**עצים בינאריים**

**שאלה 1**

נתונות ההגדרות הבאות עבור עץ בינארי של מספרים אי-שליליים:

typedef struct treeNode

{

unsigned int data;

struct treeNode \*left;

struct treeNode \*right;

TreeNode;{

typedef struct tree

{

TreeNode \*root;

}Tree;

כתבו את הפונקציה :

Tree BuildTreeFromArray( int \*arr, int size)

הפונקציה מקבלת כפרמטר מערך בגודל ובונה ממנו עץ באופן הבא:

שורש העץ ייקבע להיות הערך שבמקום האמצעי במערך. את הערך אשר במרכז תת-המערך הימני (מימין למיקום השורש במערך) תציב הפונקציה כילד הימני של השורש. את הערך אשר במרכז תת-המערך השמאלי (משמאל למיקום השורש במערך) תציב הפונקציה כילד השמאלי של השורש. הפונקציה ממשיכה באופן דומה עם יתר תאי המערך.

ידוע כי העץ מכיל אך ורק ערכים אי-שליליים ולכן הערך 1- מייצג כי לא קיים בעץ צומת במיקום המתאים.

**לדוגמא:** עבור המערך

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1- | 17 | 1- | 3 | 1- | 1- | 1- | 55 | 1- | 4 | 1- | 15 | 11 | 2 | 3 |

ייבנה העץ:

55

15

3

17

2

4

11

3

**שאלה 2**

פעמים רבות הקומפיילר בונה עץ ביטוי לצורך חישוב ביטוי חשבוני שמופיע בקוד.

לשם הפשטות, נניח שבביטוי שלנו יופיעו רק מספרים חד ספרתיים (אופרנדים) ופעולות חשבוניות (בינריות) בסיסיות (אופרטורים) – חיבור, חיסור, כפל, חילוק ומודולו.

נניח שסביב כל אופרטור ושני האופרנדים עליהם הוא פועל מופיעים סוגריים.

למשל:

( 4 + 5 )

( (3 + 7) \* 9)

( (2 + 5) \* (8 / 2))

נגדיר עץ ביטוי באופן הבא:

צומת בעץ יכול להכיל אופרטור או אופרנד.

צומת עם בנים יכיל אופרטור אותו יש להפעיל על הביטויים המיוצגים ע"י תת העץ השמאלי ותת העץ הימני.

עלים יכילו אופרנדים.

דוגמאות:

(4+5)



((3 + 7) \* 9)



( (2 + 5 ) \* (8 / 2) )



כתבו תכנית הקולטת מהמשתמש מחרוזת המכילה ביטוי לפי ההגדרות בשאלה, בונה עץ ביטוי ומחשבת את ערך הביטוי תוך שימוש בעץ. על התכנית להציג הודעת שגיאה אם המחרוזת אינה ביטוי תקין.

עבור ביטוי תקין יש לבדוק:

* מספרים חד ספרתיים בלבד
* פעולות חיבור/חיסור/כפל/חילוק/מודולו בלבד
* סוגריים תקינים – לכל סוגר פותח ')' יש סוגר סוגר '(' במיקום המתאים, לדוגמא לא יהיה:

(2+(5) – הסוגר הפותח לפני הספרה 5 שגוי.

\* ניתן להניח שלא תהיה חלוקה ב-0

\* ניתן להניח שאין רווחים בין הספרות והאופרטורים**שאלה 3**

נתונים המבנים הבאים עבור עץ בינארי ועבור רשימה מקושרת:

typedef struct listNode {

int data;

struct listNode\* next;

} ListNode;

typedef struct list {

ListNode\* head;

ListNode\* tail;

} List;

typedef struct treeNode {

int data;

struct treeNode\* parent;

struct treeNode\* left;

struct treeNode\* right;

} TreeNode;

typedef struct tree{

TreeNode\* root;

List leafList; /\*רשימה מקושרת של כל העלים בעץ\*/

} Tree;

כתבו את הפונקציה הבאה:

#define LEFT 0

#define RIGHT 1

Tree AddLeaf (Tree tr, TreeNode \*p, int branchSelect, int data);

הפונקציה מקבלת עץ בינרי tr, צומת בעץ הבינרי p, נתון חדש data אותו יש להכניס לעץ מתחת ל p. הצד אליו יכנס הנתון החדש יקבע בהתאם לערך שיופיע ב branchSelect.

הערה: הניחו כי הנתון החדש יוכנס להיות עלה בעץ (כלומר, במקום אליו הוא אמור להיות מוכנס לא קיים כבר איבר בעץ).

שימו לב שבמבנה העץ מופיעה גם רשימה מקושרת של עלים (הרשימה מסודרת לפי סדר העלים משמאל לימין). הרשימה צריכה להיות מעודכנת ונכונה לאחר כל הוספה של איבר לעץ.

דוגמא לעץ כזה:



לשם ההבהרה, אם נקבל בקשה להכניס את המספר 16 בצד שמאל של העלה המכיל את הערך 27 בקריאה:

AddNode (tr, כתובת של העלה המכיל את הערך 27, LEFT , 16);

נקבל את מבנה הנתונים הבא:



**שאלה 4**

עץ בינארי מוגדר כך:

typedef struct treeNode{

int data;

struct treeNode\* left;

struct treeNode\* right;

} TreeNode;

typedef struct tree{

TreeNode\* root;

} Tree;

כתבו את פונקציה:

void printByLevels(Tree tr);

המדפיסה את הנתונים בעץ tr לפי רמות (קודם את הנתון ברמה 0, אחריו את נתונים ברמה 1, אחריהם את הנתונים ברמה 2, וכך הלאה), כל רמה תודפס משמאל לימין.

על הפונקציה לרוץ ביעילות , כאשר n הוא מספר הצמתים הכולל בעץ.

למשל עבור העץ:

יודפס: 3 2 5 4 7 9 1

הערות:

1. בכדי לעמוד בדרישות היעילות ייתכן ותרצו להשתמש במבנה נתונים נוסף.

2. אין להשתמש במשתנים סטאטיים.