**Ministerul Educației și Cercetării**

|  |
| --- |
| **Universitatea Tehnica a Moldovei** |

**Facultatea Calculatoare, Informatica si Microelectronica**

**Structuri de date și Algoritmi**

**Raport**

|  |
| --- |
| Lucrarea de Laborator nr. 6  Tema: Algoritmi de prelucrare a tipului abstract de date „Arbori binari” |

|  |  |
| --- | --- |
| A efectuat stundent(ul/a) grupei: | SI-212 |
| Numele si prenumele elev(ului/ei): | Vozian Vladimir |
|  |  |
| Profesorul: | Mititelu Vitalie |

|  |
| --- |
| Chișinău 2022 |

**Scopul lucrării:**

Obținerea deprinderilor practice de implementare și de utilizare a tipului abstract de date (TAD) „Arbore binar” cu asigurarea operațiilor de prelucrare de bază ale arborelui binar oarecare prin parcurgerea recursivă a nodurilor arborelui folosind algoritmi recursivi sau structurile respective de date „coadă” și „stivă”.

**Sarcina:**

Să se scrie 3 fișiere-text în limbajul C pentru implementarea și utilizarea TAD „Arbore binar” cu asigurarea operațiilor de prelucrare de bază ale arborilor binari oarecareprin parcurgerea nodurilor arborelui cu ajutorul algoritmilor recursivi sau iterativi:

1. Fișier antet cu extensia .h, care conține specificarea structurii de date a nodului arborelui binar (conform variantelor) și prototipurile funcțiilor de prelucrare de bază ale arborilor binari.

2. Fișier cu extensia .c sau .cpp, care conține implementările (codurile) funcțiilor declarate în fișierul antet.

3. Fișier al utilizatorului, funcția mаin() pentru prelucrarea arborelui binar oarecare cu afișarea la ecran a următorului meniu de opțiuni de bază:

1. Crearea nodurilor arborelui binar oarecare în memoria dinamică și introducerea informației despre nodurile arborelui de la tastatură în mod interactiv.

2. Afișarea informației despre nodurile arborelui la ecran.

3. Căutarea nodului în arbore.

4. Modificarea informației unui nod din arbore.

5. Determinarea numărului de noduri.

6. Determinarea înălțimii arborelui.

7. Eliberarea memoriei alocate pentru listă.

0. Ieșire din program.

**Varianta:**

16. Structura Canal TV cu câmpurile: denumirea, țara, telefonul, genul, ratingul.

**Rezumat**

### **Noțiunea de arbore. Arbori binari**

Matematic, un arbore este un graf neorientat conex aciclic.

În ştiinţa calculatoarelor, termenul de **arbore** este folosit pentru a desemna o structură de date care respectă definiţia de mai sus, însă are asociate un nod rădăcină şi o orientare înspre sau opusă rădăcinii.

Arborii sunt folosiţi în general pentru a modela o **ierarhie de elemente**.

Astfel, fiecare element (**nod**) poate deţine un număr de unul sau mai mulţi descendenţi, iar în acest caz nodul este numit **părinte** al nodurilor descendente (**copii** sau **fii**).

Fiecare nod poate avea un **singur nod părinte**. Un nod fără descendenţi este un **nod terminal**, sau **nod frunză**.

În schimb, există un singur nod fără părinte, iar acesta este întotdeauna **rădăcina arborelui** (**root**).

Un **arbore binar** este un caz special de arbore, în care fiecare nod poate avea maxim **doi descendenţi**:

* nodul stâng
* nodul drept.

În funcţie de elementele ce pot fi reprezentate în noduri şi de restricţiile aplicate arborelui, se pot crea structuri de date cu proprietăţi deosebite: heap-uri, arbori AVL, arbori roşu-negru, arbori Splay şi multe altele. O parte din aceste structuri vor fi studiate la curs şi în laboratoarele viitoare.

În acest articol ne vom concentra asupra unei utilizări comune a arborilor binari, şi anume pentru a reprezenta şi evalua expresii logice.

### **Reprezentarea arborilor binari**

Arborii binari pot fi reprezentați în mai multe moduri. Structura din spatele acestora poate fi un simplu vector, alocat dinamic sau nu, sau o structură ce folosește pointeri, așa cum îi vom reprezenta în acest articol.

[binary\_tree.h](https://ocw.cs.pub.ro/courses/_export/code/sd-ca/laboratoare/lab-08?codeblock=0)

typedef struct b\_node\_t b\_node\_t;

struct b\_node\_t {

/\* left child \*/

b\_node\_t \*left;

/\* right child \*/

b\_node\_t \*right;

/\* data contained by the node \*/

void \*data;

};

typedef struct b\_tree\_t b\_tree\_t;

struct b\_tree\_t {

/\* root of the tree \*/

b\_node\_t \*root;

/\* size of the data contained by the nodes \*/

size\_t data\_size;

};

Structura nodului de mai sus este clară:

* pointer către fiul stâng
* pointer către fiul drept
* pointer către date

Pentru a ne reaminti cum alocăm/dealocăm memorie:

/\*\*

\* Creates a new binary tree

\* @data\_size: size of the data contained by the tree's nodes

\* @return: pointer to the newly created tree

\*/

b\_tree\_t \*b\_tree\_create(size\_t data\_size);

/\*\*

\* Clear the whole memory used by the tree and its nodes

\* @b\_tree: the binary tree to be freed

\* @free\_data: function used to free the data contained by a node

\*/

void b\_tree\_free(b\_tree\_t \*b\_tree, void (\*free\_data)(void \*));

De exemplu, dacă dorim să creem un arbore binar ce conține elemente de tip \*char\*, codul arată astfel:

b\_tree\_t \*char\_tree = b\_tree\_create(sizeof(char));

b\_tree\_free(char\_tree, [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html));

### Parcurgerea arborilor

Se implementeaza foarte usor recursiv:

### **1. Preordine**

* Se parcurge **rădăcina**
* Se parcurge subarborele **stâng**
* Se parcurge subarborele **drept**

### **2. Inordine**

* Se parcurge subarborele **stâng**
* Se parcurge **rădăcina**
* Se parcurge subarborele **drept**

Exemplu:

static void \_\_b\_tree\_print\_inorder(b\_node\_t \*b\_node, void (\*print\_data)(void \*))

{

if (!b\_node)

return;

/\* TODO \*/

\_\_b\_tree\_print\_inorder(b\_node->left, print\_data);

print\_data(b\_node->data);

\_\_b\_tree\_print\_inorder(b\_node->right, print\_data);

}

void b\_tree\_print\_inorder(b\_tree\_t \*b\_tree, void (\*print\_data)(void \*))

{

\_\_b\_tree\_print\_inorder(b\_tree->root, print\_data);

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("\n");

}

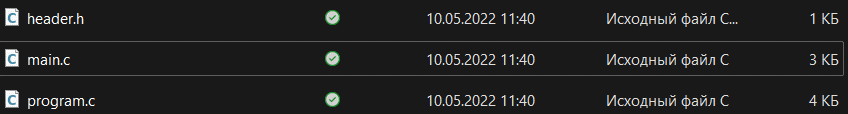
### **3. Postordine**

* Se parcurge subarborele **stâng**
* Se parcurge subarborele **drept**
* Se parcurge **rădăcina**

### **4. Lățime**

Se folosește o coadă, iar la fiecare pas se extrage din această coadă câte un nod și se adăugă înapoi în coadă nodul stâng, respectiv drept al nodului scos. Acest algoritm continuă până când coada devine goală.

**Toate file-urile:**



**Codul deplin al programului:**

**File-ul cu extensia .h:**

#ifndef HEADER\_H

#define HEADER\_H

struct canal{

    char \*denumire;

    char \*tara;

    unsigned long telefon;

    char \*gen;

    float rating;

};

struct tree{

    struct canal data;

    struct tree \*left, \*right;

};

void append(struct tree \*\*head, char \*denumire, char \*tara, unsigned long telefon, char \*gen, float rating);

void printIn(struct tree \*head);

struct tree \*search(struct tree \*head, char \*denumire);

void change(struct tree \*\*head, char \*denumire);

void numNode(struct tree \*head);

int heightOfTree(struct tree \*head);

void clearTree(struct tree \*\*head);

void citire(struct tree \*\*head, int num);

#endif

**File-ul cu extensia .c cu toate implementările:**

#include "header.h"

const char denumire[][20] = {"Moldova1","ProTV", "CTC", "THT", "Canal 3"};

const char tara[][20] = {"Moldova", "Romania", "Ucraina", "Turcia"};

const char gen[][20] = {"Filme", "Stiri", "pentru Copiii", "18+"};

void append(struct tree \*\*head, char \*denumire, char \*tara, unsigned long telefon, char \*gen, float rating){

    if(!(\*head)){

        (\*head) = (struct tree\*)malloc(sizeof(struct tree));

        (\*head) -> data.denumire = (char\*)malloc((strlen(denumire) + 1) \* sizeof(char));

        strcpy((\*head) -> data.denumire, denumire);

        (\*head) -> data.tara = (char\*)malloc((strlen(tara) + 1) \* sizeof(char));

        strcpy((\*head) -> data.tara, tara);

    (\*head) -> data.telefon = telefon;

        (\*head) -> data.gen = (char\*)malloc((strlen(gen) + 1) \* sizeof(char));

        strcpy((\*head) -> data.gen, gen);

        (\*head) -> data.rating = rating;

        (\*head) -> left = (\*head) -> right = NULL;

        return;

    }

    if(strcmp((\*head) -> data.denumire, denumire) <= 0)

        append(&(\*head) -> left, denumire, tara, telefon, gen, rating);

    if(strcmp((\*head) -> data.denumire, denumire) > 0)

        append(&(\*head) -> right, denumire, tara, telefon, gen, rating);

}

void citire(struct tree \*\*head, int num){

    srand(time(NULL));

    for(int i = 0; i < num; i++){

        append(&(\*head), denumire[rand() % 5], tara[rand() % 4], rand() % 100000 + 899999, gen[rand() % 4], rand() % 10 + 1);

    }

}

void printIn(struct tree \*head){

    if(head){

        printIn(head -> left);

        printf("%s %s %u %s %.2f\n",head -> data.denumire, head -> data.tara, head -> data.telefon, head -> data.gen, head -> data.rating);

        printIn(head -> right);

    }

}

struct tree \*search(struct tree \*head, char \*denumire){

    if(!head || strcmp(head -> data.denumire, denumire) == 0)

        return head;

    if(strcmp(head -> data.denumire, denumire) < 0)

        search(head -> left, denumire);

    if(strcmp(head -> data.denumire, denumire) > 0)

        search(head -> right, denumire);

}

void change(struct tree \*\*head, char \*denumire){

    struct tree \*temp = search(\*head, denumire);

    if(temp){

        char str[50];

        struct canal newCanal;

        printf("Dati denumirea noua al canalului:");

        scanf("%s", str);

        newCanal.denumire = (char\*)malloc((strlen(str) + 1) \* sizeof(char));

        strcpy(newCanal.denumire, str);

        printf("Dati tara noua al canalului:");

        scanf("%s", str);

        newCanal.tara = (char\*)malloc((strlen(str) + 1) \* sizeof(char));

        strcpy(newCanal.tara, str);

        printf("Dati genul nou al canalului:");

        scanf("%s", str);

        newCanal.gen = (char\*)malloc((strlen(str) + 1) \* sizeof(char));

        strcpy(newCanal.gen, str);

        printf("Dati telefonul nou al canalului:");

        scanf("%u", &newCanal.telefon);

        printf("Dati ratingul nou al canalului:");

        scanf("%f", &newCanal.rating);

        free(temp -> data.denumire);

        free(temp -> data.tara);

        free(temp -> data.gen);

        temp -> data = newCanal;

        return;

    }

    printf("Canal cu astfel de denumire nu a fost gasit.\n");

}

int count = 0;

void numNode(struct tree \*head){

    if(head){

        numNode(head -> left);

        numNode(head -> right);

        count++;

    }

}

int heightOfTree(struct tree \*head){

    if(head == NULL)

        return 0;

    int left = heightOfTree(head -> left);

    int right = heightOfTree(head -> right);

    if(left >= right)

        return left + 1;

    else return right + 1;

}

void deleteTree(struct tree \*\*head){

    if(\*head == NULL)

        return;

    deleteTree(&(\*head) -> left);

    deleteTree(&(\*head) -> right);

    free((\*head) -> data.denumire);

    free((\*head) -> data.tara);

    free((\*head) -> data.gen);

    free(\*head);

}

void clearTree(struct tree \*\*head){

    deleteTree(&(\*head) -> left);

    deleteTree(&(\*head) -> right);

    (\*head) = NULL;

}

**File-ul cu funcția main():**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "program.c"

#include "header.h"

void menu(){

    printf("1) Adaugarea elementrlor in lista.\n");

    printf("2) Cautarea nodului in arbore.\n");

    printf("3) Modificarea informatiei unui nod din arbore.\n");

    printf("4) Determinarea numarului de noduri.\n");

    printf("5) Determinarea inaltimii arborelui.\n");

    printf("6) Eliberarea memoriei alocata pentru arbore.\n");

    printf("0) Iesirea din program.\n");

}

int main(){

    struct tree \*head = NULL;

    int num, option;

    char str[30], str2[30];

    srand(time(NULL));

    system("cls");

    printf("Dati numarul de canale:");

    scanf("%d",&num);

    citire(&head,num);

    do{

        system("cls");

        printIn(head);

        printf("\n");

        menu();

        printf("Alegeti optiunea:");

        scanf("%d",&option);

        switch(option){

            case 1:

                printf("Dati numarul de canale:");

                scanf("%d",&num);

                citire(&head,num);

            break;

            case 2:

                printf("Dati denumirea canalului care doriti sal cautati:");

                scanf("%s",str);

                struct tree \*temp = search(head, str);

                printIn(temp);

            break;

            case 3:

                printf("Dati denumirea canalului care doriti sal modificati:");

                scanf("%s",str);

                change(&head, str);

            break;

            case 4:

                numNode(head);

                printf("Arborele are %d noduri.", count);

                count = 0;

            break;

            case 5:

                printf("Inaltimea arborelui este: %d.",heightOfTree(head));

            break;

            case 6:

                clearTree(&head);

            break;

            case 0:

            break;

        }

        printf("\nPress any key to continue...");

        getch();

    }while(option != 0);

    /\*

    printf("123");

    append(&head, "qwe", "dhgdfgh", 123123, "gdsf", 12.2);

    append(&head, "safgs", "dhgdfgh", 123123, "gdsf", 12.2);

    append(&head, "ljf", "dhgdfgh", 123123, "gdsf", 12.2);

    append(&head, "fdgh", "dhgdfgh", 123123, "gdsf", 12.2);

    printIn(head);

    \*/

    // struct tree \*temp = search(head, "ljf");

    //

    // if(temp)

    //  printf("%s \n", temp -> data.denumire);

    //

    //change(&head, "ljf");

    //printIn(head);

    // numNode(head);

    // printf("%d\n", count);

    //

    // int height = heightOfTree(head);

    // printf("%d", height);

    //

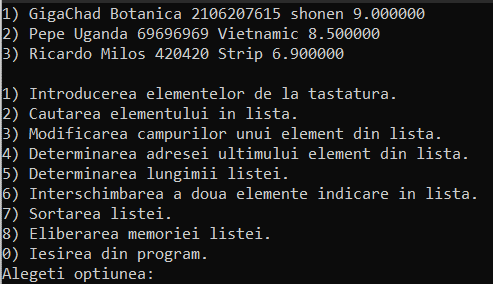
    // clearTree(&head);

    // printIn(head);

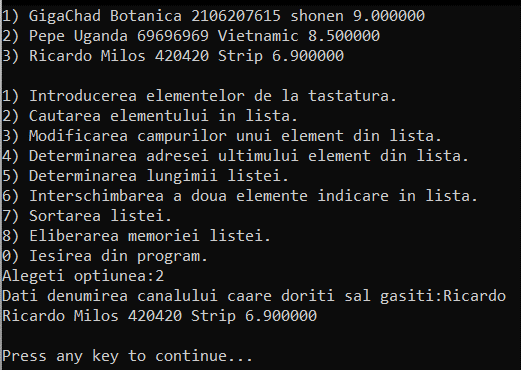
    return 0;

}

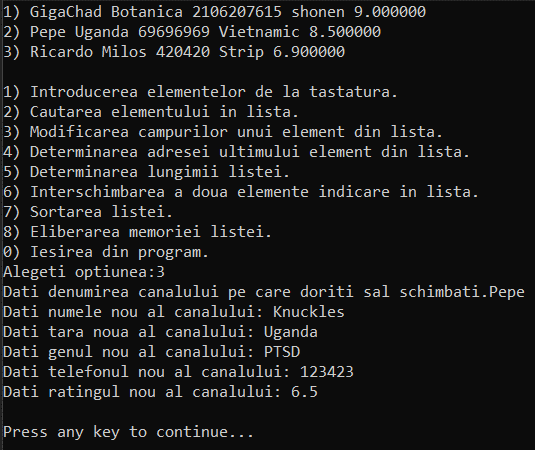
**Exemple de execuție a programului:**



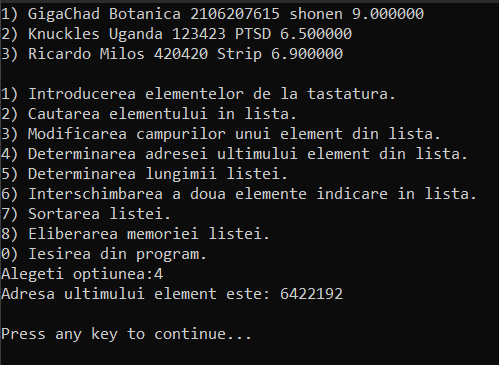
Cautarea elementului din listă:



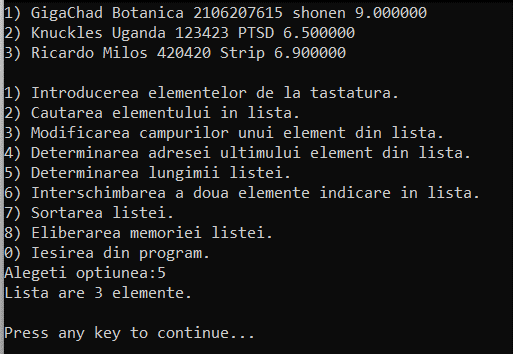
Modificarea câmpului unui element:



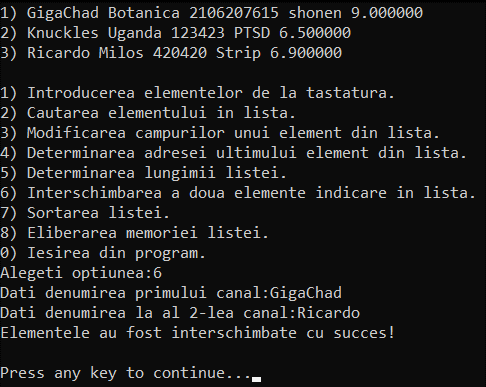
Adresa ultimului element:



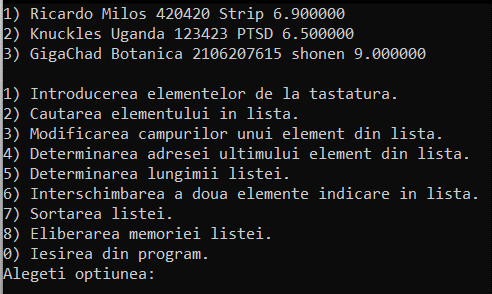
Determinarea lungimii listei:



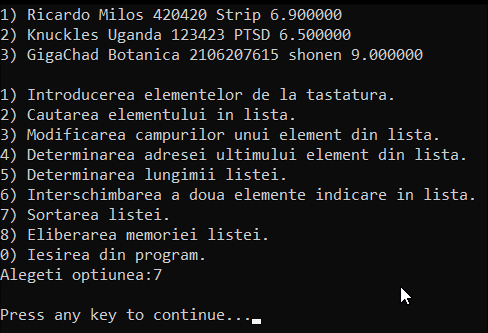
Interschimbarea a doua elemente:

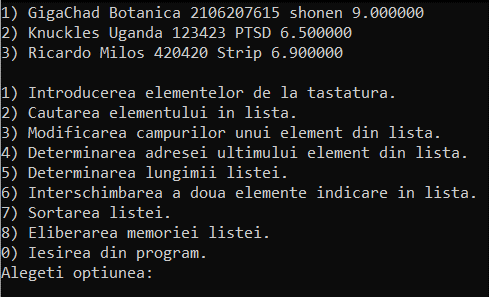


Aici vedem ca sau schimbat cu succes:

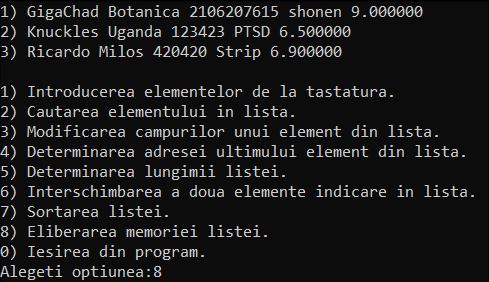


Sortarea listei:





Eliberarea memoriei:



**Concluzii pe baza efectuării lucrării de laborator:**

* Deci am lucrat cu lista simplu lănțuită și am îndeplinit sarcinele similare cu laboratoarele trecute când am lucrat cu structurile.
* Lucrul asupra lucrării este util în întelegerea mai profundă a lucrului cu listele simplu lănțuite și a operării mai efectivă cu informația.

**Surse**

<https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic>