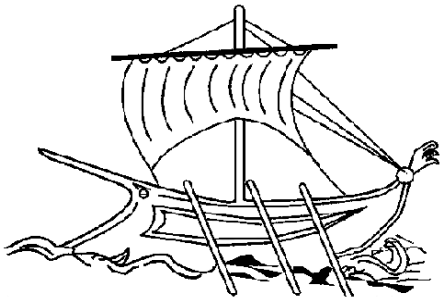
ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΙΡΑΙΑ



Α.Τ.Ε.Ι ΠΕΙΡΑΙΑ

ΑΛΓΟΡΙΘΜΟΣ ΑΥΤΟΜΑΤΗΣ ΣΥΜΠΛΗΡΩΣΗΣ ΚΕΙΜΕΝΟΥ ΜΕ ΧΡΗΣΗ ΣΥΜΠΑΓΩΝ ΔΕΝΤΡΩΝ ΚΑΙ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ

ΤΣΟΛΑΚΟΓΛΟΥ ΙΟΡΔΑΝΗΣ

ΓΙΑΚΟΥΜΑΤΟΣ ΑΡΓΥΡΗΣ

ΣΟΥΤΑΣ ΑΠΟΣΤΟΛΗΣ

ΔΡΟΣΟΣ ΧΡΗΣΤΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΑΥΤΟΜΑΤΙΣΜΟΥ

ΝΟΕΜΒΡΙΟΣ 2017

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η Τσολάκογλου Ιορδάνης ,

του Κυριάκου , με αριθμό μητρώου 38529 φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Επίσης δηλώνω υπεύθυνα ότι έχω παρακολουθήσει το σεμινάριο συγγραφής και εκπόνησης πτυχιακής εργασίας που διοργανώνεται από το Τμήμα Μηχανικών Αυτοματισμού Τ.Ε. κατά το Χειμερινό/Εαρινό Εξάμηνο του Ακ. Έτους 2016 .

Ο Δηλών Ημερομηνία

ΤΣΟΛΑΚΟΓΛΟΥ Ι. **23/11/2017**

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η Γιακουμάτος Αργύριος ,

του Γεράσιμος , με αριθμό μητρώου 41280 φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Επίσης δηλώνω υπεύθυνα ότι έχω παρακολουθήσει το σεμινάριο συγγραφής και εκπόνησης πτυχιακής εργασίας που διοργανώνεται από το Τμήμα Μηχανικών Αυτοματισμού Τ.Ε. κατά το Χειμερινό/Εαρινό Εξάμηνο του Ακ. Έτους 2016 .

Ο Δηλών Ημερομηνία

ΓΙΑΚΟΥΜΑΤΟΣ Α. **23/11/2017**

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η Σούτας Απόστολος ,

του Χαρίτου, με αριθμό μητρώου 38700 φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Επίσης δηλώνω υπεύθυνα ότι έχω παρακολουθήσει το σεμινάριο συγγραφής και εκπόνησης πτυχιακής εργασίας που διοργανώνεται από το Τμήμα Μηχανικών Αυτοματισμού Τ.Ε. κατά το Χειμερινό/Εαρινό Εξάμηνο του Ακ. Έτους 2016 .

Ο Δηλών Ημερομηνία

ΣΟΥΤΑΣ Α. **23/11/2017**

Περιεχόμενα

Κατάλογος εικόνων

Βιβλιογραφια καταλόγου εικόνων

Εικόνα 2.1 http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C101/36/198,1060/

Εικόνα 3.1 http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C101/36/198,1061/

Εικόνα 3.2 http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C101/36/198,1061/

Εικόνα 4.1 http://mcqquestion.blogspot.gr/2012/08/data-structures-and-algorithm-analysis_2686.html

Κατάλογος Πινάκων

Βιβλιογραφία καταλόγου πινάκων

Πίνακας 1.1 https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

Πίνακας 1.2 https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

Πίνακας 1.3 https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

Πίνακας 1.4 https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

Πίνακας 1.5 https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

ΠΕΡΙΛΗΨΗ

Ο κώδικας του προγράμματος αποτελείται από 3 αρχεία .cpp 2 αρχεία .h και το makefile. Συγκεκριμένα τα main.cpp , Node.cpp , Trie.cpp , Node.h και Trie.h .

Αρχίζοντας απ'το πιο συγκεκριμένο , ο κόμβος(Node) του κάθε δέντρου περιέχει ένα value που έχει μέσα ενα char και είναι στην προσομοίωση του trie σαν τη τιμη που έχει η από πάνω του ακμή .Επίσης έχει ένα πίνακα 'παιδιά' με 27 θέσεις όπου βρίσκονται τα παιδιά του . Στις θέσεις 1-26 είναι κάθε παιδί για ένα γράμμα της αγγλικής αλφαβήτου και στην θέση 27 είναι τελικός κόμβος δηλαδή σχηματίζεται λέξη εκεί και έχει το @ μέσα (δηλαδή φύλλο) . Ακόμα, έχει έναν πίνακα Ν θέσεων leafPointers που μέσα έχει δείκτες απευθείας στα φύλλα (με το μεγαλύτερο app_ratio(συχνότητα)).

Στο Trie υπάρχει το root που είναι η ρίζα του δέντρου με value το ' ' και πατέρα έχει τη συνάρτηση add_node η οποία παίρνει σαν ορίσματα όλο το string απ'τη main με την fgets και το βάζει στο δέντρο. Άμα μετά κάποια λέξη υπάρχει ήδη αυξάνει το app_ratio. Κάνει χρήση των συναρτήσεων της κλάσης Node checkChild() που ελέγχει αν υπάρχει συγκεκριμένο παιδί ,της goToChild που επιστρέφει ένα συγκεκριμένο παιδί ,της addChild() που δημιουργεί ένα συγκεκριμένο παιδί,της setLeaf() που δημιουργεί ένα φύλλο στον τελευταίο χαρακτήρα του string ώστε να δημιουργηθεί λέξη , της increase_app_ratio() που αυξάνει το app_ratio(syxnothta) εάν ο κόμβος που εισήχθη υπήρχε ήδη στο trie και τέλος της fixLeafPointers() που αναδρομικά φτιάχνει τα leafPointers του κάθε δέντρου . Επιπρόσθετα παίρνει σαν όρισμα έναν int που λέγεται READ_FROM και άμα είναι 0 σημαίνει ότι το string που πήρε είναι από αρχείο ενώ άμα είναι 1 το πήρε απ'τον χρήστη. Αυτό έχει σημασία γιατί πρώτη φορά που βάζει ο χρήστης μια καινούρια λέξη θέλουμε και να την βάλουμε στο δέντρο και να της κάνουμε το app_ratio 1.

Στη main με την χρήση της getchar_silent() διαβάζει χαρακτήρα χαρακτήρα και έχει περιπτώσεις για το καθένα. Αν πάρει γράμμα (κεφαλαίο ή μικρό) τότε το προσθέτει στο input_buffer που είναι το string της κάθε λέξης και στο line_text που είναι το string ολόκληρης της γραμμής μέχρι να πατηθεί ENTER(\n) που μηδενίζει το line_text . Αν πατηθεί backspace σβήνει χαρακτήρες και αν πατηθεί TAB καλεί την συνάρτηση getTAB της Trie η οποία με τη σειρά της εμφανίζει μέσω της printBestMatch (η οποία καλεί την printFinal που είναι συνάρτηση της κλάσης Node) τις προτεινόμενες λέξεις για την συμπλήρωση της λέξης πριν το TAB . Μετά με τη chooseWord άμα ο χρήστης πληκτρολογήσει έναν αριθμό που αντιστοιχεί στην επιλογή της λέξης που θέλει , του την επιστρέφει καλώντας αυτή με την σειρά της την getWord που είναι συνάρτηση της κλάσης Node η οποία επιστρέφει τη λέξη στην chooseWord που με την σειρά της την επιστρέφει στην main .Έτσι διαγράφουμε αυτό που είχε γραφτεί πριν το TAB και το αντικαθιστούμε με την λέξη που επέλεξε ο χρήστης.

Τέλος έχει μια συνάρτηση freeTrie() που αποδεσμεύει τη μνήμη που πιάνει όλο το δέντρο.

Βιβλιογρφική ανασκόπηση

1. http://www.cs.uoi.gr/~stergios/teaching/plh111/dbindex_files/slides2005/PLH111-7.pdf

2. https://eclass.teicrete.gr/modules/document/file.php/TP329/Θεωρία/Σημειώσεις%20θεωρίας%20Δομών%20Δεδομένων%202015.pdf

3. https://www.ebooks4greeks.gr/2011.Download_free-ebooks/Pliroforikis/glossa_programmatismoy_C++__eBooks4Greeks.gr.pdf

4. Sahnii Sartaj '*Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++', Εκδόσεις Τζιόλα , Θεσσαλονίκη , 2004*

5. Γιαγλής Γεώργιος, '*Αρχές Λειτουργίας & Προγραμματισμού Ηλεκτονικών Υπολογιστών*' , Εκδόσεις ΟΠΑ, Αθήνα , 2012

6. Βακάλη Αθηνά, Γιαννόπουλος Ηλίας, Ιωαννίδης Νέστωρ, Κοίλιας Χρήστος, Μάλαμας Κων/νος , Μανωλόπουλος Ιωάννης , Πολίτης Παναγιώτης , '*Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*' , Ινστιτούτο Τεχνολογίας Υπολογιστών και εκδόσεων ΄Διόφαντος΄ , Αθήνα

C++

Μια συνηθισμένη γλώσσα προγραμματισμού μεταγλωτίζει τον πηγαίο κώδικα του προγράμματος σε εκτελέσιμη μορφή που να καταλαβαίνει ο επεξεργαστής. Αυτού του είδους οι γλώσσες είναι γνωστές ως γλώσσες μηχανής. Τα τελευταία χρόνια έχει αρχίσει να γίνεται ιδιαίτερα δημοφιλής μια ακόμα κατηγορία γλωσσών προγραμματισμού, οι οποίες ονομάζονται αντικειμενοστραφείς γλώσσες, χαρακτηριστικό παράδειγμα εκ των οποίων είναι και η C++ . Ο αντικειμενοστραφής προγραμματισμός αποτελεί μια φιλοσοφία διαφορετική από αυτή του διαδικασιακού, καθώς στηρίζεται στα δεδομένα και όχι στις διαδικασίες.

ΔΟΜΗ ΤΗΣ C++

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.

ΜΕΤΑΒΛΗΤΕΣ

Η μεταβλητή πρόκειται για μια θέση μνήμης που μπορεί να περιέχει δεδομένα οποιουδήποτε είδους. Προτού χρησιμοποιηθεί πρέπει να δηλωθεί, δηλαδή να ενημερωθεί ο compiler για το όνομα τους και τον τύπο τους.Το περιεχόμενό της μπορεί να μεταβάλλεται , εξ'ού και το όνομά της , με άλλα παρόμοια δεδομένα. Οι μεταβλητές ορίζονται ως εξης : type var; όπου type το είδος της μεταβλητής και var το όνομά της. Μια μεταβλητή θεωρείται έγκυρη, δηλαδή μια αναφορά σε αυτήν έχει νόημα μόνο μέσα στο πλαίσιο στο οποίο ορίζεται. Ταυτόχρονα με τη δήλωση μπορεί να γίνει και αρχικοποίηση των μεταβλητών. Η εκχώρηση τιμών γίνεται με το σύμβολο ανάθεσης τιμής =,
όπου εκχωρεί την τιμή ή το αποτέλεσμα των πράξεων που βρίσκονται στα δεξιά του, στην μεταβλητή που βρίσκεται στα αριστερά του.

ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Πίνακας 1.1 Τύποι δεδομένων στη C++

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

Όνομα	Μέγεθος (σε bytes)	Όρια
Float	4	**1**,4*10^-45 έως 3,4*^38
double	8	4,9*10^-324 έως 1,8*^308
boole	1	true/false
wchar	2	-
string	μεταβλητό	-

ΣΤΑΘΕΡΕΣ

Αν και οι μεταβλητές έχουν σχεδιαστεί να μεταβάλλονται, υπάρχουν περιπτώσεις που χρειαζόμαστε κάποιες σταθερές. Μπορούμε να δηλώσουμε μια μεταβλητή ως σταθερά με τη λέξη const.

ΤΕΛΕΣΤΕΣ

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. Οι κυριότερες κατηγορίες στις οποίες χωρίζονται είναι οι παρακάτω .

Πίνακας 1.2 Αριθμητικοί τελεστές

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

Πίνακας 1.3 Σχεσιακοί τελεστές

Σύμβολο	Είδος
==	Ισότητα
!=	Ανισότητα

Σύμβολο	Είδος
>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο από ή ίσο με
<=	Μικρότερ από ή ίσο με

Πίνακας 1.4 Λογικοί τελεστές

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

Πίνακας 1.5 Τελεστές καταχώρησης

Σύμβολο	Είδος
x += y	το ίδιο με x = x + y
x -= y	το ίδιο με x = x – y
x *= y	το ίδιο με x = x * y
x /= y	το ίδιο με x = x ? y
x %= y	το ίδιο με x = x % y
x &= y	το ίδιο με x = x & y
x |= y	το ίδιο με x = x | y
x ^= y	το ίδιο με x = x ^ y

C++

ΠΑΡΑΣΤΑΣΕΙΣ

Οποιοσδήποτε έγκυρος συνδυασμός μεταβλητών , σταθερών , αριθμών και τελεστών καλείται μια παράσταση. Το αποτέλεσμα της παράστασης μπορεί να είναι κάποιος από τους τύπους δεδομένων της C++ (π.χ. int,long,double,bool) ή κάποιο αντικείμενο (π.χ. string).

ΠΡΟΣΠΕΛΑΣΗ

Στη C++ δίνεται η δυνατότητα άμεσης προσπέλασης στα περιεχόμενα της περιοχής της μνήμης που περιέχει κάποια μεταβλητή. Αυτό ίσως φαίνεται κάπως επικίνδυνο αλλά προσφέρει μεγάλες δυνατότητες στον προγραμματιστή να εισχωρήσι βαθιά στη δομή του προγράμματος. Αυτή η πρόσβαση πραγματοποιείται με δυο τρόπους, με pointers και references. Αν έχουμε μια μεταβλητή a τύπου string , μπορούμε να έχουμε άμεσα την αναφορά (reference) στη διεύθυνση μνήμης που περιέχει το αντικείμενο string με τον τελεστή & πριν το όνομα της μεταβλητής. Οι pointers χρησιμοποιούνται κατά κόρον στη δυναμική δημιουργία αντικειμένων.

ΕΛΕΓΧΟΜΕΝΗ ΡΟΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Σε κάθε πρόγραμμα είναι απαραίτητο να ελέγχουμε τη ροή του αναλόγως κάποιες συνθήκες και να την ανακατευθύνουμε κατάλληλα. Υπάρχουν πολλοί τρόποι να κατευθύνουμε τη ροή του προγράμματος και ο κάθε ένας εξυπηρετεί συγκεκριμένο σκοπό.
if : χρησιμοποιείται όταν θέλουμε να εκτελέσουμε ορισμένες εντολές στην περίπτωση μόνο που χρησιμοποιείται κάποια συγκεκριμένη συνθήκη

switch : χρησιμοποιείται όταν έχουμε πολλαπλές επιλογές ή τιμές για μια μεταβλητή και θέλουμε να εκτελεστούν διαφορετικές εντολές για κάθε τιμή
for χρησιμοποιείται για επανάληψη (loop) υπό συνθήκη κάποιων εντολών

while : χρησιμοποιείται όταν δεν μπορούμε να προβλέψουμε εύκολα πόσες φορές θέλουμε να εντελεστούν οι εντολές, ή όταν δεν έχει σημασία ο αριθμός των επαναλήψεων αλλά η ικανοποίηση ή όχι της συνθήκης

do/while : χρησιμοποιείται όταν θέλουμε οι εντολές εντός της επανάληψης να εκτελεστούν τουλάχιστον μια φορά ανεξαρτήτως αν είναι αληθής η συνθήκη ή όχι

break/continue : χρησιμοποιείται όταν θέλουμε να διακόψουμε την εκτέλεση των εντολών σε μια επανάληψη πριν από την προκαθορισμένη στιγμή

C++

ΚΛΑΣΕΙΣ

Όλα τα αντικείμενα που έχουν κοινά χαρακτηριστικά ανήκουν στην ίδια κλάση και κάθε ένα από αυτά λέγεται ότι είναι 'στιγμιότυπο' (instance) της κλάσης.[3] Στη C+ η κλάση δηλώνεται με τη λέξη class. Οι μεταβλητές της κλάσης έχουν διαφορετικές τιμές για κάθε αντικείμενο και κάθε αντικείμενο έχει πρόσβαση μόνο στις δικές του μεταβλητές. Επίσης , ο τύπος δεδομένων που θα επιλεχθεί για κάθε μεταβλητή θα εξαρτάται από το είδος της πληροφορίας που θα κρατάει αυτή η μεταβλητή. Στη C++ συνήθως η δήλωση μιας κλάσης , μεταβλητής ή συνάρτησης γίνεται σε ξεχωριστό αρχείο του οποίου το όνομα λήγει σε .h ή .hpp, ενώ το αρχείο ορισμού (δηλαδή το αρχείο που περιέχει τον πηγαίο κώδικα ορισμού της αντίστοιχης κλάσης, μεταβλητής ή συνάρτησης) λήγει σε .cpp ή .cxx .

ΜΕΘΟΔΟΙ

Οι μέθοδοι σε μια κλάση , δεν είναι παρά συναρτήσεις που προσφέρουν πρόσβαση στα δεδομένα του εκάστοτε αντικειμένου της κλάσης.[3] Η αλληλεπίδραση του χρήστη με κάθε αντικείμενο γίνεται μέσω των μεθόδων της κλάσης οι οποίες καθορίζουν και τον τρόπο χειρισμού των μεταβλητών του αντικειμένου. Έχοντας ορίσει την κλάση μας, θα πρέπει έπειτα να δημιουργήσουμε τα αντικείμενα/ στιγμιότυπα της κλάσης. Στη C++ μπορούμε να δημιουργήσουμε αντικείμενα είτε στατικά είτε δυναμικά. Η στατική δημιουργία γίνεται όπως και ο ορισμός κάποιας μεταβλητής , ενώ η δυναμική γίνεται με τη χρήση της εντολής new της C++. Η new δημιουργεί μια φυσική
αναπαράσταση της κλάσης, ένα μοναδικό στιγμιότυπο, και αν είναι επιτυχής επιστρέφει ένα δείκτη (pointer) σε αυτή, διαφορετικά επιστρέφει μηδέν (0). Με αυτόν τον δείκτη μπορούμε να προσπελάσουμε το αντικείμενο με όποιον τρόπο θέλουμε (και εφόσον το επιτρέπει το ίδιο το αντικείμενο). Ορισμένοι μέθοδοι δέχονται και κάποιες παραμέτρους (εντός παρενθέσεων). Οι παράμετροι μπορεί να είναι μεταβλητές οποιουδήποτε αποδεκτού τύπου στη C++ ή ακόμα και άλλα αντικείμενα. Αν έχουμε περισσότερες από μία παραμέτρους τις διαχωρίζουμε με κόμμα.

ΔΗΜΙΟΥΡΓΟΣ

Όταν δημιουργείται ένα αντικείμενο (στατικά ή δομικά) στην πραγματικότητα η C++ , αφού δεσμεύσει την απαραίτητη μνήμη για το αντικείμενο , εκτελεί μια συγκεκριμένη μέθοδο της κλάσης : τον δημιουργό (constructor).[3] Ο δημιουργός πραγματοποιεί τις απαραίτητες ενέργειες για να καταστεί κάποιο αντικείμενο έτοιμο προς χρήση. Αυτές μπορεί να είναι κάτι απλό όπως η ρύθμιση κάποιων μεταβλητών με αρχικές τιμές ή πιο περίπλοκο όπως η δημιουργία σύνδεσης με μια βάση δεδομένων, η αρχικοποίηση των πινάκων SQL , η δέσμευση κάποιων δικτυακών θυρών(sockets) για κάποιο server ή ακόμα και το άνοιγμα κάποιου παραθύρου για εμφάνιση γραφικής πληροφορίας. Ο δημιουργός έχει τη μορφή μιας μεθόδου με το όνομα της κλάσης και χωρίς τύπο (δηλαδή δεν δηλώνεται ο τύπος δεδομένων που

επιστρέφει). Τέλος, μπορούμε να έχουμε περισσότερους από έναν δημιουργούς οι οποίοι να δέχονται διαφορετικές παραμέτρους ο καθένας.

ΚΑΤΑΣΤΡΟΦΕΑΣ

Στη C++ ό,τι αντικείμενο δημιουργούμε δυναμικά πρέπει να το καταστρέφουμε ελευθερώνοντας όλους τους πόρους που δεσμεύσαμε κατά την ύπαρξή του. Η καταστροφή ενός αντικειμένου γίνεται με τη μέθοδο που λέγεται καταστροφέας (destructor). Το όνομα του είναι το ίδιο με της κλάσης προπορευόμενο από τον τελεστή ~ . Σε αντίθεση με το δημιουργό που είναι απαραίτο να οριστεί, τις περισσότερες φορές δεν ισχύει το ίδιο και για τον καταστροφέα. Κάτι τέτοιο έχει νόημα μόνο όταν στον δημιουργό πραγματοποιούμε κάποια δυναμική λειτουργία την οποία πρέπει να τερματίσουμε κατά την καταστροφή του αντικειμένου.

ΔΕΙΚΤΗΣ ΑΝΑΦΟΡΑΣ

Μέσα σε μια μέθοδο, μπορούμε να χρησιμοποιήσουμε μια μεταβλητή της κλάσης απλώς με το όνομά της , αναφερόμενοι φυσικά στην τιμή που έχει η μεταβλητή για το συγκεκριμένο αντικείμενο. Όταν αυτό δεν είναι αρκετό και απαιτείται πιο ρητή αναφορά χρησιμοποιούμε τη μεταβλητή this που επιστρέφει πάντα ένα δείκτη στο τρέχον αντικείμενο (δηλαδή αυτό που καλεί τη μέθοδο). Ο δείκτης this είναι ιδιαίτερα χρήσιμος ειδικά σε περιπτώσεις που μεταχειριζόμαστε περισσότερα από ένα όμοια αντικείμενα στην ίδια μέθοδο για αποφυγή σύγχυσης.

ΑΛΓΟΡΙΘΜΟΙ

Η έννοια του αλγορίθμου

Ο όρος αλγόριθμος, χρησιμοποιείται για να δηλώσει μεθόδους που εφαρμόζονται για την επίλυση προβλημάτων. Αποτελεί μια καλά προσδιορισμένη διαδικασία η οποία παρέχει τις οδηγίες σύμφωνα με τις οποίες τα δεδομένα ενός προβλήματος μετασχηματίζονται και συνδυάζονται για να προκύψει η λύση του προβλήματος. Κάθε γλώσσα προγρμματισμού χρησιμοποιείται για τη συγγραφή προγραμμάτων τα οποία λύνουν ανθρώπινα προβλήματα. Για να μετασχηματιστεί όμως η λύση που έχει σκεφτεί ένας άνθρωπος σε πρόγραμμα που μπορεί να εκτελεστεί από τον υπολογιστή , ο προγραμματιστής πρέπει να τη μεταφράσει σε μια σειρά βημάτων που πρέπει να ακολουθηθούν για να οδηγήσουν τελικά στην επίλυση του προβλήματος. Αυτή ακριβώς είναι και η έννοια του αλγορίθμου : είναι μια σειρά ενεργειών αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο που καθορίζουν έναν τρόπο επίλυσης ενός προβλήματος. Κάθε αλγόριθμος πρέπει να αποτελείται από ακριβείς οδηγίες και μπορεί να γραφτεί με τη μορφή πολλών διαφορετικών προγραμμάτων , για παράδειγμα σε διαφορετικές γλώσσες προγραμματισμού , απαραίτητο όμως είναι να ικανοποιεί τα επόμενα κριτήρια.

Είσοδος(input). Καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται, όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια άλλων απλών εντολών.

Έξοδος(output). Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν αλλο αλγόριθμο.

Καθοριστικότητα(definiteness). Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της. Λόγου χάριν, μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέ- της λαμβάνει μηδενική τιμή.

Περατότητα(finiteness). Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά υπολογιστική διαδικασία (computational procedure).

Αποτελεσματικότητα(effectiveness). Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μία εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη. [6]

Πολυπλοκότητα αλγορίθμου

Η επίδοση ενός αλγορίθμου , δηλαδή το πόσο αποδοτικός είναι μπορεί να εκτιμηθεί είτε εμπερικά (ή αλλιώς εκ των υστέρων) είτε θεωρητικά (ή αλλιώς εκ

των προτέρων).Ο πρώτος τρόπος εξετάζει το χρόνο επεξεργασίας και την απαιτούμενη μνήμη ενός αλγορίθμου όταν αυτός εφαρμοστεί σε ένα σύνολο δεδομένων. Τα μεινεκτήματα του σχετίζονται με τη δυσκολία πρόβλεψης της συμπεριφοράς του αλγορίθμου σε περίπτωση που αλλαχτούν τα δεδομένα και στη διαφοροποίηση του χρόνου επεξεργασίας ανάλογα τη γλώσσα προγραμματισμού , το υλικό και την εμπειρία και γνώση του χρήστη. Ο δεύτερος τρόπος χρησιμοποιεί τη συνάρτηση κεφαλαίο-Ο (Big-O) η οποία αποτελεί ένα μέτρο της χειρότερης δυνατής περίπτωσης ενός αλγορίθμου και μετράει την ταχύτητα αυτού όσο το πλήθος των δεδομένων αυξάνει. Η τυπική της μορφή είναι αυτή μιας αλγεβρικής παράστασης η οποία προσδιορίζει των αριθμό των βημάτων που εκτελεί ο αλγόριθμος , δηλαδή την ταχύτητα του, όταν εφαρμόζεται σε μεγάλο αριθμό δεδομένων. Υπάρχουν διάφορες περιπτώσεις Big-O όπως για παράδειγμα Η συνάρτηση Big-O χρησιμοποιείται επίσης και στην εκτίμηση της αναγκαίας μνήμης για την υλοποίηση ενός αλγορίθμου. Η θεωρική προσέγγιση έχει το πλεονέκτημα ότι δεν εξαρτάται από τη γλώσσα προγραμματισμού ή το μεταφραστή αλλά ούτε και από το υλικό του Η/Υ ή τις ικανότητες του προγραμματιστή . [2]

Τρόποι αναπαράστασης

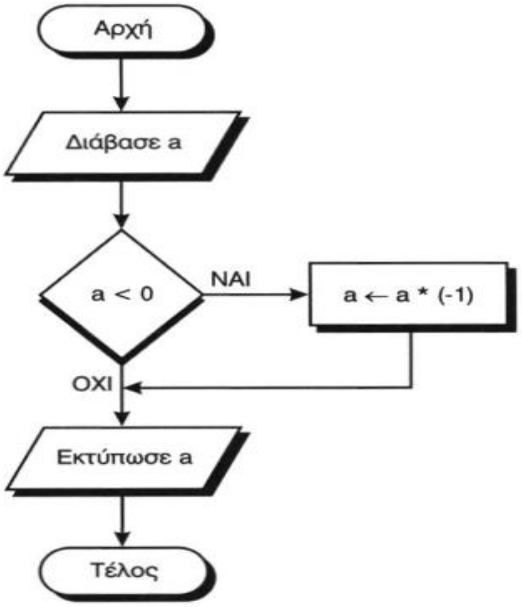
Οι τρόποι που χρησιμοποιούνται για να περιγραφεί ένας αλγόριθμος είναι οι παρακάτω.

Ελεύθερο κείμενο(free text): Αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Οι εντολές εκφράζονται με απλή καθημερινή γλώσσα και ελεύθερη ροή κειμένου. Η χρήση του εγκυμονεί τον κίνδυνο να προκύψει μη εκτελέσιμος αλγόριθμος δηλαδή να οδηγεί σε παραβίαση του κριτηρίου της αποτελεσματικότητας.

Φυσική γλώσσα: Ο αλγόριθμος εκφράζεται με τη χρήση απλής γλώσσας στην οποία οι προτάσεις έχουν διαχωριστεί σε παραγράφους-βήματα και έχουν αριθμηθεί. Εγκυμονεί κίνδυνος παραβίαση του κριτηρίου της καθοριστικότητας.

Διαγραμματικές τεχνικές: Γραφικός τρόπος παρουσίασης ενός αλγορίθμου. Η πιο γνωστή τεχνική είναι το διάγραμμα ροής στην οποία χρησιμοποιούνται ειδικά γεωμτρικά σχήματα που το καθένα δηλώνει μια συγκεκριμένη απλή ενέργεια ή λειτουργία του αλγορίθμου. Τα σχήματα ενώνονται μεταξύ τους με βέλη που δηλώνουν τη σειρά εκτέλεσης των ενεργειών.

ΑΛΓΟΡΙΘΜΟΙ



Εικόνα 2.1 Παράδειγμα αναπαράστασης αλγορίθμου με διαγραμματική τεχνική

Κωδικοποίηση: Ένα πρόγραμμα γραμμένο είτε σε μία ψευδογλώσσα είτε σε κάποια γλώσσα προγραμματισμού που όταν εκτελεστεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

Παρόλο που η ορθότητα του αλγορίθμου εξαρτάται από την ικανότητα του να βρίσκει όλες τις πιθανές λύσεις ενός προβλήματος, η πολυπλοκότητά του ορίζει το βαθμό απλότητας της λύσης που υλοποιεί. Η αναπαράσταση του αλγορίθμου σε γλώσσα κατανοητή από τον υπολογιστή ονομάζεται πρόγραμμα.

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Ως δομή δεδομένων (data structure) ορίζεται ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών. Τα δεδομένα αυτά δεν χρειάζεται να είναι του ίδιου τύπου παρόλο που μπορούν να χειριστούν όλα μαζί σαν ομάδα . Μια από τις πιο σημαντικές χρήσεις που έχουν οι δομές είναι να δημιουργούν νέες μορφές δεδομένων όπως για παράδειγμα στοίβες,ουρές κλπ . Αυτές οι νέες μορφές σχηματίζονται από δομές συνδεδεμένες μεταξύ τους.

Προσπέλαση (access): Πρόσβαση σε ένα κόμβο με σκοπό να εξετασθεί ή να τροποποιηθεί το περιεχόμενό του.

Εισαγωγή (insertion): Η προσθήκη νέων κόμβων σε μία υπάρχουσα δομή.

Διαγραφή (deletion): Αποτελεί το αντίστροφο της εισαγωγής, δηλαδή ένας κόμβος αφαιρείται από μία δομή.

Αναζήτηση (searching): Προσπελαύνονται οι κόμβοι μιας δομής, προκειμένου να εντοπιστούν ένας ή περισσότεροι που έχουν μια δεδομένη ιδιότητα.

Ταξινόμηση (sorting): Οι κόμβοι μιας δομής διατάσσονται κατά αύξουσα ή φθίνουσα σειρά

Αντιγραφή (copying): ‘Ολοι οι κόμβοι ή μερικοί από τους κόμβους μίας δομής αντιγράφονται σε μία άλλη δομή.

Συγχώνευση (merging): Δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή.

Διαχωρισμός (separation): Αποτελεί την αντίστροφη πράξη της συγχώνευσης. [6]

Λίστες

Απλά συνδεδεμένη λίστα

Όταν αναφερόμαστε σε απλά συνδεδεμένες δυναμικές λίστες εννοούμε μια συνεχόμενη αναπαράσταση στοιχείων τα οποία ονομάζονται κόμβοι της λίστας. Κάθε κόμβος συνδέεται με τον επόμενό του μέσω μιας μεταβλητής δείκτη που δείχνει από τον τρέχοντα κόμβο της λίστας στον επόμενο. Η θέση του πρώτου κόμβου (κεφαλή) πρέπει να είναι συνέχεια γνωστή και από εκεί μπορούμε να διατρέξουμε σε όλη τη λίστα. Για το λόγο αυτό ορίζεται μια μεταβλητή δείκτης που δείχνει πάντα αυτόν τον πρώτο κόμβο. Εάν θελήσουμε να διατρέξουμε τη λίστα θα πρέπει να χρησιμοποιηθεί ένας άλλος δείκτης που αρχικά μεν θα τοποθετηθεί στην κεφαλή και στη συνέχεια θα δείχνει από τον έναν κόμβο στον άλλον. Ο δείκτης στον τελευταίο

κόμβο της λίστας έχει μια ειδική τιμή “null” που ορίζει ότι ο δείκτης αυτός δε δείχνει πουθενά.

Διπλά συνδεδεμένη λίστα

Σε κάθε κόμβο μιας διπλά συνδεδεμένης λίστας υπάρχουν δεδομένα, ένας δείκτης προς τον επόμενο κόμβο αλλά και ένας δείκτης προς τον προηγούμενο. Τα πλεονεκτήματα μιας τέτοιας λίστας είναι ότι μπορεί να διαβαστεί και προς τις δύο κατευθύνσεις , δηλαδή δεν είναι απαραίτητο να γνωρίζουμε την κεφαλή αλλά αρκεί να έχουμε πρόσβαση σε οποιοδήποτε κόμβο, και ότι σε περίπτωση που καταστραφούν οι σύνδεσμοί προς τη μία κατεύθυνση τότε μπορούμε χρησιμοποιώντας τους άλλους συνδέσμους να διατρέξουμε και πάλι όλη τη λίστα.

Λειτουργίες στις λίστες

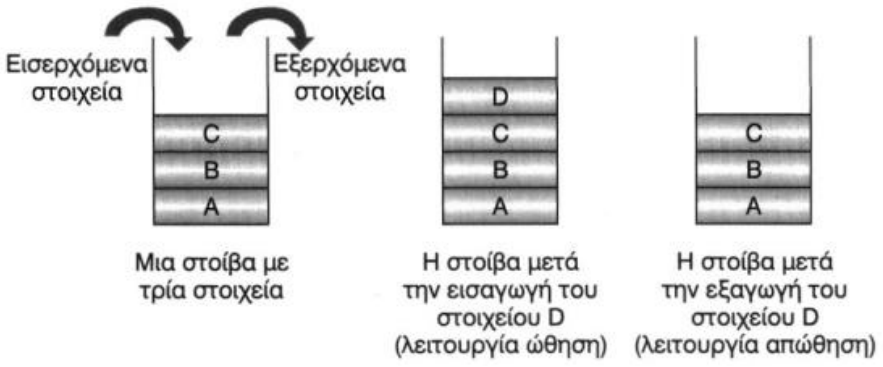
1. Εισαγωγή στοιχείου: Γίνεται αλλάζοντας το δείκτη του στοιχείου μετά από το οποίο θα γίνει εισαγωγή έτσι ώστε ο δείκτης αυτός να δείχνει το στοιχείο που πρόκειται να εισαχθεί
2. Διαγραφή στοιχείου: Γίνεται αλλάζοντας τον δείκτη κάποιου στοιχείου έτσι ώστε να δείχνει το μεθεπόμενο στοιχείο της λίστας ,παρακάμπτοντας δηλαδή το υπό διαγραφή στοιχείο
3. Συνένωση: Μπορούμε να συνδέσουμε δυο λίστες σε μια κάνοντας τον τελευταίο δείκτη της πρώτης να δείξει στην κεφαλή της δεύτερης
4. Αναζήτηση: Αναζήτηση κάποιου στοιχείου μέσα από τη λίστα με βάση κάποια κριτήρια. Απαιτείται να διασχίσουμε τη λίστα με τη βοήθεια των δεικτών
5. Αντιστροφή: Το πρώτο στοιχείο της λίστας να γίνειι τελευταίο κλπ
6. Μετακίνηση: Η μετακίνηση ενός κόμβου από μια θέση σε μια άλλη γίνεται αλλάζοντας τις διευθύνσεις στους κόμβους.

Στοίβες

Μια στοίβα (stack) είναι μια ειδική μορφή λίστας στην οποία τα στοιχεία εισάγονται και εξάγονται μόνο από το ένα άκρο της, το οποίο ονομάζεται κορυφή της στοίβας. Ο τρόπος αυτός εισαγωγής και εξαγωγής στοιχείων ονομάζεται LIFO(last in, first out). Μια στοίβα ορίζεται με τον ίδιο τρόπο που ορίζεται και μια οποιαδήποτε απλά συνδεδεμένη λίστα: η διαφορά έγκειται στο χειρισμό της από το πρόγραμμα. Ο αλγόριθμος εισαγωγής στοιχείου (ώθηση push) σε στοίβα μοιάζει με της λίστας αλλά είναι απλούστερος αφού γίνεται πάντα στην κορυφή της. Το ίδιο ισχύει και για τη

διαγραφή στοιχείου (ανάκληση pop) που επίσης γίνεται πάντα από την κορυφή της στοίβας. Έτσι, πρέπει , αφού ανακληθεί η πληροφορία του αρχικού κόμβου, να αλλάξει η τιμή του δείκτη της στοίβας ώστε να δείχνει εκεί που έδειχνε η παλιά κορυφή της και να απελευθερωθεί η μνήμη που καταλάμβανε η παλιά κορυφή. Η μόνη οριακή περίπτωση είναι να είναι άδεια η στοίβα οπότε φυσικά και δεν μπορεί να γίνει διαγραφή. Οι στοίβες έχουν ευρύτατη εφαρμογή στους υπολογιστές. Ένα από τα πιο χαρακτηριστικά παραδείγματα είναι στην περίπτωση κλήσης υποπρογραμμάτων. Σε κάθε κλήση υποπρογράμματος αποθηκεύεται η διεύθυνση επιστροφής σε μια στοίβα

απ'όπου ανακαλείται όταν τελειώσει η εκτέλεση του υποπρογράμματος και πρέπει να επιστρέψει η εκτέλεση του προγράμματος στο σημείο όπου διακόπηκε. Έτσι, ακόμα και μετά από πολλές διαδοχικές κλήσεις υποπρογραμμάτων, ο έλεγχος επιστρέφει κάθε φορά στο σημείο της τελευταίας κλήσης.



Εικόνα 3.1 Παράδειγμα εισαγωγής κι εξαγωγής στοιχείου σε στοίβα

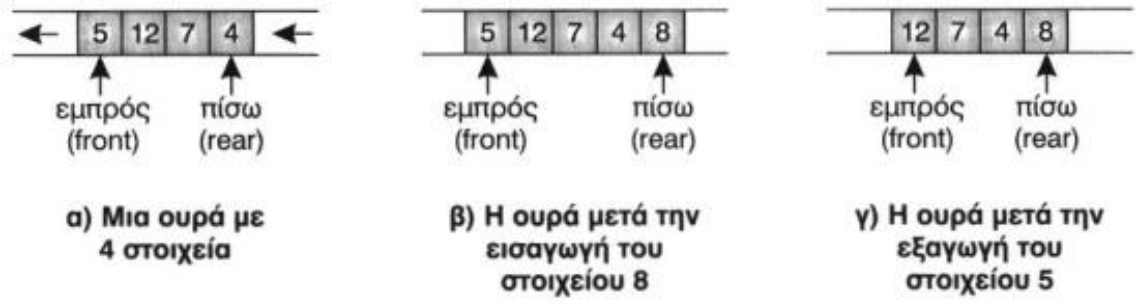
Υλοποίηση στοίβας με πίνακα

Στην περίπτωση αυτή , η στοίβα αποτελείται από δυο μεταβλητές : ένα πίνακα Ν θέσεων στον οποίο αποθηκεύονται τα στοιχεία της στοίβας και μια ακέραια μεταβλητή η οποία αναπαριστά την κορυφή της στοίβας, δηλαδή δείχνει πόσα στοιχεία έχει. Η στατική αυτή υλοποίηση της στοίβας έχει το πλεονέκτημα της ευκολότερης υλοποίησης των περισσότερων πράξεων. Μειονεκτεί όμως αντίθετα σε ότι μειονεκτούν όλες οι στατικές δομές: αν το μέγεθος Ν είναι μεγάλο, τότε οι περισσότερες θέσεις του πίνακα θα παραμένουν άδειες, ενώ αν είναι μικρό ο πίνακας θα γεμίσει και δεν θα είναι εφικτές άλλες εισαγωγές.

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Ουρές

Μια ουρά (queue) είναι μια ειδική μορφή λίστας στην οποία τα στοιχεία εισάγονται από το ένα άκρο και εξάγονται από το άλλο. Αυτή η ιδιότητα της ουράς την κάνει γνωστή ως δομή FIFO (first in , first out). Ο ευκολότερος τρόπος ορισμού μιας ουράς είναι να οριστεί μια απλά συνδεδεμένη λίστα με έναν επιπλέον δείκτη που να δείχνει κάθε φορά στο τελευταίο στοιχείο της. Βέβαια, αυτό δεν είναι υποχρεωτικό και η ουρά μπορεί να οριστεί σαν μια απλά συνδεδεμένη λίστα η οποία να διασχίζεται μέχρι τον τελευταίο κόμβο κάθε φορά που γίνεται μια εισαγωγή.



Εικόνα 3.2 Παράδειγμα εισαγωγής κι εξαγωγής στοιχείου σε ουρά.

Υλοποίηση ουράς με πίνακα

Στην περίπτωση αυτή , η ουρά αποτελείται από τρεις μεταβλητές: ένα πίνακα Ν θέσεων στον οποίο αποθηκεύονται τα στοιχεία της ουράς και δυο ακέραιες μεταβλητές οι οποίες αναπαριστούν την αρχή και το τέλος της ουράς.Όπως ισχύει και στις στοίβες , όταν μια ουρά υλοποιείται με τη χρήση πίνακα κάποιες ιδιότητες του πίνακα παύουν να ισχύουν. Έτσι, δεν επιτρέπεται η τυχαία προσπέλαση οποιουδήποτε στοιχείου του. Τα στοιχεία θα τοποθετούνται μόνο στο ένα άκρο και θα ανακαλούνται από το άλλο. Μετά την ανάκληση κάποιου στοιχείου, αυτό εξακολουθεί να υπάρχει στην ίδια θέση του πίνακα όπου βρισκόταν και πρίν την ανάκληση όμως επειδή ο πίνακας υλοποιεί μια ουρά δεν έχουμε την δυνατότητα να το πάρουμε πίσω. Στην εισαγωγή στοίχειων σε μια ουρά και στην ανάκληση τους από αυτή γίνεται πρώτα έλεγχος υπερχείλισης και έλεγχος επάρκειας αντίστοιχα προκείμενου να αποκλειστεί το ενδεχόμενο μιας γεμάτης ή μιας κενής ουράς.

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

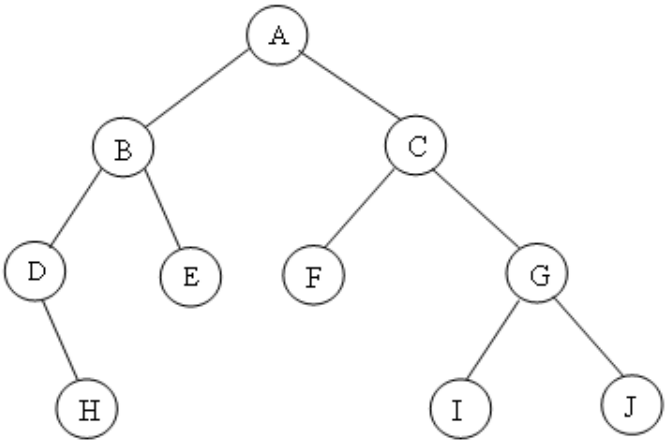
Κυκλική ουρά

Οι ουρές αυτές έχουν τη μορφή ενός δακτυλίου (ring). Στην πράξη δηλαδή είναι σαν να έχουμε για μονάδα αποθήκευσης ένα πίνακα του οποίου το τέλος είναι ενωμένο με την αρχή. Από αυτό το γεγονός προκύπτει και το όνομα κυκλική ουρά(circular queue) . Έτσι μπορούν να τοποθετούνται συνεχώς στοιχεία στην ουρά αφού αφαιρούνται άλλα με προσοχή ώστε να μην επικαλυφθούν τα πρώτα από τα τελευταία στοιχεία της ουράς.

Ουρά προτεραιότητας

Στο είδος αυτό της ουράς ανακαλείται κάθε φορά το στοιχείο το οποίο έχει τη μεγαλύτερη προτεραιότητα. Αν η προτεραιότητα όλων των στοιχείων είναι η ίδια τότε η λειτουργία είναι αυτή της κοινής ουράς. Είναι φανερό ότι κάθε στοιχείο της ουράς θα συνοδεύεται από κάποια πρόσθετη πληροφορία που θα καθορίζει την προτεραιότητα του.

ΔΕΝΤΡΑ



Εικόνα 4.1 Παράδειγμα δέντρου αλγορίθμου

Ως δέντρο ορίζεται ένα σύνολο κόμβων που συνδέονται με ακμές. Ο αρχικός κόμβος ονομάζεται ρίζα και από αυτόν ξεκινούν μηδέν, μια ή περισσότερες ακμές ενώ δεν καταλήγει καμία. Όταν από τον κόμβο Α ξεκινούν ακμές προς τους κόμβους Β και Γ , ο Α ονομάζεται πατέρας των Β και Γ , οι οποίοι είναι τα παιδιά του. Ο αριθμός των παιδιών ενός κόμβου ονομάζεται βαθμός του κόμβου. Οι κόμβοι που δεν έχουν παιδιά ονομάζονται φύλλα ή τερματικοί κόμβοι και συνεπώς έχουν βαθμό μηδέν. Οι μη τερματικοί κόμβοι λέγονται εσωτερικοί (internal) ή κλαδιά (branches). Ο μέγιστος βαθμός των κόμβων ενός δέντρου ονομάζεται βαθμός του δέντρου. Ένα δέντρο ονομάζεται πλήρες(complete) όταν περιέχει τον μέγιστο αριθμό κόμβων ο οποίος εξαρτάται από το ύψος και το βάθος του. Τέλος, το ύψος ή βάθος ενός δέντρου είναι ο μέγιστος αριθμός κόμβων από τη ρίζα προς τα φύλλα του. Ένα δέντρο είναι δυνατόν να έχει πολλά υποδέντρα (subtrees) αν θεωρήσουμε πέραν της αρχικής πραγματικής ρίζας έναν άλλον κόμβο ως νέα ρίζα. Τα δέντρα αποτελούν μια ειδική κατηγορία συνδεδεμένης λίστας που αντιμετωπίζει όμως το μειονέκτημα της αναδρομικότητας των συναρτήσεων που χρησιμοποιούνται. Κάτι τέτοιο φαντάζει λογικό αν αναλογιστεί κανείς ότι από τη φύση τους τα δέντρα είναι μια αναδρομική δομή δεδομένων καθώς κάθε υπόδεντρο μπορεί να θεωρηθεί ως δέντρο. Τα πιο συνηθισμένα δέντρα είναι τα δυαδικά (binary) , τα τριαδικά (ternary) , και τα τετραδικά (quadtree). [1]

ΔΕΝΤΡΑ

Δυαδικά δέντρα

Τα δυαδικά δέντρα ονομάζονται έτσι γιατί έχουν βαθμό δύο , δηλαδή όλοι οι κόμβοι τους έχουν μέχρι δύο παιδιά ο καθένας. Αποτελούν μια σημαντική και ενδιαφέρουσα μορφή καθώς χρησιμεύουν στη δημιουργία μοντέλων συνδυασμού δύο στοιχείων, τα οποία παράγουν ένα νέο το οποίο με τη σειρά του και τη βοήθεια ενός άλλου παράγει κι άλλο νέο κοκ.

Διάσχιση

Μια από τις λειτουργίες που συχνά εφαρμόζονται στα δυαδικά δέντρα είναι η διάσχιση (traverse). Η διάσχιση όλων των κόμβων ενός δυαδικού δέντρου , δηλαδή η επίσκεψη όλων των κόμβων του από ακριβώς μια φορά τον καθένα , μπορεί να γίνει με τρεις τρόπους.

1. Προδιατεταγμένη (preorder) διάσχιση : επισκεπτόμαστε πρώτα τη ρίζα , έπειτα το αριστερό υπόδεντρο και τέλος το δεξί υπόδεντρο.
2. Ενδοδιατεταγμένη (inorder) διάσχιση : επισκεπτόμαστε το αριστερό υποδέντρο, έπειτα τη ρίζα και τέλος το δεξί υποδέντρο
3. Μεταδιατεταγμένη (postorder) διάσχιση : επισκεπτόμαστε πρώτα το αριστερό υποδέντρο, έπειτα το δεξί υποδέντρο και τέλος τη ρίζα [5]

Δυαδικά δέντρα αναζήτησης

Μια ειδική κατηγορία δυαδικών δέντρων είναι τα δυαδικά δέντρα αναζήτησης που χρησιμοποιούνται στη γρήγορη εύρεση τιμών σε διατεταγμένα σύνολα. Η τιμή κάθε κόμβου τους είναι μεγαλύτερη από τις τιμές όλων των κόμβων που ανήκουν στο υποδέντρο του αριστερού τους παιδιού και μικρότερη από τις τιμές όλων των κόμβων που ανήκουν στο υποδέντρο του δεξιού τους παιδιού. Εάν διασχίσουμε ένα δυαδικό δέντρο αναζήτησης με τον ενδοδιατεταγμένο τρόπο το αποτέλεσμα που θα προκύψει θα είναι ταξινομημένο. Μια ειδική μορφή των διαδικών δέντρων αναζήτησης είναι τα δέντρα AVL. Σε ένα τέτοιο δέντρο κάθε κόμβος είναι ισοζυγισμένος κατά το ύψος , δηλαδή τα δυο υπόδεντρα κάθε κόμβου διαφέρουν ως προς το ύψος το πολύ κατά 1. Ο παράγοντας ισοζύγισης ενός κόμβου ισούται με τη διαφορά που έχει το ύψος του δεξιού του υπόδεντρου μείον το ύψος του αριστερού του. Για τα δέντρα AVL αυτό το αποτέλεσμα είναι πάντα -1,0 ή 1. Όταν ο παράγοντας ισοζύγισης είναι ίσος με -1 το δέντρο είναι ‘αριστοβαρές’ ενώ όταν είναι +1 ‘δεξιοβαρές’. Η εξισορρόπηση ενός τέτοιου δέντρου σε περίπτωση που πάψει να υφίσταται λόγω εισαγωγής ενός νέου κόμβου ή διαγραφή ενός άλλου, επιτυγχάνεται με τις κατάλληλες περιστροφές των

κόμβων του. Δηλαδή με αυτό το τρόπο έχουμε τη δυνατότητα αν ένα δέντρο πάψει να είναι AVL να το ξανακάνουμε.

Κοκκινόμαυρα δέντρα αναζήτησης

Αποτελούν ένα ακόμα είδος δυαδικών δέντρων. Κάθε κόμβος χαρακτηρίζεται από ένα χρώμα με τέτοιο τρόπο ώστε να ικανοποιούνται ορισμένες συνθήκες και περιορισμοί. Κατά την εισαγωγή ή διαγραφή κόμβου , οι κόμβοι του δέντρου ξαναχρωματίζονται ώστε αυτές οι συνθήκες και περιορισμοί να εξακολουθούν να ισχύουν. Προφανώς λοιπόν σε κάθε κόμβο κρατείται μια επιπλέον πληροφορία από όσες ξέρουμε για τα απλά δέντρα και αυτή αφορά το χρώμα του κόμβου. Κάθε κόμβος είναι ή κόκκινος ή μαύρος εκτός της ρίζας που είναι πάντα μαύρη. Κανένας κόκκινος κόμβος δεν έχει κόκκινο παιδί. Τα χρώματα των κόμβων απαιτούν ένα επιπλέον bit για κάθε κόμβο, ενώ τα χρώματα των δεικτών απαιτούν δύο. Καθώς και τα δυο σχήματα απαιτούν περίπου τον ίδιο χώρο, μπορούμε να επιλέξουμε μεταξύ τους στη βάση των αποτελεσμάτων από πραγματικές εκτελέσεις αλγορίθμων για ΚΜΔ. Κάθε μονοπάτι από τη ρίζα προς εξωτερικό κόμβο περνάει από τον ίδιο αριθμό μαύρων κόμβων. Τα ΚΜΔ χρησιμοποιούνται όπως και τα AVL για να μειώσουν το ύψος ενός δέντρου και να το φέρουν όσο πιο κοντά γίνεται στο πλήρες. Τα AVL είναι περισσότερο ισορροπημένα σε σχέση με τα ΚΜΔ όμως σε εφαρμογές με πολλές εισαγωγές και διαγραφές κόμβων επειδή μπορεί να χρειαστούν πολλές περιστροφές προτιμώνται τα ΚΔΜ. Η αναζήτηση σε ένα ΚΜΔ μπορεί να γίνει χρησιμοποιώντας τον κώδικα αναζήτησης σε ένα σύνηθες δυαδικό δέντρο αναζήτησης. [4]

Νηματικά δέντρα

Σε ένα δυαδικό δέντρο υπάρχουν δείκτες εκ των οποίων κάποιοι έχουν τιμή NULL. Οι δείκτες αυτοί είναι δυνατό να αξιοποιηθούν με τέτοιο τρόπο ώστε να μην έχουν αυτή τη τιμή αλλά να δείχνουν σε συγκεκριμένους κόμβους μέσα στο δέντρο. Για τον σκοπό αυτό, ο δεξιός δείκτης κάθε κόμβου όταν έχει τιμή μετατρέπεται έτσι ώστε να δείχνει στον επόμενο κόμβο του δέντρου λαμβάνοντας υπόψην τον ενδιαδιατεταγμένο τρόπο διάσχισης. Ένα τέτοιο δέντρο λέγεται δεξιό ενδονηματικό. Το αριστερό ενδονηματικό δέντρο υλοποιείται με παρόμοιο τρόπο αλλά εδώ όταν ο αριστερός δείκτης κάθε κόμβου έχει την τιμή NULL μετατρέπεται έτσι ώστε αντί για τον επόμενο να δείχνει στον προηγούμενο κόμβο του δέντρου. Οι δυο τρόπου μπορούν να χρησιμοποιηθούν και ταυτόχρονα και τότε το δέντρο που προκύπτει λέγεται απλά ενδονηματικό. Με ανάλογο τρόπο δημιουργούνται και τα προνηματικά και μετανηματικά δέντρα αν λυφθεί υπόψην η προδιατεταγμενη ή η μεταδιατεταγμένη διάσχιση αντίστοιχα.

ΔΕΝΤΡΑ

Τετραδικά δέντρα

Εκτός από τα δυαδικά δέντρα υπάρχουν και δέντρα με περισσότερα από δύο παιδιά ανά κόμβο. Μια ειδική κατηγορία τέτοιων δέντρων είναι τα τετραδικά, τα οποία έχουν ευρεία χρήση σε εφαρμογές γραφικών και επεξεργασίας εικόνας. Στις εφαρμογές αυτές χρειαζόμαστε συχνά την αποθήκευση και επεξεργασία σχημάτων,εικόνων κλπ.

Ψηφιακά δέντρα

Τα ψηφιακά δέντρα έχουν την ιδιαιτερότητα ότι οι κόμβοι τους έχουν τιμές που ανήκουν σε ένα συγκεκριμένο σύνολο τιμών. Η πληροφορία η οποία μεταφέρεται στα δέντρα αυτά σχηματίζεται από τη διαδρομή που ακολουθούμε αν διατρέξουμε το δέντρο ξεκινώντας από τη ρίζα μέχρι κάποιο φύλλο του. Ένα σημαντικό πλεονέκτημα των συγκεκριμένων δέντρων είναι ότι μπορούν να αποθηκεύσουν πληροφορίες μεταβλητού μήκους. Μια υλοποίηση των ψηφιακών δέντρων που χρησιμοποιείται για αλφαβητικά κυρίως δεδομένα είναι η δομή trie. Στις δομές αυτές τα φύλλα του δέντρου αποτελούν τους κόμβους πληροφορίας ενώ οι εσωτερικοί κόμβοι είναι απλώς κλαδιά του δέντρου. Στους κόμβους των δέντρων trie υπάρχουν μόνο δείκτες προς συγκεκριμένες θέσεις και η πληροφορία εξάγεται έμμεσα από τη θέση στην οποία δείχνει ο δείκτης. Είναι φανερό ότι αν οι καταχωρημένες ‘τιμές’ είναι λίγες και αραιές τότε υπάρχει μεγάλη σπατάλη χώρου και έτσι η δομή αυτή είναι μη συμφέρουσα.

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Δομή συμπαγούς δέντρου

Κώδικας trie.h

```
1. #ifndef TRIE_H
2. #define TRIE_H
3. #include "Node.h"
4. #include <string.h>
5. #include <stdlib.h>



6. class Trie
7. {
8. node_ptr root;           //riza dentrou pou 8a xei value to " "
9. int i , show_flag;      //i voh8itikh metavlhth gia loops kai to show_flag mas
   voh8aei sthn printBestMatch na doume an dhmiourgh8hke ekeinh thn wra h
   le3h kai na ektypwsoume oti dn yparxoun kapoies p na tairiazoun
10. node_ptr temp;   // mas voh8aei sthn prospelash tou dentrou
11. public:
12. Trie();                  //constructor
13. ~Trie();
14. void add_node(char*,int);       //3ekinaei ap th riza kai phgainei mexri ton
    teleutaio xarakthra tou dentrou ,dhmiourgwntas komvous an dn yparxoun , an
    h le3h yparxei hdh ths au3anei to app_ratio , epishs kalei kai synarthsh pou
    ftiaxnei ta leafPointers
15. void printBestmatch(char *);  // kalontas mia synarthsh ths node emfanizei ta
    kalytera apotelesmata  gia th le3h p pathsame tab
16. void checkTAB(char * str);    //elegxei an yparxoun le3eis gia auto p exei
    path8ei tab kai an yparxoun kalei thn printBestMatch()
17. char * chooseWord(int);       ////ama meta to tab plhktrologh8ei ari8mos
    dialegei thn kalyterh epilogh kai thn epistrefei
18. void freeTrie (node_ptr);    //Diagrafh olou tou dentrou anadromika
19. };



20. #endif // TRIE_H
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Κώδικας trie.cpp

```
1. #include <iostream>
2. #include "Trie.h"



3. using namespace std;



4. Trie::Trie()     //ftiaxnetai to root ston constructor tou trie
5. {
6. root = new Node(' ');   //to value tou einai to whitespace
7. root->set_parent(NULL); //o pateras tou oot einai NULL


8. }



9. Trie::~Trie(){     //apodesmeush xwrou
10. delete temp;
11. this->freeTrie(root);
12. }



13. void Trie::add_node(char * str , int READ_FROM){  //to READ_FROM an
    exei to 0 shmainei oti diavase apo arxeio , an exei 1 shmainei oti diavase apo
    ton xrhsth
14. i=0;
15. temp = root;            //kanoume to temp iso me to root wste na prospelasoume
    to dentro
16. int flag = 0;
17. while(i < (int)strlen(str)){
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
18. if (!temp->checkChild( (int)str[i])  ) {     //an yparxei hdh to paidi me auto to
    value paei se auto
19. temp = temp->goToChild((int)str[i]);     //
20. }                                        //
21. else{                                    // alliws to dhmiourgei kai phgainei se auto
22. temp->addChild(str[i]);                  //
23. temp = temp->goToChild((int)str[i]);     //
24. flag=1;                                  //
25. }
26. i++;
27. }
```

```
28. if(flag == 0 && temp->checkChild(123)){  // Periptwsh pou ws kainouria le3h
    perastei ena substring mias le3hs pou yparxei hdh.. px yparxei to "abc" kai
    mpainei to "ab",to 123 to pairname giati 123-97 8a exoume to 26 pou 8eloume
    thn 8esh 26 tou pinaka paidia
29. for(i=0;i<26;i++)  {             // elegxei mono tis prwtes 26  8eseis tou paidia
    dhladh ta grammata
30. if(!temp->checkChild(i+97) )    //+97 giati h synarthsh mesa to kanei -97 pou
    einai o ascii kwdikos tou a
31. flag = 1;       }            //kanei to flag 1 dhladh 8a ftia3ei fyllo
32. }
```

```
33. if(flag == 1)            //An to flag einai ena shmainei oti exei dhmiurgh8ei
    toulaxiston enas komvos synepws yparxei ki allh telikh le3h dhladh leaf
34. temp->setLeaf();
35. else                     //Ama einai 0 tote shmainei oti h le3h yparxei hdh ara apla
    au3anei to appearence ratio ths
36. temp->increase_app_ratio();
```

```
37. if(flag == 1 && READ_FROM == 1)  //An molis ftiaxthke mia le3h kai hr8e
    ap to plhktrologio kanei to app_ratio me thn mia 1
38. temp->increase_app_ratio();
```

```
39. temp->fixLeafPointers(temp->goToChild('@'));    //kalei thn synarthsh
    fixeleafPointers h opoia anadromika ftiaxnei tous leafPointers(deiktes
    apeu8eias sta fylla) tou ka8e komvou
```

```
40. }
```

```
41. void Trie::checkTAB(char * str){        //an paththei TAB
42. i=0;
43. temp = root;
44. show_flag = 0;
45. while(i < (int)strlen(str)){           //An den yparxei hdh auth h le3h pou exei
    path8ei tab tote kanei to show_flag 1 kai leei sthn printBestMarch p kalei na
    emfanisei to sxetiko mynhma
```

```
46. if (!temp->checkChild( (int)str[i])  ) {
47. temp = temp->goToChild((int)str[i]);
48. }
49. else{
50. show_flag = 1 ;
51. }
52. i++;
53. }
```

```
54. this->printBestmatch(str);
55. }
```

```
56. void Trie::printBestmatch(char* input){        //an to show_flag pou phre dn
    einai 1 tote kalei thn synarthsh printFinal h opoia emfanizei tis epiloges ths
    le3hs
```

```
57. temp = root;
58. if(show_flag == 1)
59. cout << endl <<"  No mathching words!!" << endl;
60. else
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
{
for(i=0;i<(int)strlen(input);i++)            //kanei to temp tn teleutaio komvo tou grammatos tou string wste na kalestei h synarthsh gia apo kei kai pera
    a. temp = temp->goToChild((int)input[i]);
temp->printFinal();
}
}
```

```
char * Trie::chooseWord(int c){    //pairnei enan ari8mo pou 8a einai to poia le3h exei epilegei
if(c > 0 && c <= N)
temp = temp->goToLeaf(c);  //phgainei sto katallhlo fyllo
else
return NULL;
return temp->getWord();     //kai epistrefei th le3h sthn opoia odhgei auto to fyllo
```

```
}
```

```
void Trie::freeTrie(node_ptr n) {            //Anadromika diagrafei olo to dentro
for (int i=0; i<27; i++) {
if (!n->checkChild(i+97)){
    a. n = n->goToChild(i+97);
    b. freeTrie(n);
    c. }
}
delete n;
}
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Κώδικας φύλων δέντρου node

Κώδικας node.h

```
1. #ifndef NODE_H
2. #define NODE_H
3. #include <stdlib.h>
4. #include <string.h>



5. #define N 4


6. typedef class Node * node_ptr;   // Me auto ton tropo otan grafoume node_ptr
   gia dhlwsh mia metavlhths 8a ennooume Node* dhladh deikth se Node


7. class Node {



8. private:
9. char value;       //O xarakthras pou 8a xei mesa o ka8e komvos , to fyllo exei to
   @
10. int app_ratio;    //SYXNOTHTA emfanishs
11. int i;     //voh8aei sta loops
12. int temp_min , k; //voh8aei sthn synarthsh fixleafNodes() gia na vroume to
    fyllo me thn elaxisth syxnothta an prokkeitai na antikatasta8ei
13. node_ptr pateras;   //o ka8e komvos deixnei ston patera tou
14. node_ptr leafPointers[N]; //o ka8e komvos dexnei apeu8eias sta N pio syxna
    fylla
15. node_ptr paidia[27];      //deixnei kai sta paidia tou
16. char str1[20];           // string pou xreiazetai gia na epistrefoume timh ap thn
    getWord


17. public:
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
18. Node(char);                    //Constructor ths Node pairnei xarakthra pou ton
    vazei sto value
19. ~Node();
20. void fixLeafPointers( node_ptr );  //ftiaxnei anadromika tous leafPointers ka8e
    fora pou eisagetai h ananewnetai mia le3h
21. void setLeaf(void);              //kaleitai gia to teleutaio gramma ths le3hs kai
    tou ftiaxnei ena fyllo wste na einai telikh le3h
22. void checkAlphabeticalOrder();
23. node_ptr goToChild( int);         //epistrefei to paidi tou komvou analoga me to
    pio paidi 8eloume
24. bool checkChild( int);            //elegxei to sygkekrimeno paidi kai epistrefei
    analoga
25. void addChild( char);             //dhmiourgei ena paidi
26. void set_parent(node_ptr);        // 8etei ton patera tou komvou pou thn kalei ,
    kaleitai ap to root
27. void increase_app_ratio(void);    //au3anei th syxnothta (app_ratio) kata 1
28. void printFinal(void);            //ektypwnei tis proteinomenes le3eis
    symplhrwshs , tis pairnei ap th getword
29. char* getWord(void);              //3ekinaei apo to fyllo pou deixnei o leafPointer
    kai pairnei ka8e xarakthra mexri to root , to antistrefei kai exoume ena string
    me th le3h thn opoia thn epistrefei
30. node_ptr goToLeaf(int);           // epistrefei ekei opu deixnei o leafPointer tou
    komvou p thn kalei analoga me to ti orisma 8a paei
31. };



32. #endif // NODE_H
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Κώδικας node.cpp

```
1. #include <iostream>
2. #include "Node.h"


3. using namespace std;


4. Node::Node(char val) : value(val) , app_ratio(0) , k(0)   // ta yoploipa den
   xreazontai arxikopoihsh
5. {
6. //  cout << "NODE CREATED WITH VALUE  " << value << endl;


7. for (i=0;i<27;i++)         // to 27 na ginei define
8. paidia[i] = NULL;
9. for (i=0;i<N;i++)
10. leafPointers[i] = NULL;
11. }



12. Node::~Node() {
13. for (i=0;i<27;i++)
14. delete paidia[i];
15. }



16. bool Node::checkChild( int val){       //epistrefei true an den yparxei paidi
    alliws false


17. if(this->paidia[val-97] == NULL)
18. return true;
19. else
20. return false;
```

```
21. }


22. node_ptr Node::goToChild(int val){          //epistrefei to paidi pou zhteitai
23. if(val == '@')
24. return this->paidia[26];
25. else
26. return this->paidia[val-97];




27. }



28. void Node::addChild( char val){             //dhmiourgei paidi
29. int n = (int)val;
30. this->paidia[n-97]=  new Node(val);        //(node_ptr)
    malloc(sizeof(node_ptr));
31. this->paidia[n-97]->pateras = this;


32. }




33. void Node::fixLeafPointers( node_ptr ypopsifioFillo) {     //me anadromh
    ftiaxnei ta fylla


34. temp_min = ypopsifioFillo->app_ratio;                  //8etoume to temp_min me
    isio me to app_ratio tou fyllou pou elegxoume
35. k = -1;



36. for ( i=0;i< N;i++) {                          //An yparxei hdh mesa stous
    leafPointers
37. if (leafPointers[i] == ypopsifioFillo){
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
38. if (pateras != NULL)
39. pateras->fixLeafPointers(ypopsifioFillo);    //kaleitai pali gia ton apo panw
    komvo , an yparxei
40. return;}
41. }
```

```
42. for(i=0 ; i<N; i++){                          //an yparxei xwros mesa to vazei
    me thn mia
43. if(leafPointers[i] == NULL){
44. leafPointers[i] = ypopsifioFillo ;
45. if (pateras != NULL)
46. pateras->fixLeafPointers(ypopsifioFillo);
47. return;
48. }   }
```

```
49. //  temp_min = leafPointers[0]->app_ratio;
50. for(i = 0;i < N; i++){                         //An den isxyei kanena ap ta
    prohgoumena tote vriskei to mikrotero kai to ektopizei
51. if(leafPointers[i]->app_ratio < temp_min){
        a. temp_min = leafPointers[i]->app_ratio;
        b. k = i;                                  }
52. }
53. if(k >=0)
54. leafPointers[k] = ypopsifioFillo;
```

```
55. if (pateras != NULL)
56. pateras->fixLeafPointers(ypopsifioFillo);
```

```
57. }
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
58. void  Node::setLeaf() {              //gia ton komvo tou teleutaiou gammatos ths
    le3hs dhmiourgeitai paidi me @


59. this->paidia[26] = new Node('@');
60. this->paidia[26]->pateras = this;


61. }


62. void Node::checkAlphabeticalOrder(){









63. }


64. void Node::increase_app_ratio(){      //thn kalei o komvos prin to fyllo
    synepws au3anoume ato app_ratio tou paidiou tou pou einai fyllo
65. this->paidia[26]->app_ratio++;
66. }



67. void Node::set_parent(node_ptr n){   //dinei timh ston patera tou , kaleitai ap
    th root


68. pateras = n;
69. }



70. char * Node::getWord(){          //3ekinaei ap to teleutaio komvo ths le3hs kai
    phgainei pros ta panw
71. char str[20];
72. strcpy(str1,"");
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
73. int j =0;
74. i = 0;
75. node_ptr temp = this->pateras;
76. while(temp != NULL){
77. str[i++] = temp->value;
78. temp = temp->pateras; }
79. str[i] = '\0';


80. for(i=strlen(str)-2 ; i >= 0 ; i--)   //to str exei mesa to string ths le3hs alla
    anapoda ara to antistrefoume kai to peistrefei
81. str1[j++] = str[i];
82. str1[j] = '\0';


83. return str1;
84. }


85. void Node::printFinal(){          //emfanizei gia ka8e leaf pointer thn le3h ,
    dhladh tis pio syxnes le3eis pou proteinontai gia th symplhrwsh
86. cout << endl << "  Word Matches : " ;
87. for(i=0; i < N; i++){
88. if(this->leafPointers[i]!=NULL)
89. cerr << i+1 <<". " <<  this->leafPointers[i]->getWord() << "  ";  //standard
    error
90. }


91. if(this->leafPointers[0] == NULL)
92. cout <<endl<< "No Match sorry :(" << endl; ;


93. cout << endl;


94. }


95. node_ptr Node::goToLeaf(int x){

96.
```

```
97. (leafPointers[x-1]->app_ratio)++; //au3anei kai to app_ratio
98. return leafPointers[x-1];


99. }
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Κώδικας main του προγρμματος

```
1. #include <iostream>
2. #include "Trie.h"
3. #include <string.h>
4. #include <stdio.h>
5. #include <sys/ioctl.h>
6. #include <ctype.h>
7. #include <unistd.h>
8. #include <termios.h>


9. using namespace std;


10. #define WORD_LENGTH 30        //megisto megethos tou buffer ths le3hs
11. #define BACKSPACE 0x7f
12. #define TAB '\t'
13. #define WHITESPACE ' '
14. #define END_LINE '\n'
15. #define LINESIZE 128
16. #define PATH_LENGTH 40


17. //Nikolaos Maroulis
18. //AM : 1115201000212


19. int getchar_silent()
20. {
21. int ch;
22. struct termios oldt, newt;
23. /* Retrieve old terminal settings */
24. tcgetattr(STDIN_FILENO, &oldt);
25. /* Disable canonical input mode, and input character echoing. */
26. newt = oldt;
27. newt.c_lflag &= ~( ICANON | ECHO );
28. /* Set new terminal settings */
29. tcsetattr(STDIN_FILENO, TCSANOW, &newt);
30. /* Read next character, and then switch to old terminal settings. */
31. ch = getchar();
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
32. tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
33. return ch;
34. }


35. int main(int argc , char * argv[])
36. {
37. char PATH[PATH_LENGTH];
38. FILE * dict;
                                    a.  //DHLWSH PATH TOU DICTIONARY
39. if (argc != 1 && argc != 3){                    // an dw8oun la8os orismata , dekta
    ginontai ka8olou orismata kai -d PATH
40. perror( " Wrong number of arguments! ");
41. return -1;
42. }
43. if (argc == 1)                                  //an den exei dw8ei orisma tote dinetai to
    default orisma sto PATH
44. strcpy(PATH,"$HOME/.dict");
    ///ALLAGH//////////////////////////////"$HOME/.dict"
45. else{                                          //an exei de8ei kapoio orisma
46. if (strcmp(argv[1],"-d") == 0)                 //an to prwto orisma einai to -d tote
    proxwrame swsta sto path
47. strcpy(PATH,argv[2]);
48. else{                                          //alliws emfanizei la8os
49. cout << "Wrong arguments given" << endl;
50. return -1;
51. }
52. }
53. if((dict= fopen(PATH,"r+")) == NULL){           //Anoigei to arxeio kai
    elegxei an anoi3e swsta alliws emfanizei mhnyma la8ous
54. perror("Error opening file");
55. return -1;   }
56. else
57. cout << "Success : File loaded to Trie!!" << endl;


58. char ch;              // einai o xarakthras pou xrhsimopoieitai gia na pairnei to ti
    plhktrologoume
59. int i=0 , j;          //voh8itika gia loops kai gia na kinoumai stous xarakthres
    twn buffers
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
60. char line_text[LINESIZE];  //pairnei olh th grammh pou plhkrologeitai
61. char  input_buffer[WORD_LENGTH];      //8a parei mesa to string ths le3hs
    ap thn fgets


62. Trie tr;                        //Dhmiourgia dentrou


63. while( fgets (input_buffer , WORD_LENGTH ,dict) != NULL ){   //Diavazei
    olo to dictionary grammh grammh
64. if(input_buffer[strlen(input_buffer) - 1] == '\n')        //h fgets krataei sthn
    teleutai 8esh th grammh ara thn svhnoume
65. input_buffer[strlen(input_buffer) -1] = '\0' ;
66. tr.add_node(input_buffer,0);                              // eisagwgh sto dentro mesw
    ths add_node , pairnei 2o orisma to 0 dhladh kanei to READ_FROM 0 pou
    shmainei oti phre to string apo arxeio
67. }


68. strcpy(line_text,"");             //arxikopoihsh twn string me to keno
69. strcpy(input_buffer,"");
70. i=0;




71. do{      //Do ths VEOF
72. cout << "> " ;
73. line_text[0] = '>';            //Autoi 8a nai oi prwtoi xarakthres tou string ths
    grammhs
74. line_text[1] = ' ';
75. j = 2;


76. ch = getchar_silent();          //diavazoume xarakthra xarakthra mesw ths
    getchat_silent()
77. putchar(ch);                    //emfanizei ton xarakthra gia na 3eroume ti grafoume


78. do{   //Do ths allaghs grammhs
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
79. do{ //Do ths alaghs grammhs, kenou h TAB


80. while(isalpha(ch)){              //An einai gramma ths alfavhtou
81. if(ch>='A' && ch <= 'Z')      //An einai kefalaio to kanei mikro
        a. ch = ch + 32;
        b. input_buffer[i++] = ch;     //to vazei sto buffer ths le3hs
        c. line_text[j++] = ch;        //kai ths grammhs
        d. ch = getchar_silent() ;
        e. putchar(ch);
82. }
83. while(ch == BACKSPACE ){         //An einai backspace
84. if(j > 2){                  //
        a. printf("\b \b");          //me to \b phgainei pisw kai fainetai san na
           svhnetai o xarakthras
        b. j--;}                    //afaireitai ena ap ton mertrhth ths grammhs
85. if(i > 0)
        a. input_buffer[i--] = '\0'; //kanoume to gramma ths le3hs keno kai
           afairoume ena ap ton metrhth tou buffer ths le3hs
        b. ch = getchar_silent() ;
        c. putchar(ch);    //
86. }


87. if(ch == TAB){                  //  PERIPTWSH TAB
88. input_buffer[i] = '\0' ;     //kleinei th le3h wst na mporei na th dwsei
89. tr.checkTAB(input_buffer);   // kaleitai h synarthsh checkTAB ths klashs Trie
    pou elegxei kai emfanizei ta kalytera apotelesmata gia th symplhrwsh ths le3hs
    pou path8hke TAB
90. line_text[j] = '\0';
91. cout << line_text;          //emfanizei thn grammh apo katw wste na
    synexizoume na grafoue apo kei pou hmastan
92. ch = getchar_silent();       //diavazei allo xarakthra
93. putchar(ch);
94. if(ch > '0' && ch <= '9'){         //An diale3ei kapoia ap tis epiloges tote kaleitai
    h chooseWord pou oloklhrwnei th le3h
        a. if(j>2){
        b. for(int k=0;k<=(int)strlen(input_buffer);k++)  //svhnei auto p xame
           grapsei mexri tote wste na vgalei oloklhrh th kainouria le3h dinonta
           mas thn entypwsh oti th symplhrwse
```

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

c. printf("\b \b");
1. }
d. j= j - strlen(input_buffer); //afairoume to palio string ap to bufferths grammhs

e. strcpy(input_buffer,tr.chooseWord(ch-48)); //vazoume sto input_buffer thn le3h pou epile3ame na symplhrwsoume
f. cout << input_buffer ;//<< flush; //thn ektypwnoume
g. tr.add_node(input_buffer,1); // kai thn pros8etoume h au3anoume to app_ratio(suxnothta) ths an yparxei hdh sto dentro , 2o orisma(READ_FROM) einai 1 afou phre to string ap ton xrhsth

h. for(i=0 ; i < (int)strlen(input_buffer) ; i++) //prosthetoume to string pou epilex8hke sthn grammh
i. line_text[j++] = input_buffer[i];
j. line_text[j]= '\0';
k. i=0; //mhdenismos tou i gia na arxisei ap thn arxh h leksi
l. strcpy(input_buffer,""); //adeiazoume to input_buffer gia na mporoume na to xrhsimopoihsoume se allh le3h pio meta
m. ch = getchar_silent();
n. putchar(ch);
95. }
96. else if (isalpha(ch)){ //an meta to TAB einai gramma ananewnei ta strings
a. input_buffer[i++] = ch;
b. line_text[j++] = ch;
c. ch=getchar_silent();
d. putchar(ch);
97. }
a.

98. }

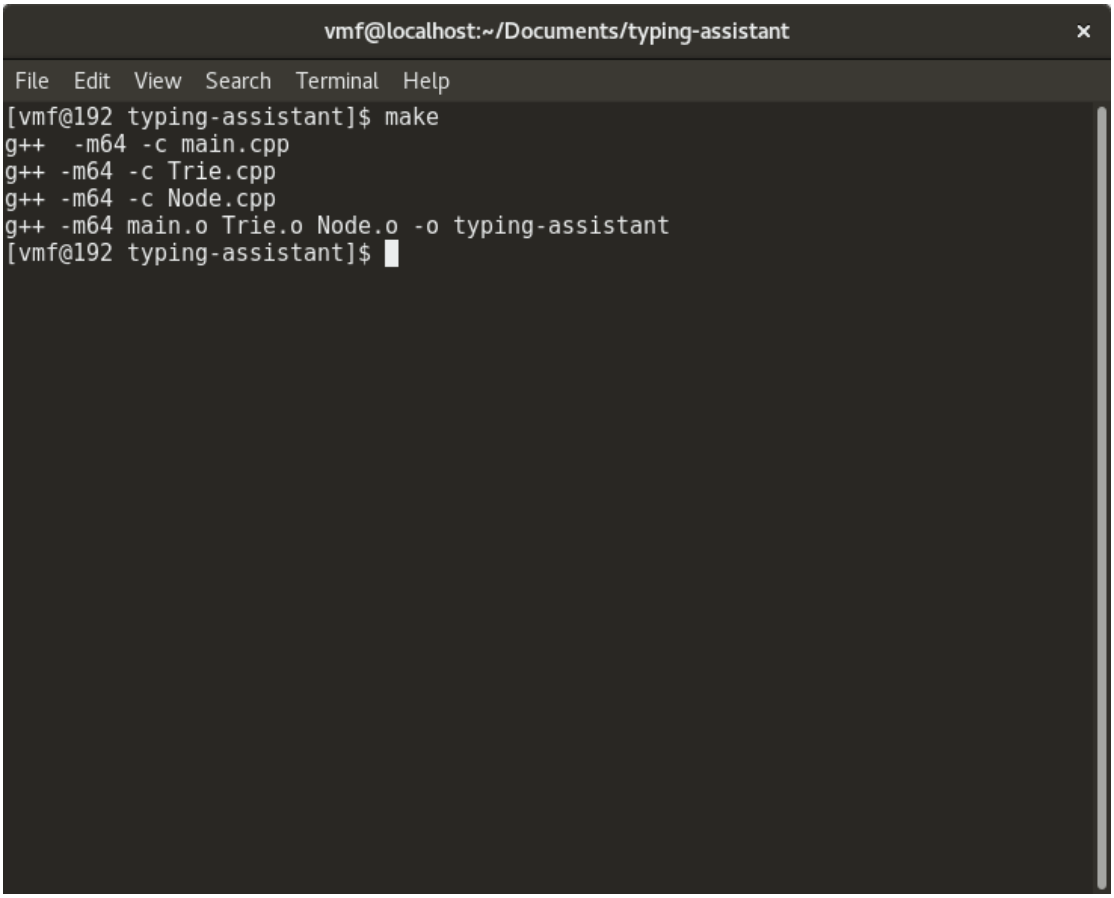
99. if (ch == WHITESPACE) { //an einai keno vazei keno stro string ths grammhs
100. line_text[j++] = ' ';
i. }

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
101.        }while(ch != END_LINE && ch != WHITESPACE && ch != TAB
    && (ch < '0' || ch > '9'));


102.        if((ch == WHITESPACE || ch == END_LINE) && i > 0) {
    //Dhmiourgia/elegxos le3hs se periptwsh pou eisax8ei kai path8ei keno h
    allagh grammhs
103.        input_buffer[i] = '\0' ;
104.        tr.add_node(input_buffer,1);   //pairnei 2o orisma 1 afou exei diavasei
    apo xrhsth
105.        i=0;                    //mhdenismos tou i gia na arxisei ap thn arxh h
    leksi
106.        strcpy(input_buffer,"");       //adeiasma tou buffer
107.        }
108.        if(ch == WHITESPACE)
109.        { ch = getchar_silent();
110.        putchar(ch);
111.        }


112.        }while(ch != END_LINE);


113.        }while(ch != VEOF);


114.        fclose(dict);   //kleinoume to arxeio p anoi3ame
115.        return 0;


116.        }
```

ΣΤΙΓΜΙΟΤΥΠΑ ΛΕΙΤΟΥΡΓΙΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

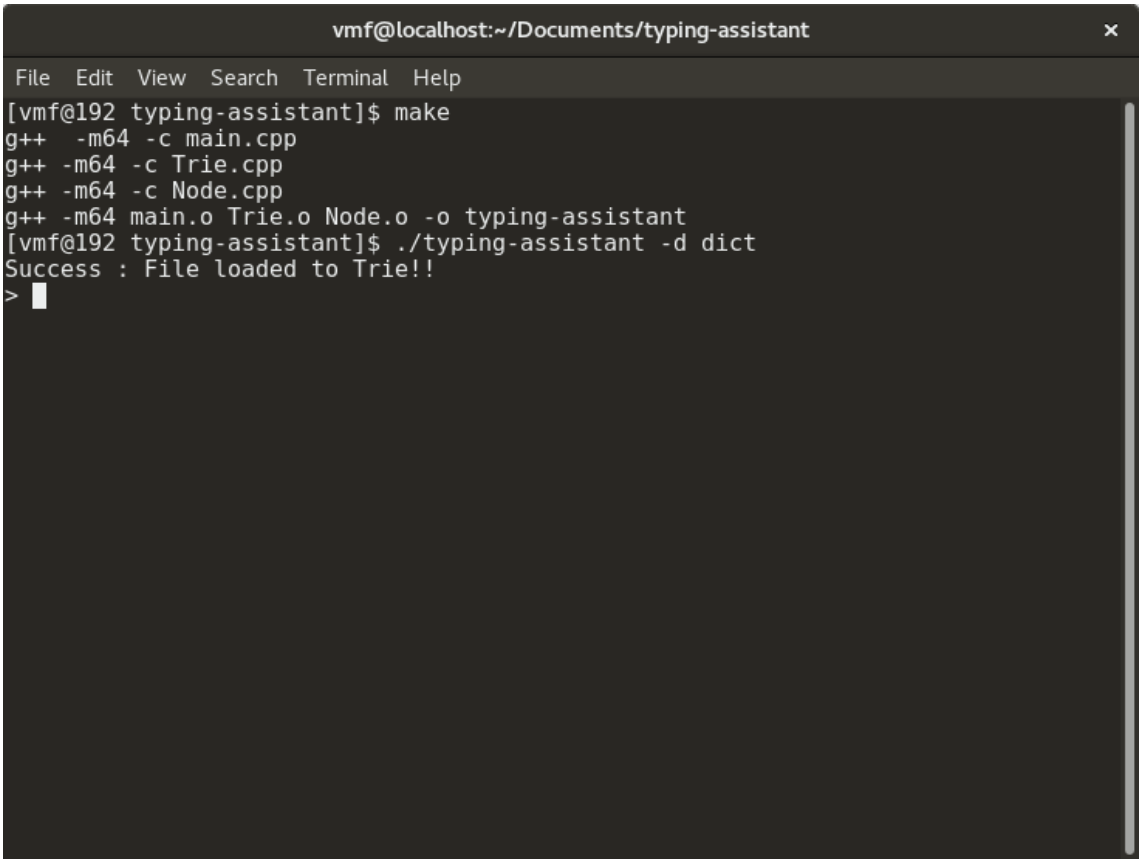
Τρέχουμε την εντολή make από το τερματικό ενώ βρισκόμαστε στον φάκελο που βρίσκονται τα αρχεία μας



Η εντολή make δημιούργησε τα αρχεία main.o, Trie.o, Node.o και το εκτελέσιμο typing-assistant.

ΣΤΙΓΜΙΟΤΥΠΑ ΛΕΙΤΟΥΡΓΙΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

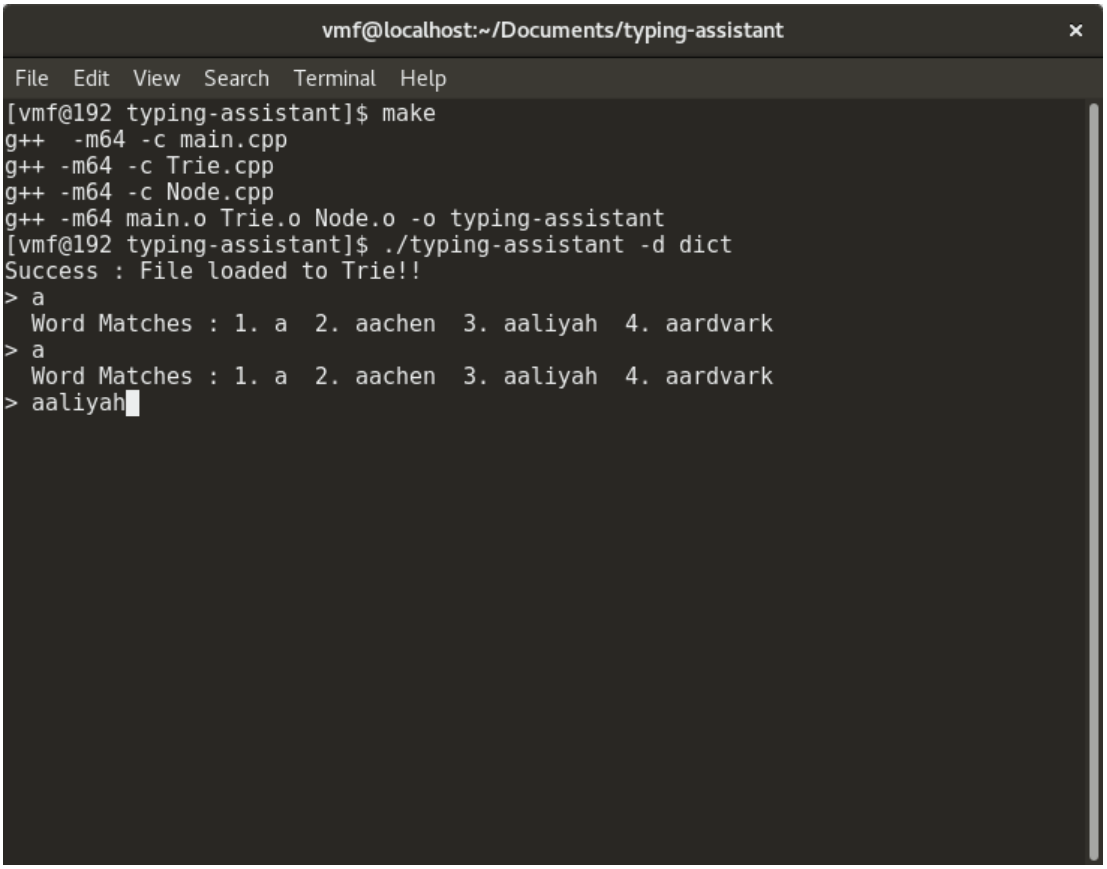
Τρέχουμε το εκτελέσιμο αρχείο typing-assistant πληκτρολογώντας ./typing-assistant –d dict, όπως φαίνεται παρακάτω και τότε ξεκινάει να τρέχει το πρόγραμμα.



ΣΤΙΓΜΙΟΤΥΠΑ ΛΕΙΤΟΥΡΓΙΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Πληκτρολογούμε κάποιο γράμμα ή την αρχή κάποιας λέξης και στη συνέχεια πιέζοντας TAB, μας δίνονται κάποιες επιλογές για την συμπλήρωση της λέξης.

Επιλέγουμε λέξη πιέζοντας το αντίστοιχο νούμερο και στη συνέχεια ολοκληρώνουμε με ENTER.



Σε περίπτωση που πληκτρολογήσουμε κάποια λέξη η οποία δεν είναι καταχωρημένη στη βάση δεδομένων μας και πατήσουμε ENTER τότε γίνετε αποθήκευση.