Găujăneanu Nicoleta Monica
Grupa 414D

Lucrare 1 SDA

Problema 2

Pseudocod:

```
Insert(ora, min, sec){
tempNode = create_node(ora,min,sec) // creez nodul pe care doresc sa-l inserez
//creez nodul curent si nodul parinte pentru a ma ajuta sa parcurg arborele
current_node = create_node()
parent_node = create_node()

if root_node == NULL
        root_node = tempNode //daca arborele este gol tempNode va devenii root

else
current = root // incep parcurgerea
parent = null

while true
        parent = curent // parcurg arborele si actualizez nodurile
        if( timp < parent.timp) // daca timpul e mai mic trebuie sa merg la stanga
                current = current->left
                if current == NULL //daca nu mai am ce sa parcurg inserez elementul
                        insert(tempNode)
                                return
        else //merg in dreapta
```

```
            if current == null
                insert(tempNode)
                    return



Search(ora,min,sec){
current_node=root
while(current.timp != timp)
    if(timp < current.timp) // daca timpul e mai mic merg la stanga
        current = current->left
    else
        current = current->right

    if current == null //daca am ajuns la final si nu am gasit elementul intorc null
        return null
return current;
}

Inorder_traversal(root_node){
if root != null
    inorder_traversal(root.left) // parcurg tot arborele din stanga
    inorder_traversal(root.left) // parcurg tot arborele din dreapta
}
```

Găujăneanu Nicoleta Monica
Grupa 414D

COD:

```
#include <stdio.h>
#include <stdlib.h>

struct node {  //structura arbore
   int ora;
   int min;
   int sec;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

// functie de inserare a unui element
void insert(int ora, int min, int sec) {
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   //aloc memorie pentru creearea unui nou nod pentru a putea fi adaugat in arbore
   struct node *current;
   struct node *parent;
   // noduri pentru a putea tine cont de nodul curent si de nodul parinte


   // initializez nodul
```

```
  tempNode->ora = ora;
  tempNode->min = min;
  tempNode->sec = sec;
  tempNode->leftChild = NULL;
  tempNode->rightChild = NULL;

  //daca arborele este gol, nodul devine root
  if(root == NULL) {
    root = tempNode;
  } else { // daca nu incep parcurgerea arborelui
    current = root; //root va fi nodul curent
    parent = NULL;

    while(1) {
      parent = current; // actualizez nodul parent cu nodul curent ca sa pot trece mai departe prin arbore

      //parcurg arborele in stanga
      if(ora < parent->ora || (ora == parent->ora && min < parent->min) || (ora == parent->ora && min == parent->min && sec < parent->sec) ) {
        current = current->leftChild;

        //daca gasesc loc liber in partea stanga inserez elementul
        if(current == NULL) {
          parent->leftChild = tempNode;
          return;
        }
      }  //parcurg arborele in dreapta
      else {
```

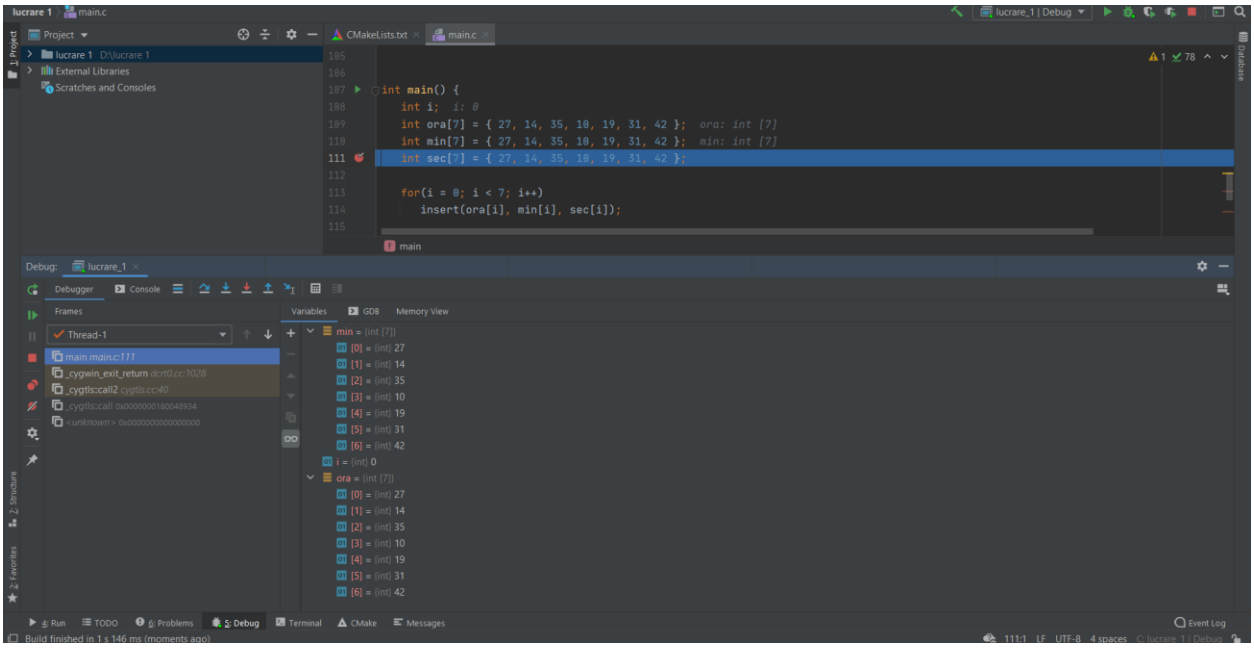
```
            current = current->rightChild;

            //inserez elementul la dreapta
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}

struct node* search(int ora, int min, int sec) { // functia de cautare
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->ora != ora || current->min != min || current->sec != sec) { //fac aceasta operatie cat timp nodul nu are informatiile cautate in el
      if(current != NULL){
         printf("%d ",current->ora);
         printf("%d ",current->min);
         printf("%d ",current->sec);
      }
      //parcurgerea in stanga
      if(ora < current->ora || (ora == current->ora && min < current->min) || (ora == current->ora && min == current->min && sec < current->sec)) {
         current = current->leftChild;
      }
      //parcurgearea in dreapta
```

```
    else {
      current = current->rightChild;
    }

    //daca termin de parcurs arborele si nu gasesc nodul intorc null
    if(current == NULL) {
      return NULL;
    }
  }

  return current; //altfel intorc nodul
}



void inorder_traversal(struct node* root) { //parcurgerea in ordine
  if(root != NULL) {
    inorder_traversal(root->leftChild);
    printf("%d\n ",root->ora);
    printf("%d\n ",root->min);
    printf("%d\n ",root->sec);
    inorder_traversal(root->rightChild);
  }
}

int main() {
  int i; //contor pt
  int ora[7] = { 27, 14, 35, 10, 19, 31, 42 };
```

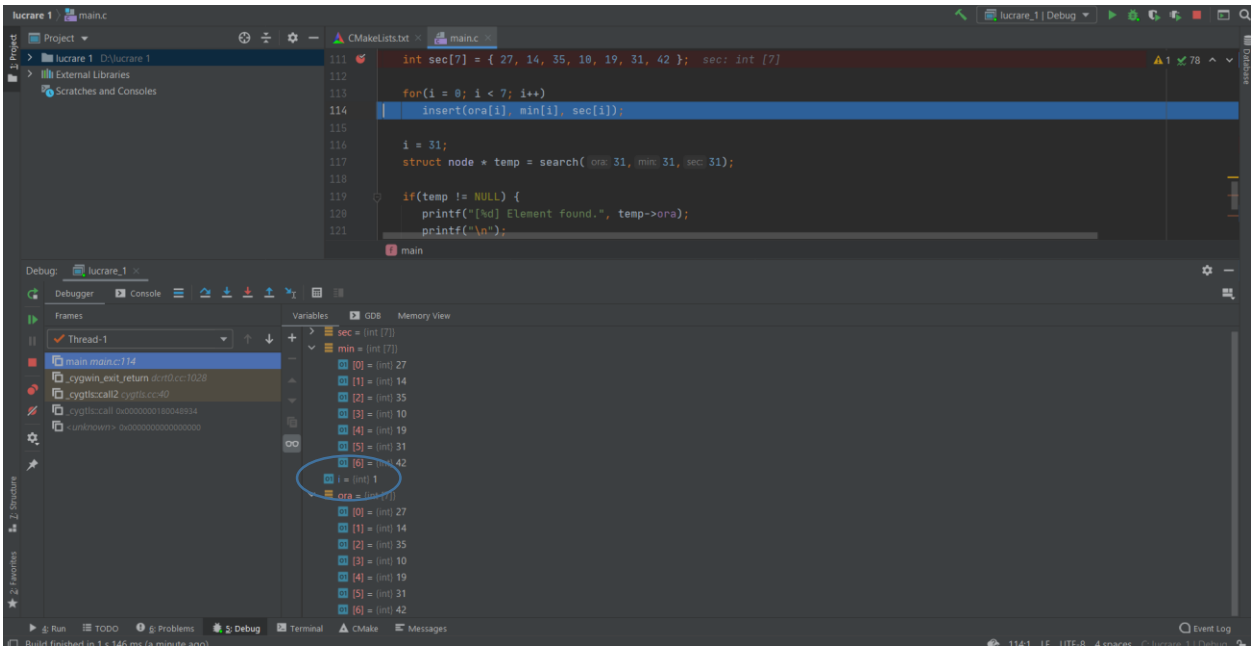
```
int min[7] = { 27, 14, 35, 10, 19, 31, 42 };
int sec[7] = { 27, 14, 35, 10, 19, 31, 42 };

for(i = 0; i < 7; i++)
   insert(ora[i], min[i], sec[i]);

i = 31;
struct node * temp = search(31,31,31);

if(temp != NULL) {
   printf("[%d] Element found.", temp->ora);
   printf("\n");
}else {
   printf("[ x ] Element not found (%d).\n", i);
}

i = 15;
temp = search(15,15,15);

if(temp != NULL) {
   printf("[%d] Element found.", temp->ora);
   printf("\n");
}else {
   printf("[ x ] Element not found (%d).\n", i);
}
```

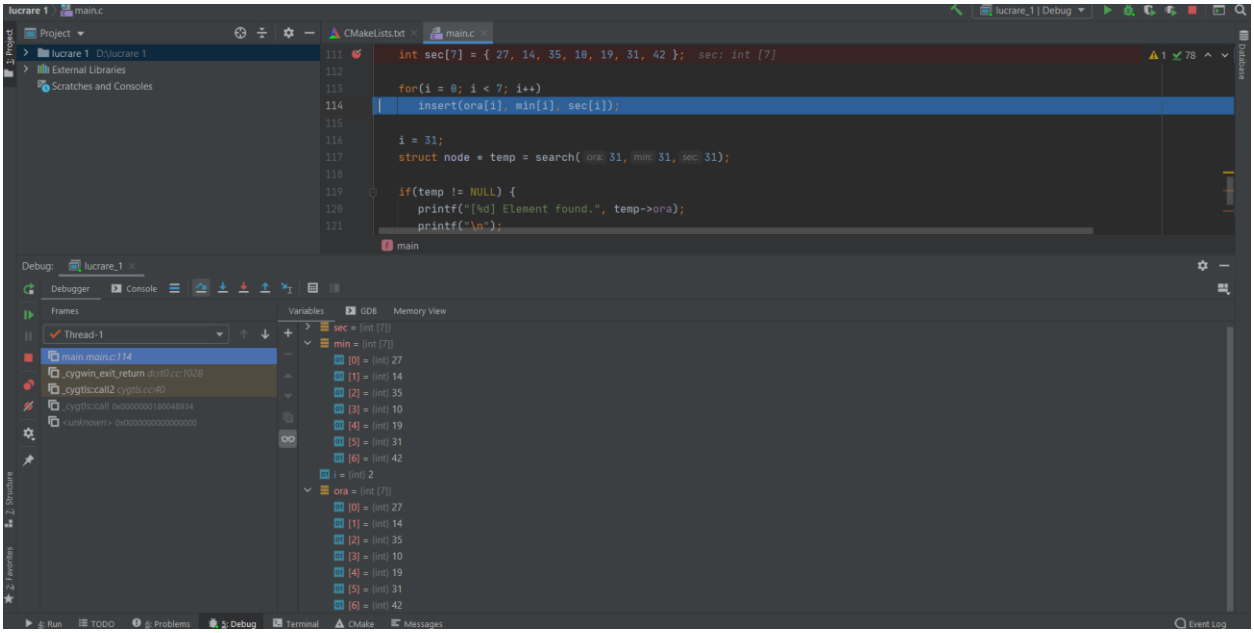
```
    printf("\nInorder traversal: ");
    inorder_traversal(root);



    return 0;
}
```

Verificare:

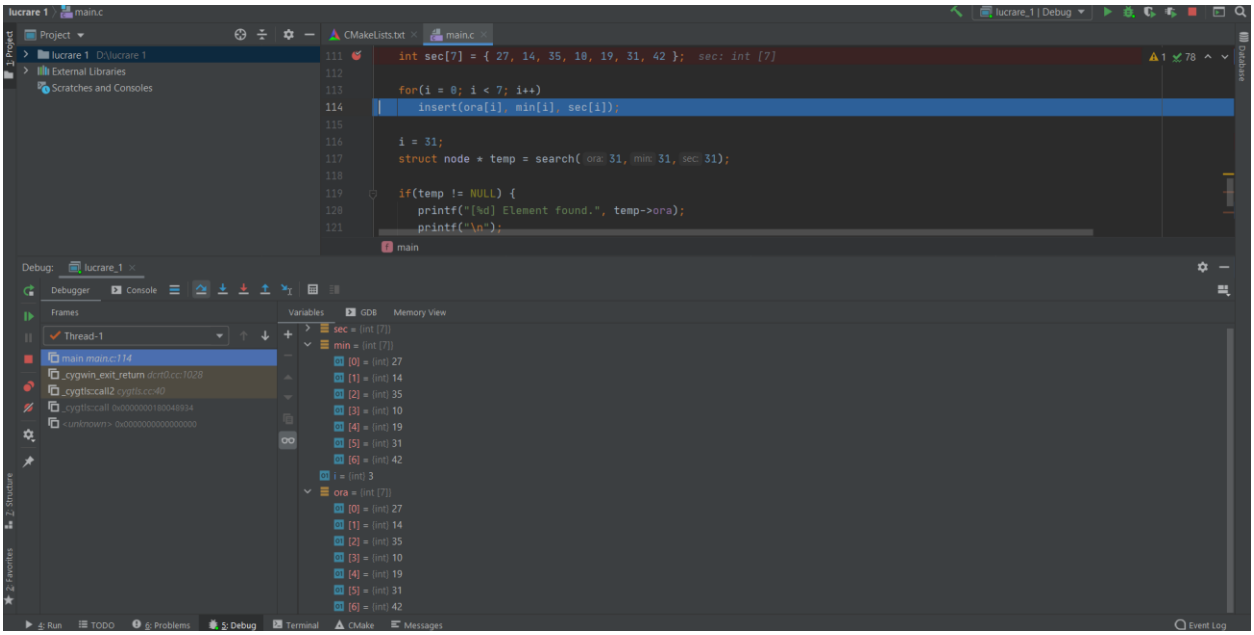


Găujăneanu Nicoleta Monica
Grupa 414D

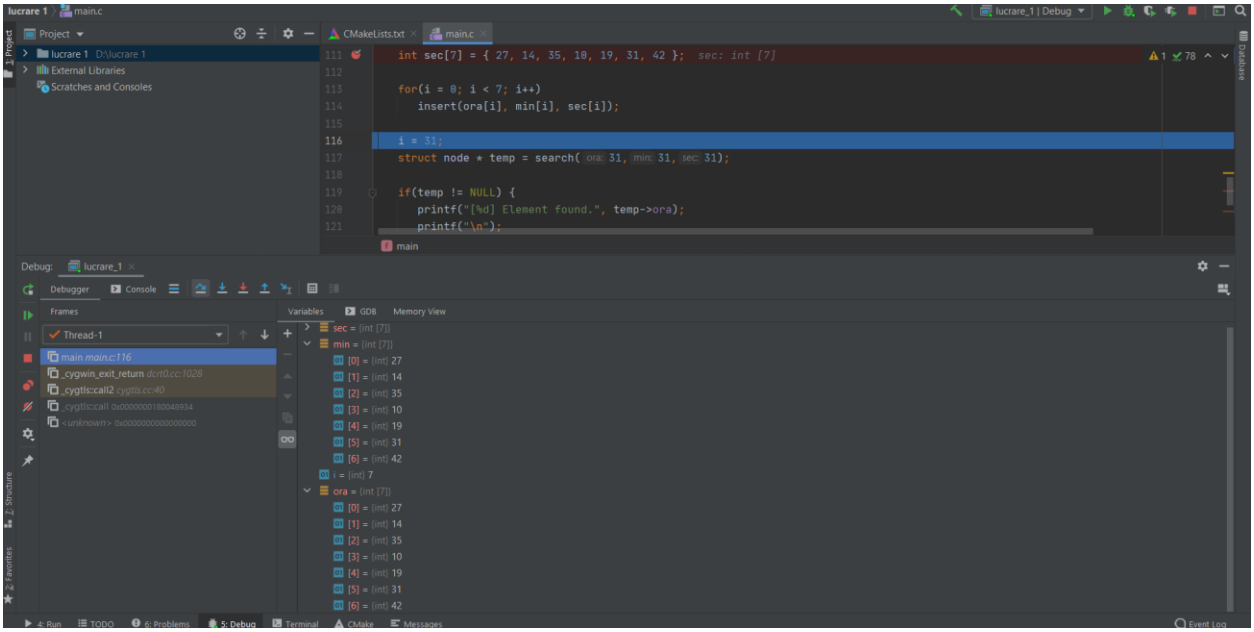




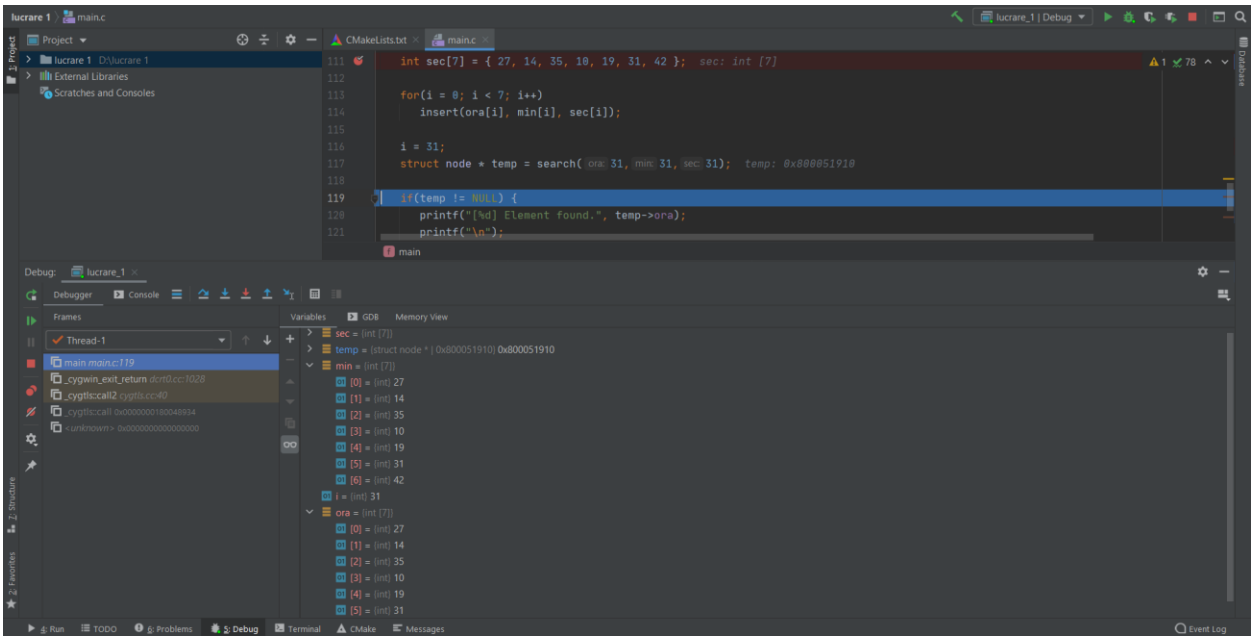
Găujăneanu Nicoleta Monica
Grupa 414D

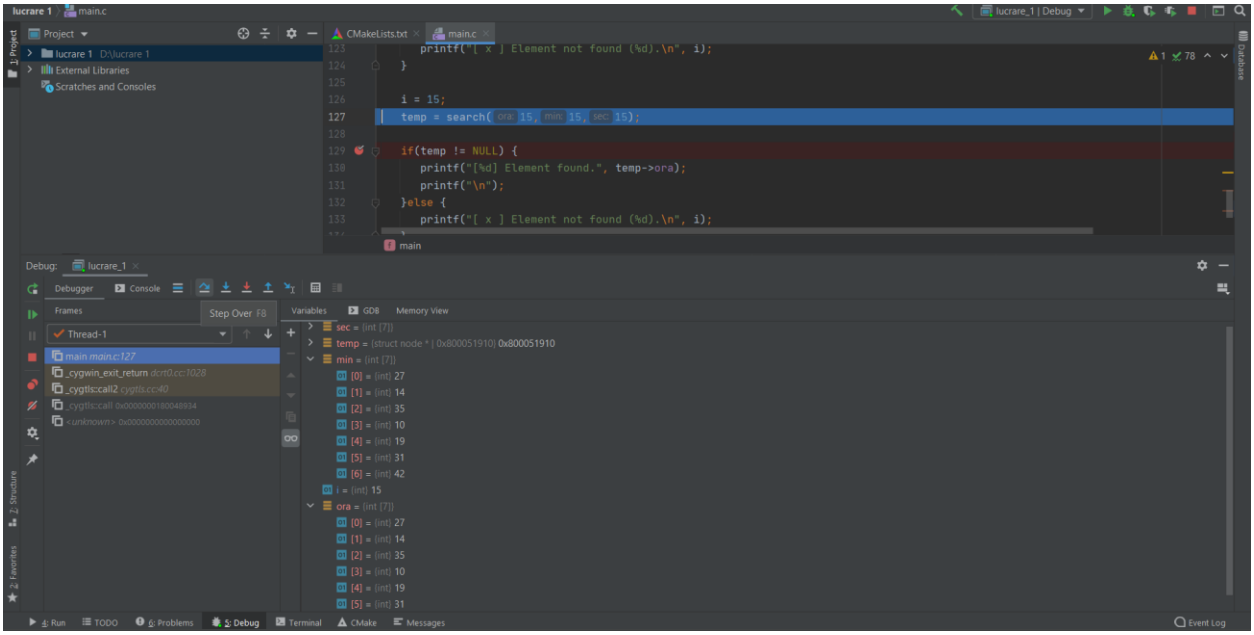


Am rulat pas cu pas pana la elementul 7, dar nu am mai inserat fiecare pas

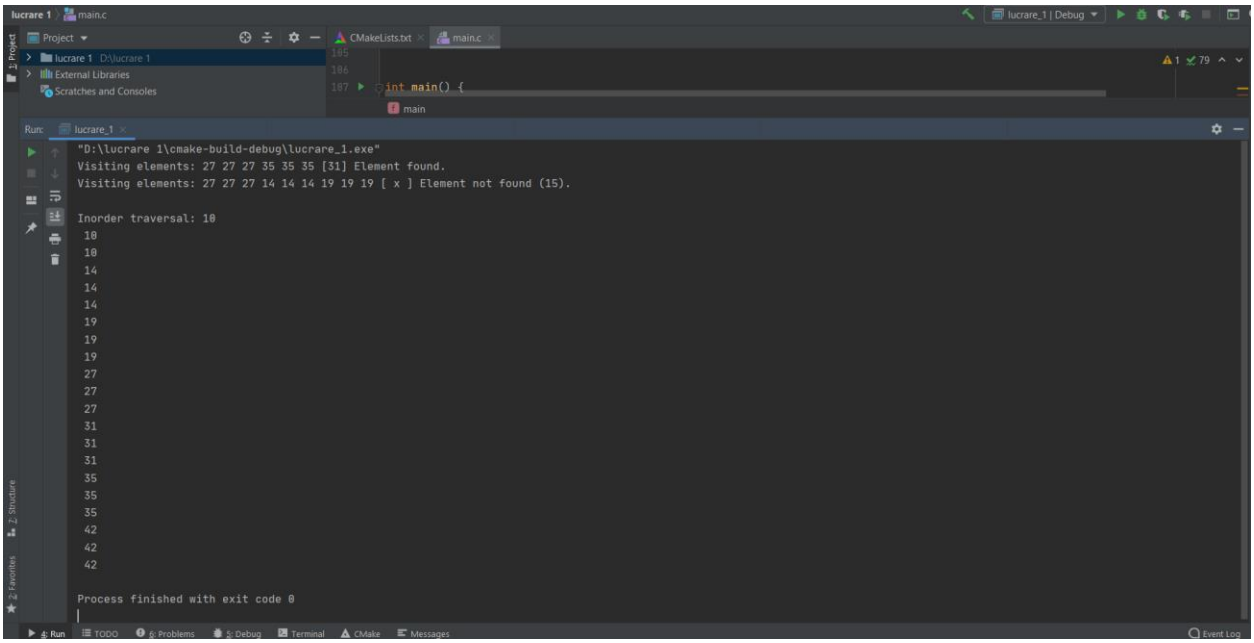


Găujăneanu Nicoleta Monica
Grupa 414D





Găujăneanu Nicoleta Monica
Grupa 414D



Problema 1:

PSEUDOCOD:

creare nod

daca lista este goala

    primul nod din lista devina nodul creat

altfel

    se adauga nodul creat la incepului cozii

adaugare nod

se sterge nodul

se parcurge coada

daca primul nod e nul

coada este goala

altfel

cat timp nodul este diferit de null

afisam valoare nod

se trece la urmatorul nod

COD :

```
#include <stdio.h>
#include <stdlib.h>


struct node{        // nod pentru coada dublu inlantuita

   int info;                // valoarea din nod
   struct node *left;     // nodul stang
   struct node *right;   // nodul drept

};


struct node* new_node(int a){    // functie creare nod nou coada

   struct node* p;        // creeaza o structura de tip node
```

```
    p=(struct node*)malloc(sizeof(struct node));        // alocare dinamica a nodului

    p->data=a;              // in nodul p inseram valoarea a, unde a este parametrul lui new_node
    p->left=NULL;       // atribuim NULL nodului din stanga
    p->right=NULL;     // atribuim NULL nodului din dreapta

    return p;
}



void add_queue(struct node* r, int a){      // adaugare coada

    struct node* p;

    r->data = a;
    p=new_node(-1);
    r->next=p;
    r=p;

}


int del_queue(struct node* f, struct node* r){  // stergere coada

    struct node* p;
    int x;
```

```
    if (f->next != r ){

        p=f->next;
        x=p->data;
        f->next=p->next;
        free(p);
        return x;

    }
}


void parcurgere_queue(struct nod *first){    // listare + cautare coada

    struct node* tmp;
    int i = 0;

    tmp = first;

    if (first == NULL){

        printf("Coada este goala.\n");
        return;

    }

    while (tmp!=NULL){
```

```
        printf("nodul %d are valoarea %d\n", i, tmp->info);
        i++;
        tmp = tmp->right;

    }

}



int main()
{

    return 0;
}
```