(p + 2, t + j + 2, m - 2) = = 0 &&  p[0] = = t[j])
   cout << "Pattern is found at position " << j << endl;
```

Είτε ταιριάζει το πρότυπο στους χαρακτήρες του παραθύρου είτε όχι, σε αυτό το σημείο ο δείκτης j αυξάνει κατά ell. Όπως προαναφέραμε, το ell είναι 2 στην περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου δεν ταυτίζονται. Αυτό γίνεται ώστε να γλιτώσουμε τον σίγουρα αποτυχημένο έλεγχο του πρώτου χαρακτήρα του προτύπου με τον αμέσως επόμενο χαρακτήρα του παραθύρου, διότι αυτός ταυτιζόταν στο προηγούμενο βήμα με τον χαρακτήρα στη δεύτερη θέση του προτύπου, ο οποίος είναι διαφορετικός από αυτόν στην πρώτη θέση. Αυτή η σύγκριση παρακάμπτεται, με κέρδος σε υπολογιστικούς πόρους. Αν ο πρώτος με τον δεύτερο χαρακτήρα του προτύπου ταυτίζονται, το ell είναι 1 και σε αυτή την περίπτωση ο δείκτης j αυξάνει κατά 1.

```
    j += ell;
   }
 system("PAUSE");
 return 0;
}
```

5.11.4. Αλγόριθμος Quick Search

Ο αλγόριθμος Quick Search αναζητά ένα αλφαριθμητικό πρότυπο (pattern) μέσα σε κείμενο (text), χρησιμοποιώντας bad character shift, ενώ η σειρά με την οποία εκτελούνται οι συγκρίσεις των χαρακτήρων των αλφαριθμητικών είναι αδιάφορη.

```
#include <iostream>

#include <string>

using namespace std;
```

Το μέγεθος του αλφαβήτου είναι 256, όσοι και οι χαρακτήρες του κώδικα ASCII. Η μεταβλητή ASIZE ισούται με το μέγεθος του αλφαβήτου μείον 1.

```
int ASIZE=255;
```

Ακολουθεί η πρόσθια δήλωση (forward declaration) των συναρτήσεων preQsBc και QS, όπως απαιτεί η C++ για συναρτήσεις που χρησιμοποιούνται ενώ ο κώδικάς τους ακολουθεί.

```
void preQsBc(char *x, int m, int qsBc[]);

void QS(char *x, int m, char *y, int n);

int main ()

{
```

p είναι ο πίνακας χαρακτήρων που κρατά το πρότυπο, ενώ t είναι ο πίνακας χαρακτήρων που κρατά το κείμενο. plen και tlen τα μήκη των αντίστοιχων αλφαριθμητικών

```
  char p[50]="";

  char t[250]="";
```

```
  int plen, tlen;
  cout << "Enter the pattern: " << endl;
  cin. getline(p, 50);
  cout << "Enter the text: " << endl;
  cin. getline(t, 250);
```

Με τη δομή επανάληψης for υπολογίζεται το μήκος του προτύπου, καθώς και του κειμένου. Ουσιαστικά, δεν εκτελείται καμία εντολή εντός της δομής, απλά πετυχαίνουμε σε κάθε επανάληψη να αυξάνεται η μεταβλητή μετρητής (plen, tlen), η οποία στο τέλος καταλήγει να έχει τιμή ίση με το μήκος του αλφαριθμητικό που είναι αποθηκευμένο σε πίνακα χαρακτήρων.

```
  for(plen=0;p[plen]!='\0';plen++) ;
  for(tlen=0;t[tlen]!='\0';tlen++) ;
```

Καλείται η συνάρτηση QS, με ορίσματα τα αλφαριθμητικά και τα μήκη τους.

```
  QS(p, plen, t, tlen);
  system("PAUSE");
  return 0;
}
```

Στη φάση της προ επεξεργασίας δημιουργείται ο πίνακας bad character shift (qsBc) που έχει μέγεθος όσα και τα γράμματα του αλφαβήτου μας, δηλαδή 256. Σε κάθε θέση του φαίνεται πόση θα είναι η μετατόπιση του παραθύρου σε κάθε νέα απόπειρα εύρεσης του προτύπου στο κείμενο.

Η συνάρτηση preQsBc γεμίζει τον πίνακα qsBc ως εξής: αν ένα γράμμα του αλφαβήτου δεν υπάρχει στο πρότυπο, αντιστοιχίζεται σε αυτό τιμή για shift η m+1.

```
void preQsBc(char *x,  int m,  int *qsBc) {
```

```
  for (int i = 0; i < ASIZE; i++)
    qsBc[i] = m + 1;
```

Όταν ένα γράμμα του αλφαβήτου υπάρχει στο πρότυπο, αντιστοιχίζεται η μικρότερη τιμή του i για την οποία ισχύει ότι x[m-1-i]=c και 0<=i<=m-1 όπου x το πρότυπο και c το γράμμα. Ουσιαστικά, βρίσκεται η δεξιότερη εμφάνιση κάθε χαρακτήρα x[i] του προτύπου, πχ αν x[i]='a', επειδή ο κωδικός του σε ASCII είναι 97, το qsBc[x[i]] ισοδυναμεί με qsBc[97]. Και αφού η θέση του χαρακτήρα στο πρότυπο είναι I, στη θέση x[i] του qsBc μπαίνει η τιμή m-i.

```
  for (int i = 0; i < m; i++)
    qsBc[x[i]] = m - i;
]
```

Η συνάρτηση QS επιτελεί το ρόλο της αναζήτησης του προτύπου στο κείμενο.

```
void QS(char *x,  int m,  char *y,  int n) {
  int j;
```

Δημιουργείται ο πίνακας qsBc.

```
  int qsBc[ASIZE];
```

Καλείται η συνάρτηση preQsBc, ώστε να γεμίσει ο πίνακας qsBc

```
preQsBc(x,  m,  qsBc);
```

Ο δείκτης j που δείχνει στο κείμενο αρχικά είναι 0 και ακολουθούν συγκρίσεις του προτύπου με το κείμενο. Ο βρόγχος while επαναλαμβάνεται τόσες φορές όσες το μέγεθος του κυλιόμενου παραθύρου είναι μεγάλο σε μήκος όσο και το πρότυπο.

```
  j = 0;
  while (j <= n - m) {
```

Η συνάρτηση memcmp συγκρίνει τα δύο αλφαριθμητικά. Αν αυτά ταυτίζονται, η συνάρτηση επιστρέφει 0 και εμφανίζεται μήνυμα στο χρήστη που δηλώνει σε ποια θέση βρέθηκε το πρότυπο.

```
    if (memcmp(x,  y + j,  m) == 0)

      cout << "found pattern at position " << j << endl;
```

Ο δείκτης j μετατοπίζεται δεξιά τόσες θέσεις όσες η τιμή που έχει στον πίνακα qsBc ο χαρακτήρας που συγκρίθηκε. Αν ο χαρακτήρας υπάρχει μια φορά στην πρώτη θέση στο πρότυπο, θα γίνουν m μετατοπίσεις. Όταν ο χαρακτήρας y[j+m] δεν υπάρχει στο πρότυπο, γίνονται m+1 μετακινήσεις.

```
    j += qsBc[y[j + m]];

  }

}
```

6. Η γλώσσα προγραμματισμού Java

Η Java αναπτύχθηκε το 1990 από το μηχανικό της Sun James Gosling για να χρησιμοποιηθεί στους εγκεφάλους έξυπνων συσκευών (διαλογικές τηλεοράσεις, «έξυπνες» ηλεκτρικές κουζίνες και άλλα). Ο Gosling δεν ήταν ευχαριστημένος με τα αποτελέσματα της δημιουργίας προγραμμάτων C + +, μιας άλλης γλώσσας προγραμματισμού και έτσι κλείστηκε στο γραφείο του και έφτιαξε μια καινούργια γλώσσα που κάλυπτε καλύτερα τις ανάγκες του. Εκείνη την εποχή, ο Gosling ονόμασε τη γλώσσα Oak (βελανιδιά), γιατί ήταν το δέντρο που μπορούσε να δει από το παράθυρο του γραφείου του. Η γλώσσα αποτέλεσε μέρος της στρατηγικής της Sun για κέρδη εκατομμυρίων, όταν η διαλογική τηλεόραση έγινε βιομηχανία πολλών εκατομμυρίων. Αυτό δεν έχει συμβεί ακόμα σήμερα (αν και στην Αμερική έγιναν κάποιες προσπάθειες από τις TiVo, Replay-TV και WebTV), αλλά με τη νέα γλώσσα του Gosling συνέβη κάτι εντελώς διαφορετικό. Ενώ η Sun ήταν έτοιμη να απορρίψει την Oak και να μεταφέρει τους ανθρώπους που εργάζονταν πάνω σε αυτή σε άλλα τμήματα της εταιρείας, έγινε δημοφιλές το World Wide Web. Από ευτυχή σύμπτωση, πολλές από τις ιδιότητες που έκαναν τη γλώσσα του Gosling κατάλληλη για τις ηλεκτρικές συσκευές, την έκαναν κατάλληλη και για προσαρμογή στο World Wide Web. Οι προγραμματιστές της Sun επινόησαν ένα τρόπο ασφαλούς εκτέλεσης των προγραμμάτων των ιστοσελίδων και επέλεξαν ένα εμπορικότερο όνομα, για το καινούργιο πεδίο εφαρμογής της γλώσσας: την ονόμασαν Java. Αν και η Java μπορεί να χρησιμοποιηθεί για πολλά άλλα πράγματα, το Web παρείχε τη βιτρίνα που χρειαζόταν για να τραβήξει την παγκόσμια προσοχή. Ένας προγραμματιστής που βάζει ένα πρόγραμμα της Java σε μια ιστοσελίδα, το κάνει αυτόματα διαθέσιμο σε όλο τον κόσμο του Web. Επειδή η Java ήταν η πρώτη τεχνολογία που μπορούσε να προσφέρει αυτή τη δυνατότητα, έγινε η πρώτη γλώσσα προγραμματισμού την οποία ανέδειξαν πανηγυρικά τα ΜΜΕ. Όταν η γλώσσα ανέβηκε στην κορυφή το 1996, θα έπρεπε να βρίσκεται κανείς σε απομόνωση ή σε διαστημική αποστολή για να μην ακούσει για την Java.

Μπορεί να έχετε ακούσει ότι το όνομα Java είναι αρχικά των λέξεων Just Another Vague Acronym (άλλο ένα ασαφές αρκτικόλεξο). Μπορεί επίσης να έχετε ακούσει ότι το όνομα προήλθε από τη μεγάλη αγάπη του δημιουργού της για τον καφέ (java). Στην πραγματικότητα, η ιστορία πίσω από την ονομασία της Java δεν περιέχει κάποιο μυστικό και δεν αποτελεί έκφραση αγάπης για τον καφέ. Αντίθετα, το όνομα Java επιλέχθηκε γιατί άρεσε στους δημιουργούς της.

Υπάρχουν 5 βασικές εκδόσεις της Java:

- Φθινόπωρο 1995: Java 1. 0 – Μια έκδοση που ήταν ιδανική για χρήση στο World Wide Web και που είχε δυνατότητες επέκτασης σε άλλα είδη προγραμματισμού.
- Άνοιξη 1997: Java 1. 1 – Μια αναβάθμιση της γλώσσας που περιλάμβανε πολλές βελτιώσεις στον τρόπο δημιουργίας και χειρισμού των περιβαλλόντων εργασίας χρήστη.
- Καλοκαίρι/Φθινόπωρο 1998: Java 2, έκδοση 1. 2 –Μια έκδοση τρεις και πλέον φορές μεγαλύτερη από την Java 1. 0 με βελτιώσεις που έκαναν τη γλώσσα επάξιο ανταγωνιστή άλλων γλωσσών προγραμματισμού γενικής χρήσης.
- Φθινόπωρο 2000: Java 2, έκδοση 1. 3 –Μια έκδοση που υποστηρίζει πιο γρήγορη εκτέλεση των προγραμμάτων και βελτιωμένα στοιχεία πολυμέσων, καθώς και την πρώτη επίσημη υποστήριξη για την ανάπτυξη της Java στο λειτουργικό σύστημα Linux.
- Άνοιξη 2002: Java 2, έκδοση 1. 4 –Μια ουσιαστική αναβάθμιση με ένα στοιχείο που είχε ζητηθεί επανειλημμένα και ονομάζεται assertion, εκτεταμένη υποστήριξη δικτύου και επεξεργασία XML.

<u>Χαρακτηριστικά της Java</u>

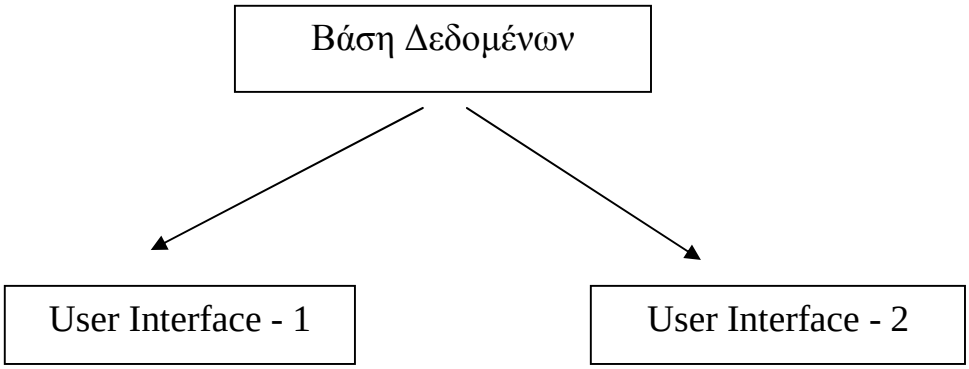
- Απλή
- Αντικειμενοστραφής
- Μεταφέρσιμη (portable) (και σε επίπεδο μεταγλωττισμένου κώδικα)

- Ανιχνεύει προβλήματα κατά την εκτέλεση και εμφανίζει κατανοητά μηνύματα
- Υποστηρίζει κατανεμημένο προγραμματισμό (distributed programming)
- Επιτρέπει την υλοποίηση ασφαλών προγραμμάτων
- Υποστηρίζει πολυνηματική επεξεργασία (multithreaded processing)
- Αυτόματη αποκομιδή αχρήστων (garbage collection)
- Κατάλληλη για εφαρμογές στο Internet στον πελάτη (applets) και στον εξυπηρετητή (servlets)

Ένα πρόγραμμα δεν είναι παρά ένα ή περισσότερα αρχεία κώδικα, δηλαδή αρχεία με εντολές για τις ενέργειες που πρέπει να πραγματοποιήσει η εφαρμογή. Αυτό περνά από ένα μεταγλωττιστή ο οποίος διαβάζει τον κώδικα και παράγει ένα εκτελέσιμο πρόγραμμα (εφαρμογή) το όποιο περιλαμβάνει τις εντολές που περιέχονται στον κώδικα σε μια μορφή ώστε να μπορεί να τις εκτελέσει άμεσα ο υπολογιστής (ουσιαστικά όλες οι εντολές μετατρέπονται για οδηγίες για μαθηματικές πράξεις και μετακίνηση δεδομένων από/προς τη μνήμη). Όπως και κάθε γλώσσα προγραμματισμού, έτσι και η Java ακολουθεί κάποιους κανόνες στη σύνταξη και υλοποίηση ενός προγράμματος. Οι κανόνες αυτοί αφορούν τη δήλωση μεταβλητών, τους τύπους δεδομένων, τους ορισμούς κλάσεων κλπ.

6.1. Αντικεμενοστραφής Προγραμματισμός

Ο αντικειμενοστραφής προγραμματισμός, (OOP - Object Oriented Programming),
είναι μια προγραμματιστική φιλοσοφία όπως και ο προστακτικός ή ο λογικός προγραμματισμός. Σύμφωνα με τον OOP ένα πρόγραμμα ΔΕΝ αποτελείται από τα δεδομένα και τον κώδικα που τα επεξεργάζεται αλλά από αντικείμενα (objects) τα οποία εμπεριέχουν τα δεδομένα και τα οποία (αντικείμενα) ανταλλάσσουν μεταξύ τους πληροφορίες και μηνύματα προκειμένου να επιτευχθεί ο στόχος του προγράμματος.
Για παράδειγμα έστω ότι έχουμε ένα σύστημα που περιλαμβάνει μια βάση δεδομένων καθώς και user interfaces με τα οποία οι χρήστες επικοινωνούν και αλληλεπιδρούν με την Β.Δ..



Τα UIs στέλνουν τις εντολές των χρηστών στην ΒΔ και παρουσιάζουν στην οθόνη τις
απαντήσεις που λαμβάνουν. Δηλαδή όλο το πρόγραμμα αποτελείται από ένα αντικείμενο ΒΔ και 2 αντικείμενα UI τα οποία είναι ολόιδια (αλλά εξυπηρετούν άλλους χρήστες). Συνεπώς ο κώδικας που θα γράφαμε θα περιείχε τον ορισμό ενός αντικειμένου ΒΔ, τον ορισμό ενός αντικειμένου UI και την κατασκευή ενός ΒΔ και 2 UI αντικειμένων.

Ορολογία

Ο κώδικας που ορίζει ένα αντικείμενο λέγεται κλάση (class) του αντικειμένου αυτού.

Η κλάση χρησιμοποιείται σαν μήτρα για την κατασκευή πανομοιότυπων αντιγράφων -αντικειμένων. Τα αντικείμενα - αντίγραφα, λέγονται στιγμιότυπα (instances) της κλάσης αυτής και η κατασκευή και αρχικοποίηση ενός από αυτά λέγεται instantiation.

Το αντικείμενο αυτό καθ' εαυτό είναι μια οντότητα στη μνήμη η οποία περιέχει δεδομένα καθώς και μεθόδους μέσω των οποίων μπορούμε να αλλάξουμε τα δεδομένα ή ναεπικοινωνήσουμε με το αντικείμενο. Αντίθετα η κλάση είναι απλώς ένα πρότυπο για τηνδημιουργία αντιγράφων του ίδιου αντικειμένου. Είναι δηλαδή ότι είναι ο Τύπος Δεδομένων για τις μεταβλητές (στον δομημένο προγραμματισμό).

6.2. Αλφάβητο

Τελεστές (operators)

Μπορούμε να εκτελέσουμε διάφορες πράξεις μεταξύ των μεταβλητών με τη χρήση των τελεστών. Οι τελεστές είναι σύμβολα που αντιστοιχούν σε αριθμητικές ή λογικές πράξεις μεταξύ των αντικειμένων. Υπάρχουν 4 είδη τελεστών στη Java

Αριθμητικοί τελεστές

Τελεστής	Περιγραφή
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο
^	΄Υψωση σε δύναμη

Ειδικά για τους αυξητικούς και αφαιρετικούς τελεστές, πρέπει να αναφέρουμε ότι αυτοί αυξάνουν ή αφαιρούν την τιμή της μεταβλητής κατά μία μονάδα (κατά συνέπεια χρησιμοποιούνται μόνο σε ακέραιες μεταβλητές, όχι σε δεκαδικές). Παράδειγμα:

```
int x = 10;

x++; // τώρα η τιμή x έχει την τιμή 11

x--; // και πάλι την τιμή 10
```

Σχεσιακοί τελεστές

Οι σχεσιακοί τελεστές αναφέρονται στην σχέση μεταξύ δύο αντικειμένων.

Τελεστής	Περιγραφή
!=	Άνισο με
==	Ίσο με ==
<=	Μικρότερο ή ίσο
<	Μικρότερο
>=	Μεγαλύτερο ή ίσο
>	Μεγαλύτερο

Λογικοί τελεστές

Οι λογικοί τελεστές αναφέρονται στους τρόπους συνδυασμών αληθών και ψευδών προτάσεων.

Τελεστής	Περιγραφή
&	ΚΑΙ (AND)
|	Η (OR)
!	ΟΧΙ (NOT)
&&	Short-circuit AND
||	Short-circuit OR
^	XOR (ή το ένα ή το άλλο)

Η διαφορά του & και του && είναι ότι στη δεύτερη περίπτωση, αν με τον υπολογισμό της πρώτης παράστασης ο συνδυασμός βγαίνει αληθής ή ψευδής τότε αποφεύγεται ο υπολογισμός της δεύτερης.

Για παράδειγμα, αν έχουμε την παράσταση

(alpha == true) & (beta == 1)

το πρόγραμμα θα υπολογίσει και θα εξετάσει και τις δύο προτάσεις ακόμη και αν η πρώτη (alpha == true) είναι ψευδής. Με τη χρήση του &&, το πρόγραμμα θα επιστρέψει αμέσως ψευδή τιμή (false) αν η πρώτη είναι ψευδής, και θα προχωρήσει στον υπολογισμό της δεύτερης μόνο αν η πρώτη είναι αληθής. Στην πραγματικότητα είναι θέμα εξοικονόμησης χρόνου. Οι παραπάνω τελεστές χρησιμοποιούνται για συγκρίσεις μεταξύ αριθμών. Εάν η σύγκριση είναι αληθής τότε το αποτέλεσμα είναι η τιμή true διαφορετικά εάν είναι ψευδής τότε το αποτέλεσμα είναι η τιμή false.

Τελεστής αντιστοίχησης

Τελεστής	Περιγραφή
=	Τελεστής αντιστοίχησης

Τελεστής αύξησης και τελεστής μείωσης

Τελεστής	Περιγραφή
++	αύξηση κατά 1
--	μείωση κατά 1

Οι τελεστές ++ και -- χρησιμοποιούνται όταν θέλουμε να προσθέσουμε ή να αφαιρέσουμε το 1 από μία μεταβλητή. Έτσι

το ++a; ισοδυναμεί με το a=a+1;

ενώ το --a; ισοδυναμεί με το a=a-1;

Οι τελεστές ++ και -- μπορούν να χρησιμοποιηθούν είτε ως προθεματικοί τελεστές (δηλ. πριν την μεταβλητή, πχ ++a ή--a) είτε ως μεταθεματικοί (δηλ. μετά την μεταβλητή, πχ a++ ή a--).

Στην παράσταση ++a η τιμή του a αυξάνει πριν χρησιμοποιηθεί η τιμή της.

Στην παράσταση a++ η τιμή του a αυξάνει αφού χρησιμοποιηθεί η τιμή της.

Τελεστές καταχώρησης

Οι τελεστές καταχώρησης χρησιμοποιούνται για να μεταβάλουν τα περιεχόμενα μιας μεταβλητής. Μπορούν να συνδυαστούν και με άλλους τελεστές για συντόμευση κάποιων πράξεων.

Παραδείγματα τελεστών καταχώρησης:

int x = 4;

x = 10;

x += 20; (είναι το ίδιο με την εντολή x = x + 20, τελικό αποτέλεσμα 30)

x /= 10; (το ίδιο με x = x / 10, αποτέλεσμα 3).

6.3. Λεξιλόγιο

Μεταβλητές και Σταθερές

Η έννοια της μεταβλητής είναι ίσως η πιο θεμελιώδης στον προγραμματισμό γενικά. Η μεταβλητή δεν είναι άλλο από μια ονομασμένη θέση μνήμης που μπορεί να περιέχει δεδομένα οποιουδήποτε είδους. Το περιεχόμενο της μεταβλητής μπορεί να μεταβάλλεται με δεδομένα παρόμοιου είδους.

Οι μεταβλητές ορίζονται ως εξής:

type var;

όπου type το είδος της μεταβλητής και var το όνομά της. Ως είδος μπορούμε να διαλέξουμε έναν από τους διαθέσιμους τύπους δεδομένων της Java ή το όνομα μιας κλάσης αντικειμένων.

Εύρος μεταβλητών

Μια μεταβλητή θεωρείται έγκυρη, δηλαδή μια αναφορά σε αυτήν έχει νόημα μόνο μέσα στο πλαίσιο στο οποίο ορίζεται. Το πλαίσιο μπορεί να είναι μια κλάση, μια συνάρτηση ή μέθοδος, το πεδίο ισχύος ενός loop for/while/do ή των εντολών if/else/switch.

Τύποι δεδομένων

Οι βασικοί τύποι δεδομένων στη Java είναι σχεδόν οι ίδιοι με αυτούς της C και C++ (καθώς η Java έχει βασιστεί σε μεγάλο βαθμό στη C++). Πέρα από μικρές διαφορές στους ίδιους τους τύπους, είναι σχετικά εύκολο για κάποιον προγραμματιστή C++ να μεταβεί στη Java και να κρατήσει παρόμοια τη δομή ενός προγράμματος και ένας από τους λόγους που γίνεται αυτό είναι η ομοιότητα στους τύπους δεδομένων.

Ακολουθεί πίνακας με τους διαθέσιμους τύπους δεδομένων της Java:

Όνομα	Μέγεθος (σε bytes)	Όρια
byte	1	-128 έως 127
short	2	-32768 έως 32767
int	4	-2147483648 έως -2147483647
long	8	-9223372036854775808 έως -9223372036854775807
float	4	1. 4 * 10-45 έως 3. 4 * 1038
double	8	4. 9 * 10-324 έως 1. 8 * 10308
bool	1	true/false
char	2	-
String	μεταβλητό	-

Οι τύποι δεδομένων byte, short, int, long προορίζονται για χρήση με ακέραιους αριθμούς, ενώ οι τύποι float, double για χρήση αριθμών κινητής υποδιαστολής (δηλ. δεκαδικών αριθμών). Ο τύπος bool μπορεί να λαμβάνει μόνο δύο τιμές true και false. Ο τύπος char σε αντίθεση με την C/C++ έχει μέγεθος 2 bytes γιατί έχει σχεδιαστεί να περιέχει χαρακτήρες Unicode.

Παραδείγματα δηλώσεων και αρχικοποιήσεων μεταβλητών:

```
int an_integer;

an_integer = 10;
```

long a_long = an_integer *1000;

double verysmallnumber = 0. 000000000003;

bool am_i_hungry = false;

char alpha = 'a';

String text = “this is a text”;

Σταθερές (constants)

Αν και οι μεταβλητές έχουν σχεδιαστεί να μεταβάλλονται υπάρχουν περιπτώσεις που χρειαζόμαστε κάποιες σταθερές, όπως στα Μαθηματικά. Μπορούμε να δηλώσουμε μια μεταβλητή ως σταθερά (δηλαδή που να μη μεταβάλλεται) με τη λέξη final. Δηλαδή:

final double pi = 3. 1415;

6.4. Δομές Επιλογής και Επανάληψης

Οι εντολές στις δομές επιλογής και επανάληψης είναι σχεδόν πανομοιότυπες με αυτές της C, με μόνη εξαίρεση ότι στη JAVA μπορούμε να κάνουμε δηλώσεις μεταβλητών όχι μόνο στην αρχή ενός block αλλά και σε οποιοδήποτε σημείο του, αρκεί η δήλωση να γίνει πριν από τη χρήση της εν λόγω μεταβλητής. Να σημειώσουμε επίσης ότι ισχύουν όλοι οι κανόνες εμβέλειας, τοπικών μεταβλητών και περάσματος παραμέτρων (σε μεθόδους) της C.
Όσον, τώρα, αφορά τις δομές ροής ελέγχου (if - else, for, while, do - while και switch), οι μόνες αλλαγές που έχουν γίνει σε σχέση με τη C είναι το ότι στο if-else, while, do-while οι εκφράσεις που αποτιμώνται προκειμένου να γίνει άλμα ή επανάληψη, θα πρέπει να είναι boolean και όχι αριθμητικές. Δηλαδή :
while (1) {...} // ΕΙΝΑΙ ΛΑΘΟΣ
while (true) {....} // ΕΙΝΑΙ ΣΩΣΤΟ

Στη συνέχεια θα παρουσιάσουμε συνοπτικά τις δομές ροής ελέγχου προγράμματος της Java. Υπάρχουν γενικά δύο ήδη δομών ελέγχου ροής (control flow):
• Οι δομές επιλογής και
• Οι δομές επανάληψης
Ο ακόλουθος πίνακας συνοψίζει αυτές τις δομές για τη Java

Είδος δομής ελέγχου ροής	Δομή ελέγχου ροής
Δομές επανάληψης	if-else
	switch-case
Δομες επανάληψης	for

	while
	do-while

Πέρα από τις εντολές του πιο πάνω πίνακα ο έλεγχος ροής σε ένα πρόγραμμα Java μπορεί να μεταφερθεί και σε κάποιο άλλο σημείο εξαιτίας της πρόκλησης μιας εξαίρεσης. Αλλά για τις εξαιρέσεις (exceptions) και τους χειριστές τους (exception handlers) θα μιλήσουμε σε επόμενα μαθήματα.

Επίσης κάποιες άλλες εντολές πέρα από τις ίδιες τις δομές ελέγχου ροής που είδαμε προηγουμένως είναι οι: break, continue και return που μεταφέρουν το έλεγχο ροής σε άλλα σημεία του προγράμματος. Την χρήση αυτών των εντολών θα τη δούμε στη συνέχεια.

Η Java επιτρέπει και τη χρήση ετικετών αλλά δεν υποστηρίζει την εντολή goto. Αντί αυτής μπορούν και με τις ετικέτες να χρησιμοποιηθούν οι εντολές break και continue.

Η δομή επιλογής if-else

Η εντολή if μας βοηθάει στον έλεγχο μίας λογικής έκφρασης (της συνθήκης) και αν είναι αληθής εκτελούνται μία ή περισσότερες εντολές ενώ αν είναι ψευδής δεν εκτελούνται. Αν οι εντολές είναι περισσότερες από μία τότε θα πρέπει οι εντολές να περικλειστούν ανάμεσα από άγκιστρα (τα οποία ομαδοποιούν εντολές).

Το συντακτικό της εντολής if είναι το ακόλουθο:

```
if (συνθήκη)
εντολές
```

π.χ.

```
if (x!=0) {
System.out.println("Το x δεν είναι 0");
y = 1/x;
}
```

Η εντολή if έχει επίσης και τη φράση else με την οποία επιτρέπεται η εκτέλεση μίας ομάδας εντολών εναλλακτικά αν δεν ισχύει η συνθήκη. Φυσικά μπορούμε να έχουμε και εμφωλευμένα if δηλαδή if μέσα σε άλλα if όσες φορές θέλουμε.
Το συντακτικό της if επαυξημένο με τη φράση else είναι το ακόλουθο:

```
if (συνθήκη)
ομάδα-εντολών-1
else
ομάδα-εντολών-2
```

π.χ.

```
if (x!=0) {
System.out.println("Το x δεν είναι 0");
y=1/x;
}
else {
System.out.println("Το x είναι 0");
y=0;
}
```

Η δομή επιλογής switch-case

Όταν οι εναλλακτικές περιπτώσεις που πρέπει να ελέγξουμε με την if είναι πάρα πολλές και αφορούν τον έλεγχο για ισότητα της τιμής μιας μεταβλητής ή μιας έκφρασης με κάποιες τιμές προτιμάται η switch-case η οποία έχει την ακόλουθη γενική μορφή:

```
switch (έκφραση) {
case τιμή-1:
εντολές-1; [break;]
…
case τιμή-ν:
εντολές-ν; [break;]
[default:
εντολές; [break;]]
}
```

Η switch-case αποτιμά την τιμή της έκφρασης που ελέγχεται και στη συνέχεια διατρέχει με τη σειρά όλες τις περιπτώσεις που δίνονται. Αν κάποια περίπτωση βρεθεί αληθής τότε εκτελούνται οι εντολές που δίνονται μετά την άνω-κάτω τελεία γι' αυτή τη περίπτωση. Αν καμία περίπτωση δεν βρεθεί αληθής τότε εκτελείται η default περίπτωση αν υπάρχει.

Προσοχή χρειάζεται στη χρήση του break που είναι μεν προαιρετική αλλά απαιτείται τις περισσότερες φορές, μια και αν δεν υπάρχει τότε θα εκτελεστούν και οι εντολές τις επόμενης περίπτωσης (χωρίς να ελεγχθεί η τιμή της) και πιθανώς και άλλων, μέχρι να βρεθεί το επόμενο break ή να τελειώσει η switch-case.

Παρόλα αυτά ενδέχεται να υπάρχουν κάποιες περιπτώσεις που αυτό θα ήταν βολικό, όπως δείχνει το ακόλουθο τμήμα κώδικα που υπολογίζει τις μέρες ενός μήνα ανάλογα με το ποιος μήνας είναι (η τιμή της μεταβλητής month) και το αν το έτος (year) είναι δίσεκτο:

```
...
switch (month) {
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12:
numDays = 31;
break;
case 4:
case 6:
case 9:
case 11:
numDays = 30;
break;
case 2:
```

```
if ( ((year % 4 == 0) && !(year % 100 == 0))
|| (year % 400 == 0) )
numDays = 29;
else
numDays = 28;
break;
}
...
```

Η δομή επανάληψης for

Η εντολή for είναι χρήσιμη για την επανάληψη μιας σειράς εντολών όταν είναι γνωστό εκ των προτέρων πόσες φορές θέλουμε να επαναληφθούν. Έτσι συνήθως το for ελέγχεται από ένα μετρητή ο οποίος μεταβάλλεται από μία αρχική τιμή μέχρι να ξεπεράσει μία τελική τιμή. Στη for επίσης καθορίζεται το πόσο θα μεταβάλλεται αυτός ο μετρητής σε κάθε βήμα.

Η γενική μορφή της for είναι η ακόλουθη:

```
for (αρχικοποίηση; τερματισμός; αύξηση)
εντολές;
```

Για παράδειγμα το ακόλουθο τμήμα κώδικα εμφανίζει στην οθόνη τους αριθμούς από το 1 μέχρι το 10:

```
...
for (int i=1; i<=10; i++)
System.out.println(i);
...
```

Στη φράση της αρχικοποίησης το i δηλώνεται (η εμβέλειά του είναι το for loop) και αρχικοποιείται στο1.

Στη φράση του τερματισμού το i ελέγχεται για το αν έχει ξεπεράσει το 10. Άρα ο συγκεκριμένος βρόχος θα επαναληφθεί μέχρι το i να ξεπεράσει το 10.

Στη φράση της μεταβολής το i αυξάνεται κατά 1. Αυτό σημαίνει ότι το i θα πάρει διαδοχικά τις τιμές 1, 2, …, 10

Σε κάθε βήμα της επανάληψης ελέγχεται η τιμή του i. Αν το i ξεπεράσει το 10 ο βρόχος τερματίζεται, αν όχι αυξάνεται κατά 1 και εκτελείται ξανά η μία και μοναδική εντολή αυτού του βρόχου.

Η δομή επανάληψης while

Η εντολή for που είδαμε προηγουμένως είναι κατάλληλη όταν γνωρίζουμε πόσες επαναλήψεις θα γίνουν. Αν δεν γνωρίζουμε πόσες επαναλήψεις θα γίνουν μία καλύτερη εντολή είναι η while.

Η while έχει την ακόλουθη γενική μορφή:

```
while (συνθήκη)
εντολές;
```

Η εντολή ή εντολές που ακολουθούν το while θα εκτελεστούν όσο η συνθήκη είναι αληθής. Ο βρόχος δηλαδή θα τερματιστεί όταν η συνθήκη - που είναι μία boolean έκφραση - γίνει ψευδής

Στο ακόλουθο τμήμα κώδικα ελέγχεται αν η τιμή x βρίσκεται μέσα στο πίνακα numbers. Κάνουμε δηλαδή μία σειριακή αναζήτηση στα στοιχεία του πίνακα. Αν το στοιχείο βρεθεί η μεταβλητή found γίνεται true και βγαίνουμε από το βρόχο. Επίσης από το βρόχο θα βγούμε αν ελεγχθούν και τα δέκα στοιχεία του πίνακα και δεν έχει βρεθεί ακόμα το x.

```
…
boolean found = false;
int i=0;
while (!found && i!=10)
if (numbers[i++] == x)
found = true;
…
```

Η δομή επανάληψης do-while

Υπάρχουν κάποιες περιπτώσεις στις οποίες θα θέλαμε οι εντολές μέσα στο βρόχο να εκτελεστούν τουλάχιστον μία φορά και στη συνέχεια να ελεγχθεί η

συνθήκη εξόδου. Σ’ αυτές τις περιπτώσεις προτιμάται η χρήση της do-while αντί της while.

Η do-while έχει την ακόλουθη γενική μορφή:

```
do
εντολές;
while (συνθήκη);
```

Στην do-while πρώτα εκτελούνται οι εντολές και στη συνέχεια ελέγχεται η συνθήκη. Ο βρόχος τερματίζεται αν η συνθήκη βρεθεί ψευδής.

Η do-while δεν χρησιμοποιείται πολύ συχνά αλλά έχει κι αυτή τις χρήσεις της.

Για παράδειγμα όταν διαβάζουμε χαρακτήρες από ένα αρχείο μέχρι να διαπιστώσουμε το τέλος του αρχείου θα πρέπει να διαβάσουμε τουλάχιστον ένα χαρακτήρα, όπως δείχνει και το ακόλουθο τμήμα κώδικα:

```
...
int c;
Reader in;
...
do {
c = in.read();
...
} while (c != 1);
...
```

<u>Εντολές διακλάδωσης</u>

Η Java έχει τρεις εντολές διακλάδωσης οι οποίες είναι βολικές σε αρκετές περιπτώσεις και ιδιαίτερα με τις επαναληπτικές δομές που έχουμε συζητήσει.

Οι εντολές διακλάδωσης είναι οι: break, continue και return

Την εντολή break ήδη την είδαμε σε μία χρήση της με την switch-case. Μία ακόμα αρκετά συνήθη χρήση της break είναι η χρήση της για άμεση έξοδο από κάποιο βρόχο.

Για παράδειγμα η σειριακή αναζήτηση που είδαμε με το while θα μπορούσε να γίνει όπως στο For.java ως εξής:

```
int i;
```

```
boolean found = false;
for (i=0; i<10; i++)
if (numbers[i]==x) {
found = true;
break;
}
```

Η χρήση της continue μέσα σε ένα βρόχο προκαλεί την άμεση αποτίμηση και πάλι της συνθήκης εξόδου. Για να γίνει αυτό στις εντολές for και while ο έλεγχος μεταφέρεται στη πρώτη γραμμή του βρόχου ενώ στο do-while στη τελευταία. Στη συνέχεια αποτιμάται και πάλι η συνθήκη τερματισμού και η επαναληπτική διαδικασία τερματίζεται ή συνεχίζεται ανάλογα με την τιμή της συνθήκης (true ή false) ως συνήθως.

Οι εντολές break και continue έχουν και μία μορφή για ετικέτες (labeled break και labeled continue) που δεν θα συζητήσουμε εδώ μια και στη πράξη οι ετικέτες δεν χρησιμοποιούνται αφού οι υπόλοιπες δομές που είδαμε αρκούν για όλες τις πιθανές χρήσεις, χωρίς να εμφανίζουν πολλά από τα προβλήματα που πιθανώς να εμφανιστούν με την χρήση των ετικετών.

Η εντολή return διακόπτει αμέσως την εκτέλεση της μεθόδου μέσα στην οποία βρίσκετε και σε περίπτωση που η μέθοδος επιστρέφει κάποια τιμή, επιστρέφει την τιμή που βρίσκεται στα δεξιά της. Ο τύπος της επιστρεφόμενης τιμής πρέπει να είναι ίδιος με τον τύπο-αποτελέσματος που δηλώθηκε στην δήλωση της μεθόδου.

Στην ακόλουθη μέθοδο η εντολή return επιστρέφει ένα String που συγκεκριμένα την τιμή που περιέχεται στην μεταβλητή name. Η εκτέλεση της μεθόδου getName( ) τερματίζει ακόμα και αν υπάρχουν και άλλες εντολές κάτω από την εντολή: return name;

```
public String getName()
{
...
return name;
}
```

Στην παρακάτω μέθοδο η εντολή: return interest; τερματίζει την μέθοδο και επιστρέφει τον ακέραιο που περιέχει η μεταβλητή interest;

```
public int calculateInterest( )
{
... return interest;
}
```

6.5. Μέθοδοι (Methods)

Οι τύποι δεδομένων δεν είναι πολύ εύχρηστοι αν δεν μπορείς να κάνεις πράγματα με αυτούς, Γι’ αυτό το σκοπό οι κλάσεις έχουν μεθόδους. Τα μέλη λένε τι είναι μια κλάση. Οι μέθοδοι λένε τι κάνει μια κλάση. Π.χ. η κλάση του website ίσως έχει μια μέθοδο για να τυπώσει τα δεδομένα του. Εάν είναι έτσι αυτό θα μοιάζει όπως παρακάτω:

```
class website {

  String name;
  String url;
  String description;

  print() {
    System.out.println(name + " at " + url + " is " + description);
  }

}
```

Έξω από τη μέθοδο website καλούμε τη μέθοδο print, ακριβώς όπως αναφερόμασταν στις μεταβλητές μέλους, χρησιμοποιώντας το όνομα του συγκεκριμένου αντικειμένου που θέλουμε να τυπώσουμε και τον τελεστή ‘.’.

```
    website x = new website();

    x.name = "Cafe Au Lait";
    x.url = "http://metalab.unc.edu/javafaq/";
    x.description = "Really cool!";

    x.print();
```

Παρατήρησε ότι μέσα στην κλάση του website δεν χρειάζεται να χρησιμοποιούμε x.name ή x.url, name και url είναι αρκετά. Αυτό ισχύει γιατί η μέθοδος print πρέπει να καλείται από ένα συγκεκριμένο στιγμιότυπο κλάσης του website.

Η μέθοδος print() έχει ολοκληρωτικά ενθυλακωθεί μέσα στην κλάση του website. Όλοι οι μέθοδοι στη Java πρέπει να ανήκουν σε μια κλάση.

Βασικές κατηγορίες μεθόδων

Μέθοδοι Κατασκευής (Constructors).

Η δημιουργία ενός αντικειμένου γίνεται με την χρήση της δήλωσης new.

Παράδειγμα: TestClass x = new TestClass();

Η δήλωση new καλεί μια ειδική μέθοδο της κλάσης του αντικειμένου η οποία ονομάζεται μέθοδος κατασκευής γιατί αναλαμβάνει την εργασία που απαιτείται για την δημιουργία του αντικειμένου. Μία κλάση μπορεί να περιέχει και περισσότερες από μία μεθόδους κατασκευής. Όλες φέρουν το όνομα της κλάσης. Εάν δεν περιλαμβάνονται μέθοδοι κατασκευής σε μία κλάση, αυτή θα κληρονομήσει μια μέθοδο κατασκευής χωρίς ορίσματα από την υπερκλάση της.

Μέθοδοι πρόσβασης (Accesors).

Είναι απλές μέθοδοι οι οποίες επιστρέφουν τιμές των βασικών μεταβλητών ενός αντικειμένου. Οι μέθοδοι πρόσβασης μπορούν να παραληφθούν, για παράδειγμα όταν οι μεταβλητές είναι public. Καλό όμως είναι να αποφεύγουμε τις public μεταβλητές και να χρησιμοποιούμε μεθόδους πρόσβασης.

Μέθοδοι ελέγχου ισότητας αντικειμένων (equals).

Αποτελούν βασικές μεθόδους οι οποίες ελέγχουν εάν δύο αντικείμενα μιας κλάσης είναι ίσα μεταξύ τους.

Κατά την ανάπτυξή τους πρέπει να λαμβάνονται υπ’ όψη όλες οι ιδιότητες και συμπεριφορές των αντικειμένων.

Κλάσεις

Μέθοδος εκτύπωσης αντικειμένου (toString()).

Υπάρχει μια ειδική μέθοδος στη Java για την εκτύπωση ενός αντικειμένου
public String toString(){
……
}
Με αυτόν τον τρόπο είναι δυνατή η εκτύπωση ενός αντικειμένου μέσω της μεθόδου println().

Κατασκευαστής (Constructors)

Η πρώτη μέθοδος που χρειάζονται οι περισσότερες κλάσεις είναι ο κατασκευαστής. Μια δομή δημιουργεί ένα νέο στιγμιότυπο της κλάσης. Αρχικοποιεί όλες τις μεταβλητές και εκτελεί οποιαδήποτε ενέργεια χρειάζεται για να προετοιμαστεί η κλάση και να χρησιμοποιηθεί. Στη γραμμή website x = new website(); η website( ) είναι μια συνάρτηση κατασκευής. Αν δεν υπάρχει τέτοια συνάρτηση η Java δημιουργεί εξ' ορισμού μια, αλλά είναι καλύτερο να είσαι σίγουρος ότι έχεις ορίσει τη δική σου. Ο κατασκευαστής γράφει μια public μέθοδο, η οποία έχει το ίδιο όνομα με την κλάση. Έτσι, η δομή του website μας καλείται website(). Εδώ είναι μια απλή website κλάση με μια δομη που αρχικοποιεί όλα τα μέλη με μηδένικά strings.

```
class website {

  String name;
  String url;
  String description;

  public website() {
    name = "";
    url  = "";
    description = "";
  }

}
```

Ακόμα καλύτερα πρέπει να δημιουργήσουμε μια δομή που να δέχεται τρεία strings σαν ορίσματα και να τα χρησιμοποιεί για να αρχικοποιήσει τις μεταβλητές μέλους ως εξής:

```
class website {

  String name;
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

}
```

θα το χρησιμοποιήσουμε ως εξής:

```
    website x = new website("Cafe Au Lait", "http://metalab.unc.edu/javafaq/",
"Really cool!");
    x.print();
```

Αυτό ταιριάζει με την προθέσή μας να κρατάμε τον κώδικα σχετικό με την κατάλληλη λειτουργία της κλάση μέσα σε μια τάξη.

Όμως τι συμβαίνει όταν θέλουμε να δημιουργήσουμε ένα website του οποίου μερικές φορές ξέρουμε το url,το όνομα και την περιγραφή και μερικές φορές δεν το ξέρουμε; Καλύτερα ας τα χρησιμοποιήσουμε και τα δυο:

```
class website {

  String name;
```

```
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

  public website() {
    name = "";
    url  = "";
    description = "";
  }

}
```

Αυτό καλείται μέθοδος υπερφόρτωσης ή πολυμορφισμού. Ο πολυμορφισμός είναι ένα χαρακτηριστικό των αντικειμενοστρεφών γλωσσών που αφήνουν ένα όνομα να αναφέρεται σε διαφορετικές μεθόδους,εξαρτώμενο από το περιεχόμενό τους. Το σημαντικό στο περιεχόμενο είναι ο τύπος και ο αριθμός των ορισμάτων της μεθόδου. Σ'αυτήν την περίπτωση χρησιμοποιούμε την πρώτη έκδοση της μεθόδου αν έχουμε τρεία ορίσματα και την δεύτερη έκδοση αν δεν έχουμε καθόλου ορίσματα. Αν έχεις ένα, δύο ή τέσσερα ορίσματα string στη δομή, ή ορίσματα που δεν είναι string, ο compiler δημιουργεί ένα λάθος γιατί δεν έχει μια μέθοδο που να ταιριάζει με τη μέθοδο που ζητήθηκε.

Μέθοδοι toStrings

Η μέθοδος print είναι κοινή σε κάποιες γλώσσες, αλλά τα περισσότερα προγράμματα της Java την χειρίζονται διαφορετικά. Μπορείς να χρησιμοποιήσεις System.out.println() για να τυπώσεις κάποιο αντικείμενο.

Όμως για καλά αποτελέσματα η classs σου πρέπει να έχει μια μέθοδο toString(), που να μορφοποιεί τα δεδομένα των αντικειμένων με ένα λογικό τρόπο και να επιστρέφει ένα string. Παρακάτω βλέπουμε πως χρησιμοποιείται αυτό στο παράδειγμα του website:

```
public class ClassTest {

  public static void main(String args[]) {

    website x = new website("Cafe Au Lait", "http://metalab.unc.edu/javafaq/",
"Really cool!");
    System.out.println(x);

  }

}


class website {

  String name;
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

  public website() {
    name = "";
    url  = "";
```

```
    description = "";
  }

  public String toString() {
    return  (name + " at " + url + " is " + description);
  }

}
```

6.6. Πίνακες

Σε αντίθεση με τη C/C++, οι πίνακες στη Java υλοποιούνται ως αντικείμενα. Αυτό έχει πολλά οφέλη, όπως π. χ. το ότι δεν χρειάζεται να αποδεσμεύουμε τη μνήμη που χρησιμοποιεί ένας πίνακας (την αποδέσμευση την αναλαμβάνει το σύστημα περισυλλογής σκουπιδιών της Java), ή το ότι είναι μεταβλητού μεγέθους από κατασκευής (κάτι που δεν ισχύει για τη C/C++). Οι πίνακες γενικά χρησιμοποιούνται για την οργάνωση και καταχώρηση όμοιων αντικειμένων. Μπορούν να είναι μίας ή πολλών διαστάσεων και ο τρόπος προσπέλασης των στοιχείων είναι παρόμοιος με αυτόν της C++. Μπορούμε να ορίσουμε ένα μονοδιάστατο πίνακα πολύ απλά ως εξής:

type table[];

Ενώ αν θέλουμε να τον δημιουργήσουμε κιόλας, χρησιμοποιούμε την new:

table = new type[size];

Ή μπορούμε να πραγματοποιήσουμε και τα δύο στάδια με μία εντολή:

type table[] = new type[size];

Ως type θεωρούμε τον τύπο δεδομένων των αντικειμένων του πίνακα και μπορεί να είναι είτε ένας από τους απλούς τύπους (bool, byte, short, int, long, float, double, char, String) είτε το όνομα μιας κλάσης αντικειμένων (της Java ή δική μας). Το size απεικονίζει το μέγεθος του πίνακα table. Όπως και στη C++, μπορούμε να προσπελάσουμε τα στοιχεία του πίνακα με την σύνταξη table[i], όπου i η θέση του στοιχείου που μας ενδιαφέρει. Εδώ πρέπει να σημειωθεί ότι σε αντιστοιχία με τη C++ η Java πραγματοποιεί την αρίθμηση των πινάκων από το μηδέν (0) ως το size-1. Δηλαδή αν έχουμε έναν πίνακα A με 10 στοιχεία το πρώτο στοιχείο είναι το A[0] και το τελευταίο το A[9]. Ακολουθεί ένα παράδειγμα για καλύτερη κατανόηση:

int data[] = new int[10];

```
int i;

System. out. println(“Size of array data: “ + data. length);

for (i = 0; i < data. length; i++) {

data[i] = i*i;

System. out. println(“data[“ + i + “] = “ + data[i]);

}
```

Το αποτέλεσμα του κώδικα αυτού θα είναι:

```
Size of array data: 10

data[0] = 0

data[1] = 1

data[2] = 4

data[3] = 9

data[4] = 16

data[5] = 25
```

6.7. Οι Συμβολοσειρές στη Java

Αποθήκευση Κειμένου σε συμβολοσειρές

Οι συμβολοσειρές είναι συνηθισμένα στοιχεία στον προγραμματισμό γιατί παρέχουν ένα τρόπο για να αποθηκευτεί κείμενο και να παρουσιάζεται στους χρήστες. Το πιο βασικό στοιχείο μιας συμβολοσειράς είναι ένας χαρακτήρας. Ένας χαρακτήρας είναι ένα γράμμα, ένας αριθμός, ένα σημείο στίξης η κάποιο άλλο σύμβολο. Στα προγράμματα της Java, ένας χαρακτήρας είναι ένα από τα είδη πληροφοριών που μπορούν να αποθηκευτούν σε μια μεταβλητή. Οι μεταβλητές χαρακτήρα δημιουργούνται με τον τύπο char σε μια δήλωση όπως παρακάτω:

Char keyPressed;

Αυτή η δήλωση δημιουργεί μια μεταβλητή που ονομάζετε keyPressed, η όποια μπορεί να αποθηκεύσει ένα χαρακτήρα. Όταν δημιουργείτε μεταβλητές χαρακτήρων, μπορείτε να τους ορίσετε μια αρχική τιμή, με τον εξής τρόπο:

Char quitKey='@';

Παρατηρήστε ότι η τιμή του χαρακτήρα πρέπει να περιβάλλεται από μονά εισαγωγικά. Αν δεν συμβεί κάτι τέτοιο, ο μεταγλωττιστής της Java θα εμφανίσει σφάλμα κατά τη μεταγλώττιση του προγράμματος. Μια συμβολοσειρά είναι μια σειρά από χαρακτήρες. Μπορείτε να ορίσετε μια μεταβλητή η οποία θα αποθηκεύει μια τιμή συμβολοσειράς, χρησιμοποιώντας το κείμενο string και το όνομα της μεταβλητης, όπως στην επόμενη δήλωση

String fullName="Ada McGrath Stewart";

Αυτή η δήλωση δημιουργεί μια μεταβλητή συμβολοσειράς η οποία ονομάζεται fullName και αποθηκεύει σε αυτή το κείμενο Ada McGrath Stewart που είναι το όνομα της Χολι Χαντερ στην ταινία Το Πιάνο. Σε μια δήλωση της Java, μια συμβολοσειρά περικλείεται από διπλά εισαγωγικά. Αυτά τα εισαγωγικά δεν αποτελούν μέρος της συμβολοσειράς.

Σε αντίθεση με τα άλλα είδη μεταβλητών που έχετε χρησιμοποιήσει –int, float, char, Boolean και λοιπά- το ονομαστού τύπου string γράφεται με κεφαλαία. Αυτό γίνετε γιατί οι συμβολοσειρές είναι διαφορετικές σε σχέση με τους άλλους τύπους μεταβλητών στη Java. Οι συμβολοσειρές ανήκουν στην ειδική κατηγορημάτων αντικειμένων και οι τύποι όλων των αντικειμένων γράφονται με κεφαλαία. Θα μάθετε για τα αντικείμενα στο κεφάλαιο 10. το σημαντικό σημείο αυτού του κεφαλαίου είναι ότι οι συμβολοσειρές διαφέρουν από άλλους τύπους μεταβλητών και εξαιτίας αυτής της διαφορας, το String γράφεται με κεφαλαία όταν χρησιμοποιούνται συμβολοσειρές σε μια δήλωση.

Εμφάνιση συμβολοσειρών σε πρόγραμμα

Ο πιο βασικός τρόπος για να εμφανίζεται μια συμβολοσειρά σε ένα πρόγραμμα της Java είναι η δήλωση System. out. println(). Αυτή η δήλωση παίρνει τις συμβολοσειρές και τις άλλες μεταβλητές που βρίσκονται μέσα στις παρενθέσεις και τις εμφανίζει. Η παρακάτω δήλωση δείχνει μια γραμμή κειμένου στην συσκευή εξόδου του συστήματος, που είναι η οθόνη του υπολογιστή:

System. out. println(“onomazomai Aiginitoy Maria. ”)

Η δήλωση αυτή θα κάνει την παρακάτω πρόταση να εμφανιστεί

Onomazomai Aiginitoy Maria.

Η εμφάνιση μιας γραμμής κειμένου στην οθόνη συχνά ονομάζεται printing και το println() σημαίνει «εμφάνισε αυτή την γραμμή».

Ένας άλλος τρόπος για να εμφανιστεί κείμενο είναι το System. out. print(). Αυτή η δήλωση συμβολοσειρές και άλλες μεταβλητές μέσα σε παρενθέσεις αλλά σε αντίθεση με το System. out. println(), επιτρέπει σε διαδοχικές δηλώσεις να εμφανίζουν κείμενο στην ίδια γραμμή.

Παράδειγμα:

System. out. print(“onomazomai”)

System. out. print(“Aiginitoy Mpima”)

System. out. print(“Maria”)

System. out. print(“kai”)

System. out. print(“menw”)

System. out. print(“sthn”)

System. out. print(“Patra”)

Αυτές οι δηλώσεις θα εμφανίσουν το εξής κείμενο:

Onomazomai Aiginitoy Mpima Maria kai menw sthn Patra

Χρήση ειδικών χαρακτήρων σε συμβολοσειρές

Όταν δημιουργείτε ή όταν εμφανίζεται μια συμβολοσειρά το κείμενο της θα πρέπει να περιβάλλεται από διπλά εισαγωγικά για να φαίνεται η αρχή και το τέλος της. Αυτά τα εισαγωγικά δεν εμφανίζονται όμως και έτσι μπορεί να αναρωτηθεί κανείς τι πρέπει να κάνει αν θέλει να τα εμφανίσει. Για να εμφανιστούν η Java έχει δημιουργήσει έναν ειδικό κώδικα το οποίο μπορεί να τοποθετηθεί σε μια συμβολοσειρά *. Όποτε απαντάτε αυτός ο κώδικας σε μια συμβολοσειρά αντικαθίσταται από διπλά εισαγωγικά. Με αυτόν τον τρόπο μπορούν να εισαχθούν διάφοροι ειδικοί χαρακτήρες και όλα αρχίζουν με κάθετο (\).

Ειδικοί Χαρακτήρες	Εμφανίζουν
\’	Μονά εισαγωγικά
\”	Διπλά εισαγωγικά
\\	Κάθετοι
\t	Tab
\b	Χαρακτήρες διαγραφής
\f	Χαρακτήρες επιστροφής
\n	Νέα γραμμή

Παράδειγμα για το Χαρακτήρα Νέας γραμμής :

Ο χαρακτήρας αυτός κάνει το κείμενο που ακολουθεί το χαρακτήρα νέας γραμμής να εμφανίζεται στην αρχή της επόμενης σειράς.

System. out. println(“onomazomai\nAiginitoy Mpima Maria”)

Και θα εμφανίσει:

Onomazomai

Aiginitoy Mpima Maria

6.8. Τα προγράμματα της Java

Με τα ακόλουθα βήματα μπορεί να κατασκευάστει μια ανεξάρτητη Java εφαρμογή.

Δημιουργήστε το αρχείο πηγαίου κώδικα HelloWorldApp.java που να περιέχει τις ακόλουθες

εντολές:

```
/**
* The HelloWorldApp class implements an application that
* simply displays "Hello World!" to the standard output.
*/
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Κάντε τη μεταγλώττιση χρησιμοποιώντας τον Java compiler:

javac HelloWorldApp.java

Ο compiler θα δημιουργήσει το αρχείο HelloWorldApp.class

Εκτελέστε την εφαρμογή καλώντας τον Java interpreter:

java HelloWorldApp

Θα δείτε στην οθόνη σας το μήνυμα "Hello World!"

javac HelloWorld.java

Δομή ενός προγράμματος σε JAVA

< ΕΙΣΑΓΩΓΗ ΕΤΟΙΜΩΝ ΚΛΑΣΕΩΝ ΑΠΟ ΤΗ ΒΙΒΛΙΟΘΗΚΗ >

<ΟΡΙΣΜΟΣ ΝΕΑΣ ΚΛΑΣΗΣ>

<ΟΡΙΣΜΟΣ ΝΕΑΣ ΚΛΑΣΗΣ>

κ.ο.κ.

Μία από όλες τις κλάσεις θα παίξει το ρόλο της αφετηρίας του προγράμματος, με άλλα λόγια θα είναι σαν την main() της C. Αναλυτικότερα, θα αναλάβει να κατασκευάσει τα αντικείμενα που χρειάζεται το πρόγραμμα, με τη χρήση των κλάσεων που ορίσαμε ή φέραμε από τη βιβλιοθήκη. Φυσικά την αρχική κλάση θα την υποδείξουμε εμείς όταν θα ξεκινήσουμε τον interpreter της γλώσσας.

Όπως ειπώθηκε παραπάνω τα αντικείμενα περιέχουν δεδομένα και παρέχουν

μεθόδους για την επεξεργασία των δεδομένων αυτών καθώς και για την επικοινωνία με άλλα αντικείμενα.

Κατ` αρχήν τα δεδομένα μπορεί να είναι άλλα αντικείμενα που περιέχονται μέσα σε ένα άλλο αντικείμενο ή μπορεί να είναι κοινές μεταβλητές. Τις μεταβλητές και τα αντικείμενα αυτά τα καλούμε πεδία ή instance variables του αντικειμένου.

Μία κλάση (ή ένα αντικείμενο) περιέχει και μεθόδους για την επικοινωνία

του με τον έξω κόσμο, δηλαδή τα άλλα αντικείμενα.

Οι μέθοδοι αυτοί υλοποιούνται σαν συναρτήσεις. Όταν λοιπόν κάποιος θέλει να ζητήσει κάτι από ένα αντικείμενο, (ή εναλλακτικά να στείλει ένα μήνυμα -αίτημα), δεν έχει παρά να καλέσει - εκτελέσει την αντίστοιχη μέθοδο του αντικειμένου. Αυτό γίνεται πολύ απλά ως εξής :

<Αντικείμενο>.<μέθοδος>(<Λίστα παραμέτρων>);

Π.χ. mycar.startEngine();

Να σημειώσουμε ότι μια μέθοδος μπορεί να καλεί άλλες μεθόδους του ίδιου αντικειμένου ή να επεξεργάζεται τα πεδία του. Φυσικά μπορεί να

καλεί και μεθόδους ξένων αντικειμένων εφόσον έχει κάποιον δείκτη σε αυτά.

Σημειωση:

κρατά τα δεδομένα του κρυφά από τα άλλα αντικείμενα αλλά και να προσφέρει μεθόδους με τις οποίες μπορούν άλλα αντικείμενα να επικοινωνούν και να αλληλεπιδρούν μαζί του.Επιπλέον ο κώδικας που υλοποιεί αυτές τις μεθόδους είναι άγνωστος σε άλλες κλάσεις ή αντικείμενα.

Έτσι έχουμε την δημιουργία ενός interface επικοινωνίας ανεξάρτητου από την

υλοποίηση και την εσωτερική δομή του αντικειμένου. Αυτό λέγεται implementation hiding που είναι μία μορφή data abstraction.

Ορισμός κλάσεων στην JAVA

Μία κλάση της java έχει την ακόλουθη δομή :

class <class_name> {

<data_type or class_name> <variables or objects>;

<returned type> <method_name> (<parameter list>) {

<method body>

}

<data_type or class_name> <variables or objects>;

<returned type> <method_name> (<parameter list>) {

<method body>

}

κ.ο.κ.

```
}
```

Π.χ.

```
class Apple {
int num_of_vitamins; // πεδίο
int countVitamins() { // μέθοδος
return num_of_vitamins;
}
Vitamins v1, v2, v3; // πεδία
Vitamin getVitamin(int num) { // μέθοδος
return ....;
}
}
```

Το num_of_vitamins είναι μία κοινή μεταβλητή τύπου int. Αντίθετα το Vitamins είναι όνομα μιας κλάσης και συνεπώς τα v1, v2, v3 θα είναι αντικείμενα της κλάσης Vitamin ή καλύτερα αναφορές (κάτι σαν δείκτες) στα αντίστοιχα αντικείμενα. Τέλος βλέπουμε ότι οι μέθοδοι μπορούν να έχουν επιστρεφόμενο τύπο τόσο έναν απλό τύπο δεδομένων όσο και μία κλάση.

Μπορούμε να δηλώνουμε τα πεδία και τις μεθόδους με όποια σειρά θέλουμε. Επίσης μπορούμε να τα χρησιμοποιούμε πριν ακόμη τα δηλώσουμε. Ωστόσο συνιστάται τα πεδία να τοποθετούνται στην αρχή της κλάσης ώστε ο κώδικας να είναι πιο ευανάγνωστος. Ακόμη η τοποθέτηση κατατοπιστικών σχολίων συμβάλλει σημαντικά στην βελτίωση της αναγνωσιμότητας του προγράμματος και διευκολύνει την διαδικασία αποσφαλμάτωσης

(debugging).

```
class test {

float add2Accum(float f) { // Add to accumulator.

return (Accumulator += f);

}

float Accumulator = 0.0; // The accumulator.

}
```

6.9. Αλγόριθμοι εύρεσης συμβολοσειράς σε κείμενο

6.9.1. Αλγόριθμος Brute Force

```
import java. io. BufferedReader;

import java. io. InputStreamReader;

public class Brutal_force {

   public static void main(String[] args) {

      BufferedReader br = new BufferedReader(new
InputStreamReader(System. in));

      int i=0,  j=0,  index=0,  N,  M;

      boolean found=false;

      String a = "";

      String p = "";


        System. out. println("Brutal Force Algorithm");

      System. out. println("give the phrase");

      try {

         a = br. readLine();

      }

      catch (Exception e) {

      }


      System. out. println("give the word");
```

```
try {
    p = br. readLine();
}
catch (Exception e) {
}

N=a. length();
M=p. length();

do {
    if (a. charAt(i) == p. charAt(j))
    {
        i++;
        j++;
        System. out. println("i entered if and i is "+i+" and j is "+j);
    } else {
        i=i-j+1;
        j = 0;
        System. out. println("i entered else and i is "+i+ "and j is "+j);
    }
} while ((j <= M - 1) && (i <= N - 1));
```

```
        if (j > M - 1)
        {
            found = true;
            index = i - M;
        }

        if (found) {
            System. out. println("the word is found at position " + index+" of the table,  that is on the character number "+(index+1));
        } else {
            System. out. println("the word is not found inside the phrase");
        }
    }
}
```

ΤΕΚΜΗΡΙΩΣΗ ΤΟΥ ΠΑΡΑΔΕΙΓΜΑΤΟΣ

1η - 2η γραμμή

Στην java δεν έχουμε βιβλιοθήκη γιατί φτιάχνουμε γραφικό περιβάλλον. Οι κλάσεις ανήκουν στο πακέτο (package)της Java java. io. Java packages: τα πακέτα της Java είναι αυτόνομες μονάδες λογισμικού, που περιέχουν κλάσεις και μεθόδους που χρησιμοποιούνται για ένα συγκεκριμένο σκοπό. Έχουν την ίδια χρησιμότητα με τις βιβλιοθήκες ρουτινών, που χρησιμοποιούν άλλες γλώσσες προγραμματισμού (C, C++). Συνήθως είναι αρκετά γενικευμένης χρήσης, ώστε να επωφελείται ο κάθε προγραμματιστής από κάτι ήδη δοκιμασμένο και να ασχοληθεί με το ίδιο το πρόγραμμα του. Η Java παρέχει

πληθώρα τέτοιων πακέτων, για γραφικά, προγραμματισμό βάσεων δεδομένων, δικτυακό προγραμματισμό, κλπ. Ένα από τα βασικά τέτοια πακέτα, είναι για την διαχείριση των αρχείων, το java. io. Τα πακέτα τα χρησιμοποιούμε στα προγράμματά μας με την εντολή import, εκτός της κλάσης, δηλαδή:

import java. io. *;

Η παραπάνω εντολή επιτρέπει στο πρόγραμμά μας να χρησιμοποιήσει το πακέτο java. io και όλες τις κλάσεις (ή άλλα πακέτα) περιέχει αυτό.

3η γραμμή

Αυτή η εντολή σημαίνει υπολογιστή δώσε στο πρόγραμμα της Java το όνομα Brutal_force. Με την δήλωση class δίνουμε στο πρόγραμμα όνομα αλλά χρησιμοποιείται και για τον προσδιορισμό άλλων πραγμάτων σχετικά με το πρόγραμμα. Το όνομα ενός προγράμματος πρέπει να είναι ίδιο με το πρώτο τμήμα του ονόματος αρχείου του και θα πρέπει να χρησιμοποιείται η ίδια συμφωνία πεζών –κεφαλαίων.

4η γραμμή

Αυτή η γραμμή λέει στον υπολογιστή ότι το βασικό τμήμα του προγράμματος αρχίζει εδώ. Τα προγράμματα τις java είναι οργανωμένα σε διαφορετικά τμήματα και έτσι χρειάζεται ένας τρόπος για να εντοπίζεται το τμήμα ενός προγράμματος που θα αντιμετωπιστεί πρώτο.

6η -7η γραμμή

Δηλώνουμε εδώ τις απαραίτητες μεταβλητές του προγράμματος. Η μεταβλητή (δείκτης) i αφορά το κείμενο, ενώ η μεταβλητή j στο πρότυπο. Η τιμή της μεταβλητής N είναι το μέγεθος του String του κειμένου και η τιμή της μεταβλητής M το μέγεθος της String του προτύπου που αναζητείται μέσα στο κείμενο. Η index θα χρησιμοποιηθεί αν τελικά βρεθεί η λέξη μέσα στο κείμενο και θα δείχνει το σημείο από το οποίο ξεκινά αυτή. Η found είναι μια λογική μεταβλητή που θα χρησιμοποιηθεί στο τελευταίο στάδιο της του προγράμματος,

όπου θα πρέπει ανάλογα με το αν βρέθηκε η λέξη ή όχι, να τυπωθούν τα αντίστοιχα μηνύματα προς την ενημέρωση του χρήστη.

8η – 9η γραμμή

Επειδή χρειάζεται ένας χώρος για αποθηκεύονται πληροφορίες για ένα σύντομο χρονικό διάστημα χρησιμοποιείται μια μεταβλητή. Ένας χώρος στον οποίο μπορεί να αποθηκεύονται πληροφορίες όπως, ακέραιους, αριθμούς κινητής υποδιαστολής, τιμές σωστού/λάθους, χαρακτήρες και γραμμές κείμενου. Οι πληροφορίες μπορεί να αλλάζουν και έτσι εξηγείται ο όρος μεταβλητή. Σε αυτό το πρόγραμμα λέμε στον υπολογιστή να αρχικοποίηση τα a και p που είναι δύο συμβολοσειρές (string), γραμμές κειμένου και προτύπου αντίστοιχα που περιλαμβάνουν κενούς χαρακτήρες.

10η -11η γραμμή

Η εντολή αυτή λέει στον υπολογιστή να εμφανίσει μια γραμμής στην συσκευή εξόδου του συστήματος και συγκεκριμένα λέει εμφάνισε «Brutal Force Algorithm» και « give the phrase» αντίστοιχα.

12η - 22η γραμμή

Για την αντιμετώπιση προβλημάτων υπάρχουν κάποιοι μηχανισμοί που ονομάζονται, exceptions. Οι μηχανισμοί αυτοί εμφανίζουν στον προγραμματιστή πιθανά προβλήματα και τον αναγκάζει να τα αντιμετωπίσει. Ο κώδικας αντιμετώπισης του προβλήματος, καλείται μόνο αν παρουσιαστεί το πρόβλημα και χωρίζεται από τον υπόλοιπο κώδικα για λόγους καλύτερης οργάνωσης. Η τεχνική είναι η εξής: θα δοκιμάζεται η εκτέλεση ενός τμήματος κώδικα (try) και για κάθε πιθανό πρόβλημα (exception) θα πρέπει να παρέχεται ένας αντίστοιχος μηχανισμός αντιμετώπισης του προβλήματος (catch). Επόμενος σε αυτές τις γραμμές υπάρχουν εντολές για την είσοδο του κειμένου και του προτύπου από τον χρήστη. Έτσι έχει δημιουργηθεί το αντικείμενο br της κλάσης BufferedReader για το οποίο καλείται η μέθοδος readLine(). Λόγω κανονισμών όπως αναφέραμε τοποθετούμε αυτόν τον επιπλέον κώδικα, έτσι ο χρήστης μετά από προτρεπτικό μήνυμα δίνει την φράση και το πρότυπο.

Αποτέλεσμα τα String που πληκτρολογεί ο χρήστης να εκχωρούνται στις κατάλληλες μεταβλητές.

23η-24η γραμμή

Η μέθοδος length() επιστρέφει το μέγεθος του String οπότε η μεταβλητή N κρατά το μέγεθος της φράσης, ενώ η μεταβλητή M το μέγεθος του προτύπου.

25η – 36η γραμμή

Από την 25η γραμμή ξεκινάμε να προγραμματίζουμε την διαδικασία αναζήτησης μια συμβολοσειράς χρησιμοποιώντας τον αλγόριθμο Brutal_force. Οι διαδικασία εντολών απαιτεί τον προγραμματισμό δεικτών i και αντίστοιχα j. Επομένως ξεκινάει ο βρόχος do-while. Όταν ένα πρόγραμμα εκτελείται και φτάνει για πρώτη φορά στο βρόχο αυτό η επεξεργασία των δηλώσεων ανάμεσα στο do και while γίνεται αυτόματα και στην συνέχεια εξετάζεται η υπόθεση while για να προσδιοριστεί αν θα επαναληφθεί ο βρόχος. Αν υπόθεση while είναι αληθής, ο βρόχος επαναλαμβάνεται ακόμα μια φορά ενώ αν είναι ψευδής σταματάει.

Ο βρόχος έχει την εξής μορφή:

do

εντολές;

while (συνθήκη);

Στην συνθήκη αυτή πρώτα εκτελούνται οι εντολές και μετά η συνθήκη και αν βρεθεί ψευδής τερματίζεται και ο βρόχος.

Οι εντολές εδώ γίνεται με την εντολή if …. else

Η εντολή if χρησιμοποιείται όταν θέλουμε να εκτελέσουμε κάποιες εντολές μόνο όταν ικανοποιείται κάποια συνθήκη και έχει το παρακάτω συντακτικό:

if (συνθήκη)

εντολές;

else

εντολές;

Η εντολή if μας βοηθάει στον έλεγχο μίας λογικής έκφρασης (της συνθήκης) και αν είναι αληθής εκτελούνται μία ή περισσότερες εντολές ενώ αν είναι ψευδής δεν εκτελούνται με την φράση else με την οποία επιτρέπεται η εκτέλεση μίας ομάδας εντολών εναλλακτικά αν δεν ισχύει η συνθήκη. Αν οι εντολές είναι περισσότερες από μία τότε θα πρέπει οι εντολές να περικλειστούν ανάμεσα από άγκιστρα (τα οποία ομαδοποιούν εντολές). Συγκεκριμένα σε αυτές τι γραμμές αρχίζει η αναζήτηση του προτύπου μέσα στο κείμενο. Η δομή do- while εκτελείται μέχρι να φτάσει ή να δείχνει ο δείκτης i στο τέλος του κειμένου ή ο j να δείχνει στο τέλος του String του προτύπου. Όταν κάτι τέτοιο διαπιστώνεται, οι χαρακτήρες του προτύπου και του κείμενου δεν είναι ίδιοι ο δείκτης j αυξάνεται σύμφωνα με το αντίστοιχο στοιχείο και δεν μηδενίζεται

37η – 46η γραμμή

Σε αυτές τις γραμμές αφού έχει τελειώσει ο προηγούμενος κώδικας μας λέει ότι εάν ο δείκτης j φτάσει να γίνει μεγαλύτερος του M-1 και αντίστοιχα ο δείκτης i μεγαλύτερος του N-1 τότε θα έχουμε found αληθής δηλαδή το found παίρνει την τιμή «true» και πιο συγκεκριμένα η λέξη πρότυπο ταυτίστηκε με κάποια λέξη του κειμένου και βρέθηκε. Τότε το πρόγραμμα θα βγάλει στην οθόνη το αντίστοιχο μήνυμα καθώς και σε ποια θέση βρέθηκε η λέξη. Αυτό γίνεται με την βοήθεια της μεταβλητής index όπου μας δείχνει τη θέση μέσα στο κείμενο από την οποία άρχισε το πρότυπο. Διαφορετικά όταν το πρόγραμμα τρέξει και ψάξει όλους τους χαρακτήρες του κειμένου και δεν βρεθεί η λέξη πρότυπο θα βγάλει στη οθόνη ένα μήνυμα που θα επεξηγεί ότι η λέξη δεν βρέθηκε μέσα στο κείμενο.

6.9.2. Αλγόριθμος KMP Search

import java. io. BufferedReader;

import java. io. InputStreamReader;

public class KMP_Search {

```
public static void main(String[] rags) {
    Buffered Reader bra=new Buffered Reader(new InputStreamReader(System. in));
    int i=0, j=-1, index=0,  N,  M;
    boolean found=false;
    String a="";
    String p="";

    System. out. println("KMP Search Algorithm");
    System. out. println("give the phrase");
    try {
        a = br. readLine();
    }
    catch (Exception e) {
    }
    System. out. println("the phrase you wrote is: " + a);
    System. out. println("give the word");
    try {
        p = br. readLine();
    }
    catch (Exception e) {
```

```
}
System. out. println("the word you wrote is: " + p);

N=a. length();
M=p. length();
System. out. println("the phrase consists of "+N+" characters and the word consists of "+M+" characters");

//Dhmioyrgia pinaka next
int next[]=new int[M];
next[0]=-1;

do {
   if ( (j == -1) || (p. charAt(i) == p. charAt(j)) )
   {
      i++;
      j++;
      System. out. println("mpika stin if. i= "+i+" kai j="+j);

      if (p. charAt(i) != p. charAt(j)) {
         next[i] = j;
         System. out. println("mpika sto esoteriko if kai next["+i+"]="+next[i]);
```

```
            }
            else {
                next[i] = next[j];
                System. out. println("mpika sto esoteriko else kai next["+i+"]="+next[i]);
            }
        }
        else {
            j = next[j];
            System. out. println("mpika sto else kai j= "+j);
        }
        System. out. println("teleiono to do,  to i einai "+i+" kai to j einai "+j);
        System. out. println("---------------------------\n");
    } while (i < M - 1);

    //Anazitisi toy protypoy mesa sti frash
    i = 0;
    j = 0;

    do {
        System. out. println("mpika sto deytero Do\n----------------------------------");
        if ((j == -1) || (a. charAt(i) == p. charAt(j))) {
```

```
            i++;
            j++;
            System. out. println("mpika sto if toy deyteroy do i="+i+" j="+j);
        }
        else {
            j = next[j];
            System. out. println("mpika sto else toy deyteroy do i="+i+" j="+j);
        }
    } while ((j < M) && (i < N));

    //Elegxos an vrethike i leksi
    if (j >= M) {
        found = true;
        index = i - M;
    }

    if (found) {
        System. out. print("the word is found in the phrase,  at position "+index+"
of the table");
    } else {
        System. out. println("the word is not found inside the phrase");
    }
```

```
    }

}
```

ΤΕΚΜΗΡΙΩΣΗ ΤΟΥ ΠΑΡΑΔΕΙΓΜΑΤΟΣ KMP_Search

Στον αλγόριθμο KMP_Search, όταν διαπιστωθεί μια μη ταύτιση μερικών χαρακτήρων, τότε η αναζήτηση δεν συνεχίζεται από τον επόμενο χαρακτήρα του κειμένου απ' όπου άρχισε η ταύτιση, αλλά από τον τρέχοντα.

Για παράδειγμα, έστω ότι το κείμενο είναι:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Τ	Α		Δ	Ε	Δ	Ο	Μ	Ε	Ν	Α		Κ	Α	Ι	

17	18	19	20	21	22	23	24	25	26	27	28	29
Ο	Ι		Δ	Ο	Μ	Ε	Σ		Τ	Ο	Υ	Σ

και το πρότυπο που αναζητάτε είναι

1	2	3	4	5
Δ	Ο	Μ	Ε	Σ

Σε αυτό το παράδειγμα, στις θέσεις 6 έως 9 υπάρχει ταύτιση των τεσσάρων πρώτων χαρακτήρων του προτύπου. Στη 10η θέση δεν υπάρχει ταύτιση, αλλά τώρα ο δείκτης i δεν χρειάζεται να οπισθοχωρήσει στην 7η θέση, γιατί σίγουρα δεν μπορεί να υπάρξει ταύτιση από την 7η μέχρι τη 10η.

Ο αλγόριθμος KMP μετά από μια μη ταύτιση δεν αλλάζει το δείκτη του κειμένου i, αλλά μόνο τον δείκτη του προτύπου j. Προς τούτο χρειάζεται μια

προεπεξεργασία προκειμένου να σχηματιστεί ένα διάνυσμα αλμάτων next, που παρέχει την αύξηση του δείκτη j. Για παράδειγμα, αν το πρότυπο είναι ABCAADE, τότε ο πίνακας next περιέχει τις τιμές 01102211. Η δημιουργία του πίνακα next βασίζεται στη σύγκριση του προτύπου με τον εαυτό του. Ο αλγόριθμος αυτό δεν είναι και πολύ καλύτερος από τον Brutal Force στις συνήθεις περιπτώσεις αναζήτησης κειμένου. Ωστόσο, έχει ένα σημαντικό πλεονέκτημα, ο δείκτης i δεν οπισθοχωρεί ποτέ. Έτσι, όταν εφαρμόζεται σε μεγάλα αρχεία αποθηκευμένα στο δίσκο, δεν απαιτεί επιπλέον προσπελάσεις δίσκου. Εξηγούμε αναλυτικότερα πώς δουλεύει ο αλγόριθμος KMP_Search υλοποιημένος στη γλώσσα προγραμματισμού java.

Συγκεκριμένα:

1η -4η γραμμή

Στις παραπάνω γραμμές δημιουργείται η κλάση KMP_Search που περιέχει τη μέθοδο main με τις εντολές του αλγορίθμου. Η κλάση αυτή περιλαμβάνεται στο πακέτο search_algs και χρειάζεται τις κλάσεις BufferedReader και InputStreamReader του πακέτου java. io ώστε να δεχθεί είσοδο από το χρήστη.

6η -9η γραμμή

Εδώ δηλώνονται οι απαραίτητες μεταβλητές του προγράμματος. Η μεταβλητή-δείκτης i αφορά στο κείμενο, ενώ η μεταβλητή-δείκτης j στο πρότυπο. Η τιμή της μεταβλητής N θα είναι το μέγεθος του String του κειμένου και η τιμή της μεταβλητής M θα είναι το μέγεθος του String του προτύπου που αναζητείται μέσα στο κείμενο. Η μεταβλητή index θα χρησιμοποιηθεί αν τελικά βρεθεί η λέξη μέσα στο κείμενο και θα δείχνει το σημείο από το οποίο ξεκινά αυτή. Η found είναι μία λογική μεταβλητή που θα χρησιμέψει το τελευταίο στάδιο του προγράμματος, όπου θα πρέπει ανάλογα με το αν βρέθηκε η λέξη ή όχι, να τυπωθούν τα αντίστοιχα μηνύματα προς ενημέρωση του χρήστη. Τέλος, αρχικοποιούνται τα String του κειμένου και του προτύπου, με κενούς χαρακτήρες.

5η και 10η-23η γραμμή

Εδώ υπάρχουν οι εντολές για την είσοδο του κειμένου και του προτύπου από το χρήστη. Προς τούτο έχει δημιουργηθεί το αντικείμενο br της κλάσης BufferedReader, για το οποίο καλείται η μέθοδος readLine(). Λόγω κανονισμών της java, υποχρεωνόμαστε να τοποθετήσουμε επιπλέον κώδικα ώστε να χειριστούμε τις εξαιρέσεις που μπορεί να προκύψουν από κάποιο λάθος στην είσοδο των String. Ο χρήστης, μετά από προτρεπτικό μήνυμα για πληκτρολόγηση, δίνει τη φράση και το πρότυπο. Σαν αποτέλεσμα, τα String που πληκτρολογεί ο χρήστης εκχωρούνται στις κατάλληλες μεταβλητές.

25η -30η γραμμή

Εδώ, η μέθοδος length() επιστρέφει το μέγεθος του String, οπότε η μεταβλητή N κρατά το μέγεθος της φράσης, ενώ η μεταβλητή M το μέγεθος του προτύπου. Δημιουργείται επίσης και ο πίνακας ακεραίων next, που θα έχει μέγεθος όσο το μήκος του πρότυπου και θα κρατά τα «άλματα» του j, ενώ στην πρώτη θέση του (next[0]) εκχωρείται το -1.

31η-52η γραμμή

Σε αυτό το κομμάτι του κώδικα, ο πίνακας next γεμίζει με στοιχεία, ουσιαστικά με σύγκριση του προτύπου με τον εαυτό του. Για αυτό το σκοπό, χρησιμοποιείται και η μεταβλητή i, η οποία βέβαια αυτή τη φορά χρησιμεύει ως δείκτης στον πίνακα next και όχι στην αρχική φράση.

54η-67η γραμμή

Αφού έχει δημιουργηθεί και γεμίσει με στοιχεία ο πίνακας next, αρχίζει η αναζήτηση του προτύπου μέσα στο κείμενο. Αυτό κάνουν οι παραπάνω γραμμές κώδικα. Η δομή do-while εκτελείται μέχρι να φτάσει ή ο δείκτης i να δείχνει στο τέλος του κειμένου ή ο j να δείχνει στο τέλος του String του προτύπου. Όταν κάτι τέτοιο διαπιστώνεται, δηλαδή ότι οι χαρακτήρες του προτύπου και του κειμένου δεν είναι ίδιοι, ο δείκτης j αυξάνεται σύμφωνα με το αντίστοιχο στοιχείο του πίνακα next και δε μηδενίζεται, όπως στον αλγόριθμο Brutal Force.

69η-76η γραμμή

Μετά το τέλος του προηγούμενου κώδικα, η λέξη-πρότυπο είτε βρέθηκε, οπότε ο δείκτης j ισούται με M είτε δε βρέθηκε, οπότε ο δείκτης j είναι μικρότερος του M. Στην περίπτωση που το πρότυπο βρέθηκε μέσα στη φράση, η μεταβλητή found παίρνει την τιμή “true” και η μεταβλητή index υπολογίζεται, ώστε να δείχνει τη θέση μέσα στο κείμενο από την οποία άρχισε το πρότυπο. Τέλος, τυπώνεται το αντίστοιχο μήνυμα, καθώς και η θέση μέσα στο κείμενο στην οποία ξεκινά η λέξη που αναζητήθηκε. Στην περίπτωση που το πρότυπο δε βρέθηκε μέσα στη φράση, τυπώνεται ανάλογο μήνυμα.

6.9.3. Αλγόριθμος Not So Naive

Οι κλάσεις BufferedReader και InputStreamReader εισάγονται, διότι αντικείμενά τους χρησιμοποιούνται για εισαγωγή κειμένου από το χρήστη.

import java. io. BufferedReader;

import java. io. InputStreamReader;

Ο Not So Naive αλγόριθμος εκτελεί αναζήτηση αλφαριθμητικού – προτύπου (pattern) σε κείμενο (text) και για τούτο εφαρμόζει το μηχανισμό του κυλιόμενου παράθυρου. Ο μηχανισμός δουλεύει ως εξής: το κείμενο σαρώνεται με τη βοήθεια ενός παραθύρου του οποίου το μέγεθος είναι m, όσο και το μέγεθος του προτύπου. Η αριστερή άκρη αυτού του παραθύρου καθώς και του κειμένου στοιχίζονται και έπειτα συγκρίνονται οι χαρακτήρες του κειμένου που βρίσκονται μέσα στο παράθυρο με τους χαρακτήρες του προτύπου. Μετά από επιτυχημένο ταίριασμα του προτύπου ή μιας αποτυχίας ταιριάσματος το παράθυρο κυλά προς τα δεξιά. Η ίδια διαδικασία επαναλαμβάνεται ωσότου η δεξιά άκρη του παραθύρου ξεπεράσει τη δεξιά άκρη του κειμένου.

public class Not_So_Naive {

public static void main(String[] args) {

p είναι το string του προτύπου (pattern) που έχει μήκος m, ενώ t είναι το string του κειμένου (text) που έχει μήκος n. Η είσοδος των αλφαριθμητικών από το

χρήστη επιτυγχάνεται με τη συνάρτηση readLine() που εφαρμόζεται στο αντικείμενο br της κλάσης BufferedReader, μέσα σε τμήμα try – catch, όπως απαιτεί η java. Τα μήκη τους υπολογίζονται με τη μέθοδο length(). j είναι η θέση του κειμένου από την οποία ξεκινά το κυλιόμενο παράθυρο ώστε να αρχίσει η σύγκριση των γραμμάτων.

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.
in));
    String p="";
    String t="";
    int m, n, j, k, ell;
    System. out. println("Not So Naive Algorithm");
    System. out. println("Enter the pattern: ");
    try {
       p=br. readLine();
    }
    catch (Exception e) {
    }
    System. out. println("Enter the text: ");
    try {
       t=br. readLine();
    }
    catch (Exception e) {
    }
```

```
m=p. length();

n=t. length();
```

Στη φάση της προ-επεξεργασίας που ακολουθεί, ο αλγόριθμος διαχωρίζει δύο περιπτώσεις για το string του προτύπου (pattern): μία στην οποία τα πρώτα δύο γράμματα είναι ίδια και μία στην οποία αυτό δεν συμβαίνει. Αυτός ο διαχωρισμός γίνεται με τον έλεγχο p. charAt(0) == p. charAt(1) και σε περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το παράθυρο εντοπισμού του προτύπου θα κυλά κατά δύο θέσεις δεξιά κάθε φορά, το οποίο οφείλεται στην εκχώρηση της τιμής 2 στη μεταβλητή k. Αυτό το επιθυμούμε, διότι αν οι χαρακτήρες p(1) και t(j+1) είναι ανόμοιοι, σίγουρα θα είναι επίσης ανόμοια το πρώτο στοιχείο του p και το t(j+1). Η δεύτερη σύγκριση δεν θα γίνει όμως λόγω της τιμής 2 που έχει εκχωρηθεί στη μεταβλητή ell και όπως θα δούμε παρακάτω, ο δείκτης j στον πίνακα χαρακτήρων που κρατά το κείμενο θα μετατοπίζεται κατά 2 θέσεις και όχι 1.

```
if (p. charAt(0) == p. charAt(1)) {
   k = 2;
   ell = 1;
}
else {
   k = 1;
   ell = 2;
}
```

Στη φάση της αναζήτησης του προτύπου μέσα στο string, ο έλεγχος j <=n-m εξασφαλίζει ότι όταν το κείμενο θα φτάνει στο τέλος του, η σύγκριση των χαρακτήρων του προτύπου και του κειμένου θα συμβαίνει μόνο μέχρις ότου αυτά να έχουν στοιχιστεί ως προς το τέλος τους, χωρίς το πρότυπο να έχει μεγαλύτερο μήκος από τους χαρακτήρες που απομένουν στο κείμενο. Έτσι,

εξασφαλίζεται ότι το παράθυρο θα είναι αρκετά μεγάλο ώστε να χωρά το πρότυπο και όταν αυτό δεν θα συμβαίνει, ο βρόχος επανάληψης τερματίζεται. Ο έλεγχος p. charAt(1) != t. charAt(j + 1) βγαίνει αληθής όταν τα αντίστοιχα στοιχεία του προτύπου και του κειμένου δεν ταυτίζονται. Σε αυτή την περίπτωση εκτελούνται οι εντολές του τμήματος if, τουτέστιν ο δείκτης j αυξάνεται κατά k θέσεις. Όπως προαναφέραμε, αν οι 2 πρώτοι χαρακτήρες του προτύπου ταυτίζονται, το k είναι 2, ώστε να αποφευχθεί αργότερα ο σίγουρα ανεπιτυχής έλεγχος του πρώτου στοιχείου του p με το t(j).

```
j = 0;

while (j <= n - m) {

  if (p. charAt(1) != t. charAt(j + 1))
```

Αν τώρα οι χαρακτήρες στη δεύτερη θέση των δύο αλφαριθμητικών δεν ταυτίζονται, δηλαδή ο έλεγχος p. charAt(1) != t. charAt(j + 1) βγαίνει ψευδής, το παράθυρο μετατοπίζεται k θέσεις δεξιά πάνω στο κείμενο. Όπως αναφέραμε παραπάνω, το k είναι 1 στην περίπτωση που οι δύο πρώτοι χαρακτήρες του προτύπου δεν είναι ίδιοι. Και τούτο διότι μπορεί ο χαρακτήρας του παραθύρου του κειμένου που δεν ταυτιζόταν με τον δεύτερο χαρακτήρα του προτύπου, να ταυτίζεται με τον πρώτο. Και επειδή αυτό δεν μπορεί να συμβαίνει όταν οι δύο πρώτοι χαρακτήρες του προτύπου είναι ίδιοι, σε αυτή την περίπτωση το k είναι 2.

```
    j += k;

  else {
```

Αν ο έλεγχος βγει ψευδής, εκτελούνται οι εντολές του τμήματος else, οι οποίες είναι μια εμφωλευμένη δομή if και μία εντολή προσαύξησης της μεταβλητής j. Εφόσον λοιπόν τα στοιχεία στη δεύτερη θέση των strings είναι ίδια, γίνεται έλεγχος και στα δεξιά μέρη τους, στην αμέσως επόμενη θέση του πίνακα χαρακτήρων με τη μέθοδο equals, η οποία συγκρίνει τους πρώτους m-2

χαρακτήρες του προτύπου και του κειμένου, ξεκινώντας από τη θέση 2 και j+2 αντίστοιχα. . Αν η σύγκριση δείξει ταύτιση, η συνάρτηση επιστρέφει τιμή true. Για να θεωρηθεί όμως ότι βρέθηκε το πρότυπο, πρέπει επίσης οι πρώτοι χαρακτήρες του προτύπου και του παραθύρου να είναι ίδιοι. Αν αυτές οι δύο προϋποθέσεις πληρούνται, έχουμε μία επιτυχημένη απόπειρα εύρεσης του προτύπου μέσα στο κείμενο και εμφανίζεται αντίστοιχο μήνυμα στο χρήστη. Τέλος, η μεταβλητή j αυξάνεται κατά ell, ώστε να συνεχιστεί η αναζήτηση του προτύπου μέσα στο κείμενο, αφού αυτό ακόμα κι αν έχει βρεθεί μία φορά, υπάρχει πιθανότητα να ξαναβρεθεί στη συνέχεια.

```
        if (p. substring(2, m). equals(t. substring(j+2, j+m)) && p. charAt(0) == t. charAt(j))

        System. out. println("Pattern found at position"+j);

      j += ell;

      }

    }


  }

}
```

6.9.4. Αλγόριθμος Quick Search

Οι κλάσεις BufferedReader και InputStreamReader εισάγονται, διότι αντικείμενά τους χρησιμοποιούνται για εισαγωγή κειμένου από το χρήστη.

```
import java. io. BufferedReader;

import java. io. InputStreamReader;
```

```
public class Quick_Search

{
```

Το μέγεθος του αλφαβήτου είναι 256, όσοι και οι χαρακτήρες του κώδικα ASCII. Η μεταβλητή ASIZE ισούται με το μέγεθος του αλφαβήτου μείον 1.

```
final static int ASIZE=255;

   public static void main(String[] args)

   {
```

p είναι ο πίνακας χαρακτήρων που κρατά το πρότυπο, ενώ t είναι ο πίνακας χαρακτήρων που κρατά το κείμενο. plen και tlen τα μήκη των αντίστοιχων αλφαριθμητικών, που υπολογίζονται με τη μέθοδο length. Το πέρασμα των αλφαριθμητικών γίνεται με τη μέθοδο readLine που εφαρμόζεται στο αντικείμενο br της κλάσης BufferedReader.

```
      BufferedReader br=new BufferedReader(new InputStreamReader(System.
in));

      String p="";

      String t="";

      int plen, tlen;
```

Ακολουθεί η δημιουργία αντικειμένου qs της κλάσης Quick_Search

```
      Quick_Search qs=new Quick_Search();

      System. out. println("Quick Search Algorithm");

      System. out. println("Enter the pattern: ");

      try {

         p=br. readLine();

      }
```

```
        catch (Exception e) {
        }

        System. out. println("Enter the text: ");
        try {
            t=br. readLine();
        }
        catch (Exception e) {
        }

        plen=p. length();
        tlen=t. length();
```

Καλείται η συνάρτηση QS για το αντικείμενο qs, που θα αναζητήσει το πρότυπο μέσα στο κείμενο.

```
        qs. QS(p, plen, t, tlen);
    }

    static void preQsBc(String x,  int m,  int[] qsBc)
    {
        for (int i = 0; i < ASIZE; i++){
```

Αρχικά, όλα τα στοιχεία του πίνακα qsBc με τα bad character shifts παίρνουν τιμή m+1

```
        qsBc[i] = m + 1;
        }
```

Για κάθε χαρακτήρα βρίσκεται η δεξιότερη εμφάνισή του μέσα στο πρότυπο, πχ αν x[i]='a', ο κωδικός ascii του a είναι 97, άρα qsBc[x[i]] ισοδυναμεί με qsBc[97]

```
        for (int i = 0; i < m; i++){
           qsBc[(int)x. charAt(i)] = m - i;
        }
    }
```

Δημιουργείται ο πίνακας qsBc και καλείται η συνάρτηση προεπεξεργασίας preQsBc ώστε να γεμίσει ο πίνακας qsBc

```
    static void QS(String x,  int m,  String y,  int n)
    {
        int j;
        int qsBc[]=new int[ASIZE];
        preQsBc(x,  m,  qsBc);
        j = 0;
```

Δεν μπαίνει στο βρόχο αν j+1>n-m κι αυτό για να μη βγει από τα όρια του κειμένου.

```
        while (j+1 <= n - m) {
```

Η μέθοδος substring επιστρέφει το υποαλφαριθμητικό του y από τη θέση j ως την j+m-1. Η μέθοδος equals συγκρίνει το x με το υποαλφαριθμητικό και επιστρέφει boolean τιμή

```
          if (x. equals(y. substring(j, j+m)) )
                System. out. println("found pattern at position "+j);
```

Το j μετακινείται δεξιά τόσο όσο η τιμή στη θέση y[j+m] στον πίνακα qsBc, πχ αν ο χαρακτήρας υπάρχει μία φορά στην πρώτη θέση στο πρότυπο, θα γίνουν m μετατοπίσεις ενώ, όταν ο χαρακτήρας y[j+m] δεν υπάρχει στο πρότυπο, γίνονται m+1 μετατοπίσεις

```
        j += qsBc[(int)y. charAt(j + m)];
      }
   }
}
```

7. ΕΠΙΛΟΓΟΣ

Θα μπορούσαμε να ορίσουμε την γενική έννοια του προγραμματισμού, σαν μια σειρά από μελετημένες, αλληλοεξαρτώμενες και πάνω από όλα καλά σχεδιασμένες εντολές. Και πως να μην είναι άλλωστε, εφόσον και ο πιο αρχάριος προγραμματιστής ξεκινάει τα πρώτα του βήματα πάνω σε ένα χαρτί. Βασικός του σκοπός είναι να σχεδιάσει, να υπολογίσει και να μελετήσεις αλγόριθμους οι οποίοι στην συνέχεια υλοποιήθηκαν με την σειρά τους από μια γλώσσα προγραμματισμού. Η βασική εμπειρία που κατακτήθηκε από την συγκεκριμένη πτυχιακή, είναι ότι και οι τρεις γλώσσες που αναλύθηκαν και μελετήθηκαν σε γενικά πλαίσια αλλά και πιο συγκεκριμένα πάνω στις συμβολοσειρές. Είναι ότι όλη η δυσκολία ξεκινάει από τον πρωταρχικό σχεδιασμό ενός αλγόριθμου. Με την φύση της κάθε γλώσσας ο προγραμματιστής προσαρμόζετε, για παράδειγμα η pascal είναι ιεραρχικός προγραμματισμός, σε αντίθεση με την c++ που είναι αντικειμενοστραφής. Στην προσπάθεια να γίνουμε πιο ειδικευμένοι παρουσιάζοντας τρεις γλώσσες προγραμματισμού (pascsal, java, c++) και αναλύοντας πιο συγκεκριμένα το κομμάτι των συμβολοσειρών, γεννήθηκαν και άλλα ερωτήματα. Όπως το πιο απλό! τι είναι τελικά μια συμβολοσειρά;

Συμβολοσειρά είναι το όνομα σας και μαζί και με το επίθετο φτιάχνουν δύο πίνακες και πόσο γρήγορα μπορώ να σε βρω; Να σε εμφανίσω σε αντιγράψω; Εκεί ακριβώς έρχονται οι αλγόριθμοι συμβολοσειρών, που βοηθούν στην μελέτη και ανάπτυξη μιας σειράς εντολών, με βασικό σκοπό την δημιουργία ενός εργαλείου. Αυτός ακριβώς είναι και ο ρόλος όλων των υλοποιημένων προγραμμάτων, στην μετέπειτα διάρκεια ζωής του, χρησιμοποιούνται σαν εργαλεία στα διάφορα υπολογιστικά συστήματα. Γι' αυτό και από την φύση τους έχουν αρχή, μέση, τέλος αλλά και επανάληψη.

Τέλος, η κάθε γλώσσα ξεχωριστά έχει τα δικά της χαρακτηριστικά και το μόνο που μένει σταθερό και στις τρεις γλώσσες που περιέχονται στην συγκεκριμένη πτυχιακή, είναι η ιδεολογία. Σαν μια σταθερή μεταβλητή παραμένει ο σκοπός - στόχος και η κάθε γλώσσα τον αναλύει, τον κατακτά και τον υλοποιεί με τον δικό της τρόπο, εφαρμόζοντας τα δικά της ξεχωριστά χαρακτηριστικά. Αν

μπορούν και οι τρεις μαζί να συνεργαστούν. Το κάθε πρόγραμμα δημιουργείται από συγκεκριμένη γλώσσα και μόνο από αυτήν μεταβάλλεται. Αν δεν υπήρχε η pascal μπορεί να μην υπήρχε και η c++, άρα νοητά οι γλώσσες προγραμματισμού αλληλοσυνδέονται.

Βιβλιογραφία

●Fundamentals of DATA STRUCTURE IN PASCAL, Ellis Horowitz - University of Southern California, Sartaj Sahni - University of Minnesota. Computer Science Press.

●Εισαγωγή στην PASCAL και την Turbo PASCAL, Rodnay Zaks. Εκδότης: Μ. Γκιούρδας.

● Εγχειρίδιο PASCAL. Jacques Tiberghien. Εκδότης: Μ. Γκιούρδας.

●Turbo Pascal: An Introduction to the Art and Science of Programming. Second Edition. Walter J. Savitch - University of California, San Diego. The Benjamin/Cummings Publishing Company, Inc.

●Μάθετε C++ βήμα προς βήμα

Εκδότης: Μ. Γκιούρδας

● Μάθετε την *Java* 2 σε *24 ώρες*, Rogers Cadenhead, Εκδότης: Μ. Γκιούρδας

●Βήμα βήμα Microsoft Office FrontPage 2003 Εκδότης: Κλειδάριθμος

●Java Μια εισαγωγή στην επίλυση προβλημάτων και στον προγραμματισμό

Savitch Walter Εκδότης: Τζιόλα

●Internet

Σελίδες

http://apps.teipir.gr/aut_nz_cpp/lectures/lecture2.pdf
http://cgi.di.uoa.gr/~himiko/Askisis/Ask1.pdf

http://www.java.com/en/download/index.jsp

http://www.it.uom.gr/project/java/tutorial.htm

http://alpha.physics.uoi.gr/web_kokkas_java/kef1.pdf

http://mindview.net/Books/TIJ/DownloadSites

http://cc.embarcadero.com/Free.aspx?id=24728

http://epcmag.gr/forum/index.php?s=db21df016ce0e63350b781a90bbbe279&showtopic=35270

http://www.cplusplus.com/doc/tutorial/

http://www.it.uom.gr/project/cppmanual/cplman/index.htm

http://www.cppreference.com/wiki/

http://www.freetechbooks.com/

http://www.programmingtutorials.com/java.aspx