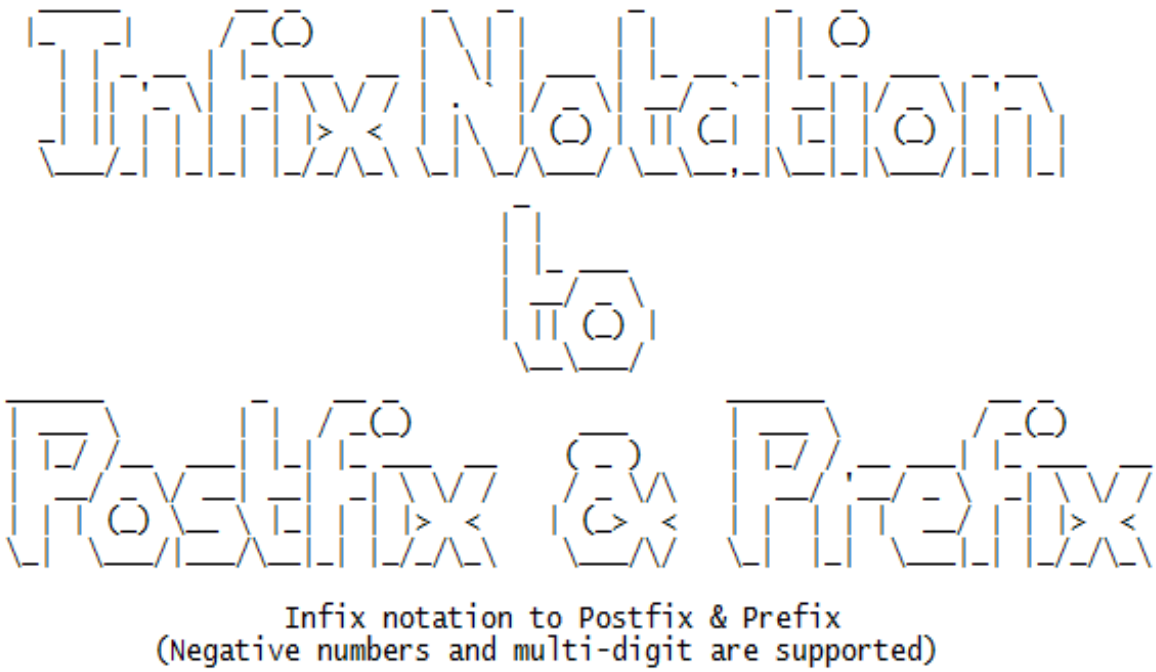
**ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ**
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ**

# ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

## 2η Εργασία – Δυαδικά Δένδρα




**ΑΛΈΞΑΝΔΡΟΣ ΠΛΈΣΣΙΑΣ**
*A.M.: 2025201100068*
*cst11068@uop.gr*

Εαρινό εξάμηνο 2013-2014

**Περιεχόμενα**

# Συναρτήσεις που χρησιμοποιήθηκαν

- **BasicFunctions.c**

void TextArt(): Εμφάνιση μηνύματος καλωσορίσματος, Θέλετε να συνεχίσετε & Κανόνες.

int Menu(): Εμφάνιση μενού επιλογών και επιστροφή έγκυρης επιλογής.
void infixToPostAndPre(): Επεξεργάζεται την επιλογή του χρήστη και κάνει την κατάλληλη ενεργεία σε περίπτωση πατήματος πρώτα τις επιλογής 2 ή 3 εμφανίζεται κατάλληλο μήνυμα.

void insertInfix():Καλώ την συνάρτηση readInfix και μου επιστρέφει το input του χρήστη, έπειτα αποθηκεύω σε έναν πινάκα τις θέσεις των operator / παρενθέσεων και καλώ την tokenazationInput.

void createPostFixList(): Δημιουργία postfix λίστας με χρήση στοίβας operator (χρήση addOperatorStack). Όταν διαβάζει αριθμό τον εισάγει στην λίστα όταν βλέπει operator τον προσθέτει στην στοίβα αφού ποιο πριν κοιτάξει την κορυφή της και ελέγξει τις προτεραιότητες και κάνει push και pop (χρήση popOperatorStack) , όταν δει κλείσιμο παρένθεσης αδειάζει την στοίβα μέχρι να βρει το άνοιγμα της και τέλος αδειάζει την στοίβα.

int precedenceOfOperator(char* oper): Επιστρέφη έναν ακέραιο που αντιστοιχή στην προτεραιότητα του operator που δίνεται σαν όρισμα.

- **InputFuntions.c**

readInfix(): Δυναμικό διάβασμα εισόδου από τον χρήστη & έλεγχος για το εάν είναι έγκυρο. Γίνεται έλεγχος για τις παρενθέσεις, εάν ξεκινάει ή τελειώνει με operator, για το εαν χρησιμοποιούνται μη έγκυρη χαρακτήρες, εαν δοθεί μόνο ένας αριθμός μονός του, εάν έχουμε χρήση operator μετά από operator (δεν αναφέρομαι στις παρενθέσεις) , εάν πατηθεί μόνο το πλήκτρο Enter και εάν ανοίγουμε και κλείνουμε παρένθεση χωρίς περιεχόμενο όλα τα παραπάνω ελέγχονται από ειδικές συναρτήσεις. Τέλος, επιστρέφει δείκτη με το περιεχόμενο της εισόδου του χρήστη.

int checkParenthesis(char* input): Γίνεται έλεγχος για τις παρενθέσεις με την χρήση του λήμματος των παρενθέσεων παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0 για μη-έγκυρη .

int checkStartAndEnd(char* input): Γίνεται έλεγχος για το εάν ξεκινάει ή τελειώνει με operator παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0για μη-έγκυρη .

int checkValidity(char* input): Γίνεται έλεγχος για το εαν χρησιμοποιούνται μη έγκυρη χαρακτήρες παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0 για μη-έγκυρη .

int checkOnlyNumber(char* input): Γίνεται έλεγχος για το εαν δοθεί μόνο ένας αριθμός μονός του παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0 για μη-έγκυρη .

int checkOpAfterOp (char* input): Γίνεται έλεγχος για το εάν έχουμε χρήση operator μετά από operator (δεν αναφέρομαι στις παρενθέσεις) παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0 για μη-έγκυρη .

int checkParenthesisAfterParenthesis (char* input): Γίνεται έλεγχος για το εάν ανοίγουμε και κλείνουμε παρένθεση χωρίς περιεχόμενο παίρνει σαν παράμετρο το input του χρήστη και επιστέφει 1 για έγκυρη μορφή και 0 για μη-έγκυρη .

int numberOfDigits(int numberCheck): Επιστρέφει τον αριθμό των ψηφίων του αριθμού που δίνεται σαν όρισμα μέσα από μια μαθηματική φόρμουλα (floor(log10(abs(numberCheck))) + 1).

void tokenazationInput(char* input,int* operatorsPositions, int size): Χωρίζει την είσοδο του χρήστη σε αριθμούς (θετικούς και αρνητικούς) & operator's και τούς αποθηκεύει στο infix list. Δέχεται σαν παραμέτρους την είσοδο του χρήστη, τον πινάκα με της θέσεις των operators και το μέγεθος του πίνακα των operators.

- **listFuntions.c**

void addList (int expesionCode, char* termIn): Προσθέτη το στοιχείο της έκφρασης στη λίστα (άπλα συνδεδεμένη) με τον αντίστοιχο κωδικό (1 για infix & 2 για postfix).

void freeList(int expesionCode): Άδειασμα της λίστας (άπλα συνδεδεμένη) με τον αντίστοιχο κωδικό (1 για infix & 2 για postfix).

void setExpessionElement (int expesionCode ,char type, char signOrOperator, int number): Μετατρέπει τους operators σε char* και μετατρέπει τους αριθμοί πάλι σε char* αλλά προσημασμένους.

- **stackFunctions.c**

void addOperatorStack(char* oper): Προσθέτη στην στοίβα τον operator που δίνεται σαν όρισμα.

char* popOperatorStack(): Επιστρέφη την τιμή (operator) που είναι στην κεφαλή της στοίβας και την απελευθερώνει.

void freeOperatorStack(): Άδειασμα της στοίβας.

- **treeFunctions.c**

treeNode* createBinaryTree(): Κάθε στοιχείο της postfix list ελέγχεται για το εαν είναι αριθμός ή operator, εαν είναι αριθμός δημιουργείτε ένα δέντρο με τον αριθμό σαν τιμή και τα παιδιά του NULL εαν είναι operator δημιουργείτε πάλι ένα δέντρο με τιμή τον operator και παιδιά τις 2 πρώτες τιμές τις στοίβας των δέντρων. Τέλος επιστρέφει την κεφαλή της τις στοίβας δέντρων που είναι και η ρίζα του τελικού δέντρου.

void addTreeNodeToStack(treeNode* tree): Προσθέτη στην στοίβα το δέντρο που δίνεται σαν όρισμα.

treeNode* popTreeNodeStack(): Επιστρέφη την τιμή (tree) που είναι στην κεφαλή της στοίβας και την απελευθερώνει.

void freeTreeNodeStack(): Άδειασμα της στοίβας

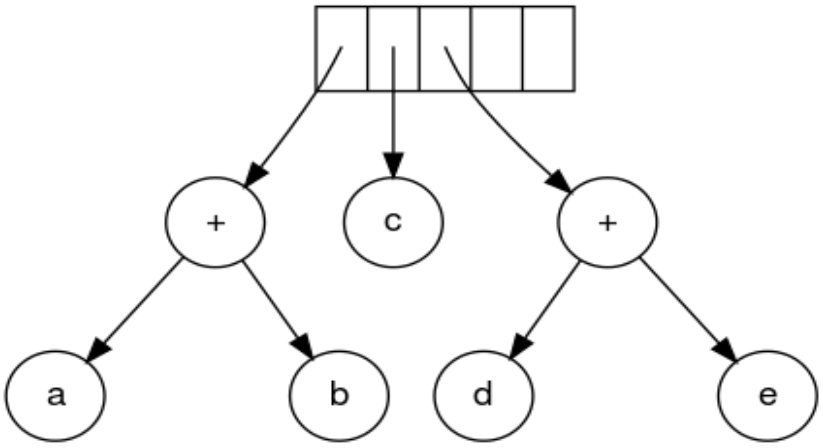
- **displayFuntions.c**

void preFix(treeNode* root): Εμφανίζει αναδρομικά καλώντας τον ίδιο τον εαυτό της και επισκέπτεται το δέντρο με την μέθοδο Value Left Right.

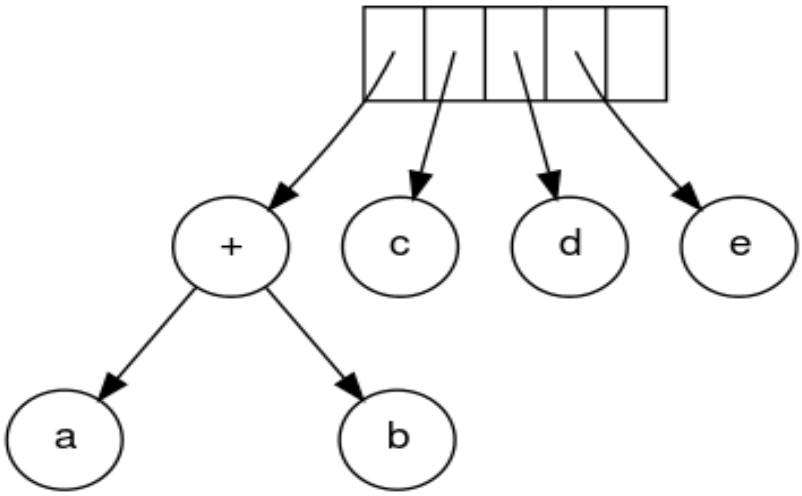
void postFix(treeNode* root): Εμφανίζει αναδρομικά καλώντας τον ίδιο τον εαυτό της και επισκέπτεται το δέντρο με την μέθοδο Left Right Value.

# Γενική περιγραφή

Αρχικά ζητάμε από τον χρήστη να μας δώσει μια αριθμητική παράσταση την οποία και αποθηκεύουμε δυναμικά και ελέγχουμε την εγκυρότητα της και κρατάμε σε έναν πινάκα τις θέσεις των operators και των παρενθέσεων για να κάνουμε το tokenazationInput δηλαδή να χωρίσουμε το input string σε char* αριθμούς (θετικούς και αρνητικούς) και σε operators με σκοπό την αποθήκευση τους στην infix λίστα. Έπειτα μετατρέπω την infix μορφή σε postfix με την βοήθεια της συνάρτησης createPostFixList (ανήκει BasicFunctions.c) και δημιουργώ το δυαδικό δέντρο μου από αυτή (την postfix list) δηλαδή κάθε στοιχείο της postfix list ελέγχεται για το εαν είναι αριθμός ή operator, εαν είναι αριθμός δημιουργείτε ένα δέντρο με τον αριθμό σαν τιμή και τα παιδιά του NULL εαν είναι operator δημιουργείτε πάλι ένα δέντρο με τιμή τον operator και παιδιά τις 2 πρώτες τιμές τις στοίβας των δέντρων (Βλέπε εικονα1 & Εικόνα 2) και επιστρέφει την κεφαλή της τις στοίβας δέντρων που είναι και η ρίζα του τελικού δέντρου .




*Εικόνα 1: Στοίβα δεικτών τύπου δένδρου*




*Εικόνα 2: Στοίβα δεικτών τύπου δέντρου μετά την είσοδο operator*

Τέλος, περνούμε την ρίζα του τελικού δέντρου και με χρήση των συναρτήσεων postFix και preFix εμφανίζουμε αναδρομικά επισκέπτεται το δέντρο με την μέθοδο Left Right Value για την postfix & Value Left Right για την prefix.

## Δομές δεδομένων

```
typedef struct expesionList
{
    char* term ;
    struct expesionList * next;
} expesionList;

typedef struct operatorStack
{
    char* op ;
    struct operatorStack * next;
} operatorStack;

typedef struct treeNode
{
    char* data;
    struct treeNode * firstChildren;
    struct treeNode * secondChildren;
} treeNode;

typedef struct treeNodeStack
{
    treeNode* roots;
    struct treeNodeStack * next;
} treeNodeStack;
```

*Εικόνα 3: Τα Structs που χρησιμοποιήθηκαν*

Χρησιμοποίησα σε όλα τα Structs δυναμική διαχείριση και όχι στατική δηλαδή όχι έναν πίνακα κάποιων θέσεων που το ποιο πιθανό θα ήταν να μην χρησιμοποιούνταν όλες. Το Struct **expresionList** χρησιμοποιήθηκε σαν μια απλά συνδεδεμένη λίστα για την αποθήκευση των infix και postfix. Το Struct **operatorStack** χρησιμοποιήθηκε σαν στοίβα operators και μας βοήθησε στην μετατροπή του infix σε postfix. Το Struct **treeNode** χρησιμοποιήθηκε για την δημιουργία αρκετών sub-trees τα οποία με την βοήθεια του Struct **trreeNodeStack** ώστε να δημιουργηθεί το τελικό δένδρου.

Για τούς κόμβους του δέντρου χρησιμοποίησα διπλά συνδεόμενη λίστα και όχι άλλη δομή όπως αποθήκευση δέντρου σε πινάκα για να εκμεταλλευτώ τους αναδρομικούς τύπους τυπώματος LRV & VLR.

## Παράδειγμα λειτουργίας του προγράμματος

```
Please enter a Numeric expression: 5*(((9+(-8))*((-4)*6))+7)

Postfix: (+5) (+9) (-8) (+) (-4) (+6) (*) (*) (+7) (+) (*)

Prefix: (*) (+5) (+) (*) (+) (+9) (-8) (*) (-4) (+6) (+7)
```