Κωνσταντίνος Σκορδούλης
ΑΜ: 1115 2016 00155

ADT Binary Tree

Το συγκεκριμένο ερώτημα απαιτεί τον ορισμό συναρτήσεων **Create,IsEmpty,MakeTree,Delete,PreOrder,InOrder , PostOrder , LevelOrder,Height,Size and Print.**

Χρησιμοποιήθηκαν και κάποιες βοηθητικές συναρτήσεις, όπως η **PrintLevel.**

Link Create(Link N)

Παίρνει σαν όρισμα ένα δείκτη που δείχνει σε ένα κόμβο, όπου εδώ θα είναι ο δείκτης σε **root**.Σκοπός της συνάρτησης είναι η δημιουργία ενός **empty binary tree**.

1) Δεσμεύουμε την απαραίτητη μνήμη με **malloc**
2) Αναθέτουμε στους δείκτες **(left,right)** την τιμή **NULL** (έτσι ώστε τα παιδιά να είναι κενά , **NULL**).

Επιστρέφει το **root**.

Int IsEmpty(Link N)

Παίρνει σαν όρισμα ένα δείκτη που δείχνει σε ένα κόμβο, όπου εδώ θα είναι ο δείκτης σε **root.**Σκοπός της συνάρτησης είναι να ελέγξει αν το **binary tree** είναι **empty.**Οπότε ελέγχει αν:**’**

1)**N->Left=NULL**

2)**N->Right=NULL**

Επιστρέφει:

- **True(1)** αν ισχύουν οι συνθήκες και άρα είναι **empty.**

- **False(0)** αν δεν ισχύουν οι συνθήκες και άρα **non empty**.

Link MakeTree(Root,Left,Right)

Παίρνει σαν όρισμα 3 δείκτες που δείχνουν σε κόμβους **root**. Σκοπός της συνάρτησης είναι η να δημιουργήσει ένα καινούργιο δέντρο , όπου **root** είναι το **Root** με δεξί παιδί το **Right subtree** και αριστερό παιδί το **Left subtree.** Αναλυτικότερα:

1) **Root->Left=Left**

2) **Root->Right=Right**

Επιστρέφει το **root.**

Void Delete(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί ορισμένα βήματα :

1) Ελέγχει η ρίζα είναι **NULL ,** δηλαδή αν το δέντρο είναι κενό, οπότε δεν έχει νόημα να εκτελεστεί.
2) Διαγράφει **αναδρομικά** το δεξί και αριστερό **subtree**, τυπώνοντας κάθε φορά το **tree node** που διαγράφει.
3) Τέλος σβήνει το **root.**

Int Height(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί τα εξή βήματα :

1) **Ελέγχει** αν η ρίζα είναι **NULL**, ηλαδή αν το δέντρο είναι κενό, οπότε δεν έχει νόημα να προχωρήσει.
2) **Υπολογίζει αναδρομικά** το δεξί και αριστερό **subtree.**
3) **Συγκρίνει** αυτά τα δύο, και **επιστρέφει** το μεγαλύτερο **+1** (**+1** για να συμπεριλάβουμε και τη **ρίζα**).

Void LevelOrder(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί ορισμένα βήματα :

1) **Ελέγχει** αν η ρίζα είναι **NULL,** αλλιώς προχωράει.
2) **Υπολογίζει** το **height** του δέντρου και το αποθηκεύει σε μια μεταβλητή **h**.
3) Για **i=1 μέχρι και i=height(i++)** καλεί την συνάρτηση **PrintLevel(N,i);**

Void PrintLevel(Link N, int level)

Δέχεται σαν **όρισμα δείκτη σε ρίζα** και **ένα integer** που δηλώνει το **current level**:

1) **Ελέγχει** αν η ρίζα είναι **NULL,** αλλιώς συνεχίζει.
2) **Αν** το **level=1** δηλαδή έχουμε φτάσει στο **level** που επιθυμούσαμε και εκτυπώνουμε το χαρακτήρα-**string**. Δηλαδή για **i=1** εκτυπώνει τη ρίζα, για **i=2…height** εκτυπώνει τα αντίστοιχα επίπεδα με τον παρακάτω τρόπο.
3) **Αν** το **level>1**, καλεί τον εαυτό της, για το δεξί και αριστερό **subtree,** έτσι ώστε να μεταβεί σε κάθε επίπεδο και να εκτυπώνει τις τιμές των **tree nodes** του επιπέδου.

Void PreOrder(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί ορισμένα βήματα :

1) **Ελέγχει** αν ο δείκτης είναι **NULL**, αλλιώς συνεχίζει.
2) **Εκτυπώνει** τη τιμή του **root**.

3) **Εκτυπώνει αναδρομικά** τις τιμές του **left subtree.**
4) **Εκτυπώνει αναδρομικά** τις τιμές του **right subtree.**

Void InOrder(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί ορισμένα βήματα :

1) **Ελέγχει** αν ο δείκτης είναι **NULL**, αλλιώς συνεχίζει.
2) **Εκτυπώνει αναδρομικά** τις τιμές του **left subtree.**
3) **Εκτυπώνει** τη τιμή του **root.**
4) **Εκτυπώνει αναδρομικά** τις τιμές του **right subtree.**

Void PostOrder(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και εκτελεί ορισμένα βήματα :

1) **Ελέγχει** αν ο δείκτης είναι **NULL**, αλλιώς συνεχίζει.
2) **Εκτυπώνει αναδρομικά τις τιμές** του **left subtree.**
3) **Εκτυπώνει αναδρομικά τις τιμές** του **right subtree**
4) **Εκτυπώνει** τη τιμή **root.**

Void Print(Link N, int k)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου, ένα **int k** που χρησιμεύει στον ορισμό των κενών (για να φαίνεται ωραία το δέντρο).Η συνάρτηση **εκτυπώνει** το δοσμένο δέντρο **οριζόντια** και λειτουργεί ως εξής :

1) **Ελέγχει** αν ο δείκτης είναι **NULL**, αλλιώς συνεχίζει.
2) **Εκτυπώνει αναδρομικά** το **right subtree** με **space k+7** κάθε φορά(**+7** μπορεί να τροποποιηθεί)**.**
3) **Εκτυπώνει** το **root.**
4) **Εκτυπώνει αναδρομικά** το **left subtree** με **space k+7** κάθε φορά(**+7** μπορεί να τροποποιηθεί).

Int Size(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου. Η συνάρτηση **υπολογίζει** το σύνολο των **nodes** του δοσμένου δέντρου και λειτουργεί ως εξής :

1) **Ελέγχει** αν ο δείκτης είναι **NULL**, αλλιώς συνεχίζει
2) **Υπολογίζει αναδρομικά** τα **nodes** του **left subtree** και του **right subtree.**
3) **Επιστρέφει** τα **nodes** των δύο **subtrees +1** (**+1** υπολογίζοντας και τη **ρίζα**).

Int PathLength(Link N, int k)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου. Η συνάρτηση **υπολογίζει** το **μήκος μονοπατιού** του δοσμένου δέντρου και λειτουργεί ως εξής :

1) **Ελέγχει** (**κάθε φορά)** αν ο δείκτης είναι **NULL (leaf node) επιστρέφοντας 0**, αλλιώς συνεχίζει.
2) **Υπολογίζει αναδρομικά** το **PathLength(N->Left,k+1)** , δηλαδή το μήκος μονοπατιού όλων των **internal nodes** από το **root** ,του **left subtree,** και αντίστοιχα το **PathLength(N->Right,k+1),** δηλαδή το μήκος μονοπατιού όλων των **internal nodes** από το **root,** του **right subtree**.
3) **Επιστρέφει** σαν αποτέλεσμα το **άθροισμα όλων** των παραπάνω, που είναι ουσιαστικά το άθροισμα όλων των επιπέδων των **internal node.**

Int InternalPathLength(Link N, int k)

Επειδή ασχολούμαστε με **extended binary trees** ισχύει ότι **InternalPathLength = PathLength,** οπότε πρακτικά η συνάρτηση είναι **ίδια.**

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου. Η συνάρτηση **υπολογίζει** το **μήκος μονοπατιού** του δοσμένου δέντρου και λειτουργεί ως εξής :

1) **Ελέγχει** (**κάθε φορά)** αν ο δείκτης είναι **NULL (leaf node) επιστρέφοντας 0**, αλλιώς συνεχίζει.
2) **Υπολογίζει αναδρομικά** το **PathLength(N->Left,k+1)** , δηλαδή το μήκος μονοπατιού όλων των **internal nodes** από το **root** ,του **left subtree,** και αντίστοιχα το **PathLength(N->Right,k+1),** δηλαδή το μήκος μονοπατιού όλων των **internal nodes** από το **root,** του **right subtree**.
3) **Επιστρέφει** σαν αποτέλεσμα το **άθροισμα όλων** των παραπάνω, που είναι ουσιαστικά το άθροισμα όλων των επιπέδων των **internal node.**

Int ExternalPathLength(Link N)

Σε όλα τα **binary trees ,** ισχύει ότι **ExternalPathLength = InternalPathLength + (2*n),** όπου **n** ο αριθμός των **internal nodes.**

Ουσιαστικά, υλοποιούμε τον τύπο αυτό μέσα στη συνάρτηση και επειδή το δοσμένο δέντρο είναι **extended binary tree** ισχύει ότι **n=Size(N) ,** δηλαδή **ίσο** με το **πλήθος** των **nodes** του δέντρου.

Επιστρέφει το άθροισμα**: InternalPathLength(N,0)+(2*Size(N))**

End of : ADT Binary Tree

Postfix Problem

Για την επίλυση αυτού του προβλήματος, έπρεπε να χρησιμοποιήσουμε τρεις καινούργιες συναρτήσεις, όπως **Link Create2(char exp[]),** και οι δύο γνωστές συναρτήσεις από το **Stack ADT , void push(Link N), Link pop().** Την **stack** την αρχικοποιούμε στη **main** , δηλαδή **top=-1.** Την ορίζουμε ως **Link stack[Size]** (στο **Implementation**), δηλαδή **περιέχει δείκτες σε ρίζες** δέντρων. Επιπλέον, το **Size** είναι **defined** (στο **Implementation**).

Void push(Link N)

Παίρνει σαν **όρισμα δείκτη σε ρίζα** ενός δέντρου και λειτουργεί ως εξής:

1) **Ελέγχει** αν η στοίβα είναι **γεμάτη,** δηλαδή αν το **top>=Size-1,** εκτυπώνοντας **Stack Full if true.**
2) **Αυξάνει** το **top κατά 1.**
3) **Εισάγει** το **N** στη **στοίβα**

Link pop()

1) **Ορίζει** ένα **Link N; ,**δηλαδή **δείκτη σε ρίζα.**
2) **Ελέγχει** αν η στοίβα είναι άδεια.
3) **N=stack[top--]** , δηλαδή το **N** παίρνει την τιμή του **stack[top]** και αμέσως μετά **μειώνεται κατά 1** το **top**.
4) **Επιστρέφει** το **N**.

Link Create2(char exp[])

Σκοπός της συνάρτησης αυτής είναι η δημιουργία του **postfix expression tree**. Δέχεται σαν **όρισμα** ένα **πίνακα char** ή καλύτερα ένα **string.** Η **Create2** μπορεί να θεωρηθεί ως μία **εμπλουτισμένη-ειδικευμένη Create**. Λειτουργεί ως εξής:

1) **Αρχικοποιεί** ένα **int position=0**, που δηλώνει το χαρακτήρα του **string**, και **καταχωρεί τον 1^{ο} χαρακτήρα** στο **char ch**, δηλαδή **char ch=exp[position]; .** Επιπλέον ορίζει ένα **Link temp.**
2) Όσο το **ch!='\0' (EOF)(position++)**
    a. **Δεσμεύει χώρο** για την **temp** μέσω της **Create** και μετά εισάγει **temp->Item=ch;**
    b. **Ελέγχει** αν το **ch** περιέχει ένα από τους **ascii '0'…'9'**, και αν έχει τότε τους βάζει στη **στοίβα.**
    c. **Ελέγχει** αν το **ch** περιέχει ένα από τους **ascii '+', '*', '/'.**
        i. **True, temp->right=pop()** (**δεξιός όρος πράξης**) **temp->left=pop()** (**αριστερός όρος πράξης**) και **temp->item** περιέχει την πράξη. **Εισάγει** την **temp** στη **στοίβα.**
        ii. **False, continue.**
    d. Αυξάνει το **position** κατά **1, position++.**
    e. Εισάγει στο **ch** το καινούργιο χαρακτήρα και επαναλαμβάνεται η διαδικασία.
3) **Επιστρέφει** το **temp=pop().**

Int main(void)

Η **int main(void)** χωρίζεται σε δύο κομμάτια:

1) Στον έλεγχο της λειτουργικότητας του **Binary Tree ADT.** Δημιουργώ ένα δέντρο με τη βοήθεια της **Create** και της **MakeTree** και ελέγχω αν λειτουργούν τα **functions** του **Implementation.c**.
2) Στην επίλυση του προβλήματος ως προς τη **μεταθεματική έκφραση.**
    a. **Ορίζω** ένα **Link root (**δείκτη σε ρίζα δέντρου), ένα **char exp πίνακα 20 θέσεων.**
    b. **Διαβάζει** ένα **string** μέσω της **scanf** και την αποθηκεύει στο **exp.**
    c. **Αρχικοποιεί** τη **στοίβα (top=-1)**
    d. **Δημιουργεί** το **δέντρο έκφρασης(root=Create2(exp))** και το **εκτυπώνει(Print(root,0)).**