

1.

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

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

```
#include <assert.h>
```

```
/*
```

```
 * Ne definim structurile pentru coada si pentru fiecare nod
```

```
 * care va fi un element al listei
```

```
*/
```

```
typedef struct Node Node;
```

```
typedef struct Queue Queue;
```

```
/*
```

```
 * Acesta este nodul din lista care contine informatia
```

```
 * adica valoarea din noi si pointerul de next, care arata spre
```

```
 * elementul din fata si pointerul de prev care arata spre elementul
```

```
 * din spate
```

```
*/
```

```
struct Node {
```

```
    int key;
```

```
    Node *next;
```

```
    Node *prev;
```

```
};
```

```
/*
```

```
 * Queue este coada propriu zisa cu cei doi pointeri de head si tail
```

```
 * care pointeaza catre capetele cozilor, spre primul element(head)
```

```
 * si spre ultimul element(tail)
```

```
*/
```

```

struct Queue {

    Node * head;

    Node * tail;

};

/*

* Aceasta functie intai aloca spatiu pentru coada noastra

* Malloc(sizeof(queue)) ii spune compilatorului sa aloce

* pentru variabila qu memorie pe heap cat pentru structura queue

* iar mai apoi setam cei doi pointeri de head si de tail

* pe NULL

*/

Queue * create_queue()

{

    Queue * qu = malloc(sizeof(Queue));

    qu -> head = NULL;

    qu -> tail = NULL;

    return qu;

}

/*

* Functie care m-a ajutat pe mine la implementare

* Verifica daca nodul de head este NULL SI cel de tail

* este NULL, caz in care coada este goala

*/

unsigned is_empty(Queue * qu)

{

    return qu -> head == NULL && qu -> tail == NULL;

}

```

```

/*
* Functie de adaugare in coada
* Aloca memorie pentru nd, nodul nostru pe care vrem sa-l
* adaugam cu valoarea key, si next si prev, cei doi pointeri
* pentru fiecare element din lista
* Daca coada este goala adaugam nodul la inceput, deci ne folosim
* de cei doi pointeri head si tail
* dar daca coada nu este goala, va trebui sa mutam pointerii de next si de prev
*/

```

```

void enqueue(Queue * qu, int key)

```

```

{
    Node * nd = malloc(sizeof(Node));

    nd -> key = key;

    nd -> next = NULL;

    nd -> prev = NULL;

    if (is_empty(qu)) {
        qu -> head = nd;
        qu -> tail = nd;
    } else {
        qu->tail->prev = qu -> tail;
        qu -> tail -> next = nd;
        qu -> tail = nd;
    }
}

```

```

/*
* Functie care sterge din coada un element
* Daca coada este goala nu o sa facem nimic si iesim din functie
* Dar daca coada nu este goala, mutam pointerul de head la nextul

```

```

* lui insusi, caz ce va face ca primul element sa nu mai existe in
* lista, iar apoi il stergem din memorie cat inca mai avem acces direct
* la el
*/

```

```

void dequeue(Queue * qu)

```

```

{
    assert(!is_empty(qu));

    Node * tmp_node = qu -> head;

    qu -> head = tmp_node -> next;

    qu -> head->prev = NULL;

    free(tmp_node);

    if (qu -> head == NULL)

        qu -> tail = NULL;

}

```

```

/*
* Functie cu care cautam in coada
* ce facem este sa luam pointerul de head, iar mai apoi
* ne mutam din next in ntext pana cand ajungem sa gasim nodul
* Daca nu il gasim afisam nod element negasit
*/

```

```

void search(Queue *qu, int key)

```

```

{
    Node *tmp_node = qu->head;

    do {

        if (tmp_node->key == key) {

            printf("%d\n", key);

            return;

        }

        tmp_node = tmp_node->next;
    }
}

```

```
} while(tmp_node != qu->tail);

printf("element negasit\n");
}

/*
 * Functie cu care parcurg coada si o afisez
 * element cu element
 * Mergem pana cand head-ul este diferit de tail si afisam
 */
void print(Queue *qu)
{
    Node *tmp_node = qu->head;
    do {
        printf("%d ", tmp_node->key);
        tmp_node = tmp_node->next;
    } while(tmp_node != qu->tail);
    printf("\n");
}

int main(int argc, char *argv[])
{
    // cream coada
    Queue *my_queue = create_queue();

    //inseram numere de la 1 la 10
    for (int i = 1; i <= 11; i++) {
        enqueue(my_queue, i);
    }
```

```

// cautam 5

search(my_queue, 5);

// afisam coada

print(my_queue);

// scoatem 3 numere

for (int i = 0; i < 3; i++) {

    dequeue(my_queue);

}

// o afisam din nou

print(my_queue);

return 0;

}

```

Testez functionalitatea programului:

The screenshot shows a C++ IDE with the following components:

- Editor:** Displays the C++ code with line numbers 143 to 167. The code includes comments in Romanian and function calls like `enqueue`, `search`, `print`, and `dequeue`.
- Run Window:** Shows the execution path: `C:\Users\m\CLionProjects\untitled9\cmake-build-debug\untitled9.exe`.
- Output:** Displays the program's output:
 

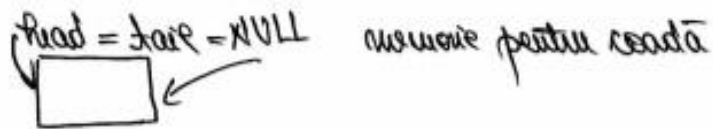
```

5
1 2 3 4 5 6 7 8 9 10
4 5 6 7 8 9 10

```
- Status:** Indicates "Process finished with exit code 0".

1. Gooda vida

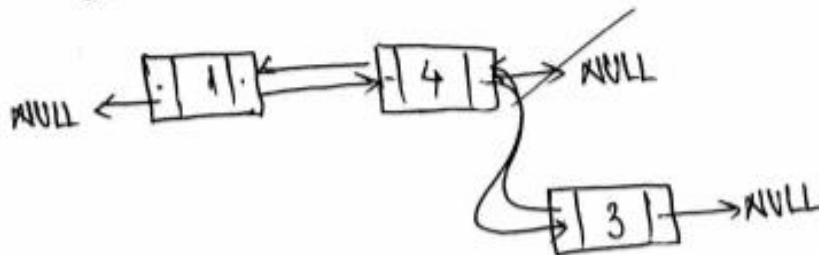
2. Create - queue



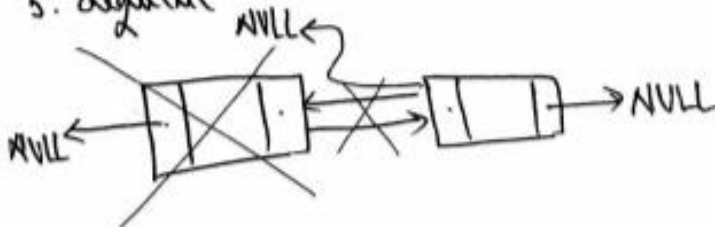
3. is empty



4. enqueue (3)



5. dequeue

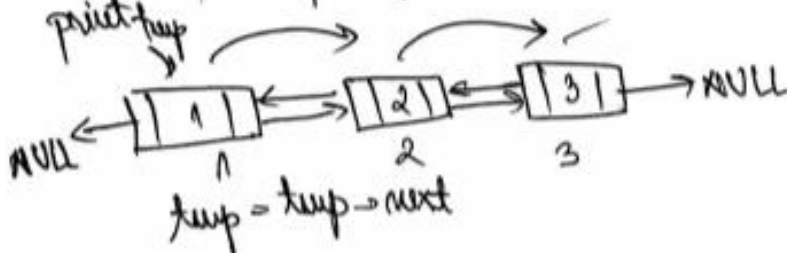


search (3)



$temp = temp \rightarrow next$

print temp



2.

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

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

```
/*
```

```
* Structura unui nod care contine data noastra insemnand
```

```
* datele din zi precum ora, minutul si secunda
```

```
* nodul de left si nodul de right care sunt de fapt,
```

```
* copiii radacinei si ale celorlalte noduri
```

```
*/
```

```
typedef struct bnode{
```

```
    int ora;
```

```
    int min;
```

```
    int sec;
```

```
    struct bnode *left;
```

```
    struct bnode *right;
```

```
}bnode;
```

```
/*
```

```
* Functie de insecare
```

```
* Ca paramentrii ii trimite adresa arborelui nostru
```

```
* Facem acest lucru ca sa fie modificat, in caz contrar
```

```
* Compilatorul ar fi facut o copie a arborelui si
```

```
* ar fi adaugat in functie, mai exact pe zona de memorie
```

```
* a functiei si la finalul functiei s-ar fi sters tot
```

```
* caz in care nu s-ar fi modificat
```

```
* In cazul in care arborele e vid, cream nodul, si il inseram
```

```
* el va fi radacina
```

```
* In caz ca arborele nu este vid, comparam dupa ora
```



\* dupa minut si dupa secunda

\* In cazul in care este mai mic decat ora/min/sec adaugam

\* in stanga, in caz ca este mai mare adaugam in dreapta

\*/

```
void insert_node(bnode **l, int ora, int min, int sec){
```

```
    if(*l == NULL){
```

```
        *l = malloc(sizeof(bnode));
```

```
        (*l)->ora = ora;
```

```
        (*l)->min = min;
```

```
        (*l)->sec = sec;
```

```
        (*l) -> left = NULL;
```

```
        (*l) -> right = NULL;
```

```
    }
```

```
    else if(ora > (*l)->ora){
```

```
        insert_node((&(*l) -> right), ora, min, sec);
```

```
    } else if (min > (*l)->min) {
```

```
        insert_node((&(*l) -> right), ora, min, sec);
```

```
    } else if (sec > (*l)->sec) {
```

```
        insert_node((&(*l) -> right), ora, min, sec);
```

```
    } else {
```

```
        insert_node((&(*l) -> left), ora, min, sec);
```

```
    }
```

```
}
```

```
/*
```

\* Functie cu care cautam in arbore, dupa ora, min, sec

\* In cazul in care gasim nodul cu datele dorite il afisam

\* Caz contrar, vedem in ce interval ne situam si ne deplasam

\* in arbore.

\*/

```

bnode *search_tree(bnode *l, int ora, int min, int sec){

    if(l == NULL)

        return NULL;

    if(l->ora == min && l->min == min && l->sec == sec)

        return l;

    if(ora > l->ora){

        return search_tree(l->right, ora, min ,sec);

    } else if (min > l->min) {

        return search_tree(l->right, ora, min ,sec);

    } else if (sec > l->sec) {

        return search_tree(l->right, ora, min ,sec);

    } else {

        return search_tree(l->left, ora, min, sec);

    }

}

/*
* Stanga-radacina-dreapta afisare, cum i se mai zice
* Cat timp arborele NU ESTE NULL, afisam cum am mentionat
* mai sus
* caz contrar iesim.
*/

void in_order_traversal(bnode *l){

    if(l != NULL){

        if(l->left != NULL)

            in_order_traversal(l->left);

        printf("ORA: %d MIN: %d SEC: %d\n",l->ora, l->min, l->sec);

```

```
        if(l -> right != NULL)
            in_order_traversal(l -> right);
    }
}
```

```
int main(){

    // arborele

    bnode *head = NULL;

    // inserare date

    insert_node(&head, 1, 2, 3);

    insert_node(&head, 3, 4, 5);

    insert_node(&head, 6, 7, 8);

    //afisare

    in_order_traversal(head);


    //inserare date

    insert_node(&head, 2, 2, 3);

    insert_node(&head, 18, 1, 2);

    insert_node(&head, 29, 1, 4);

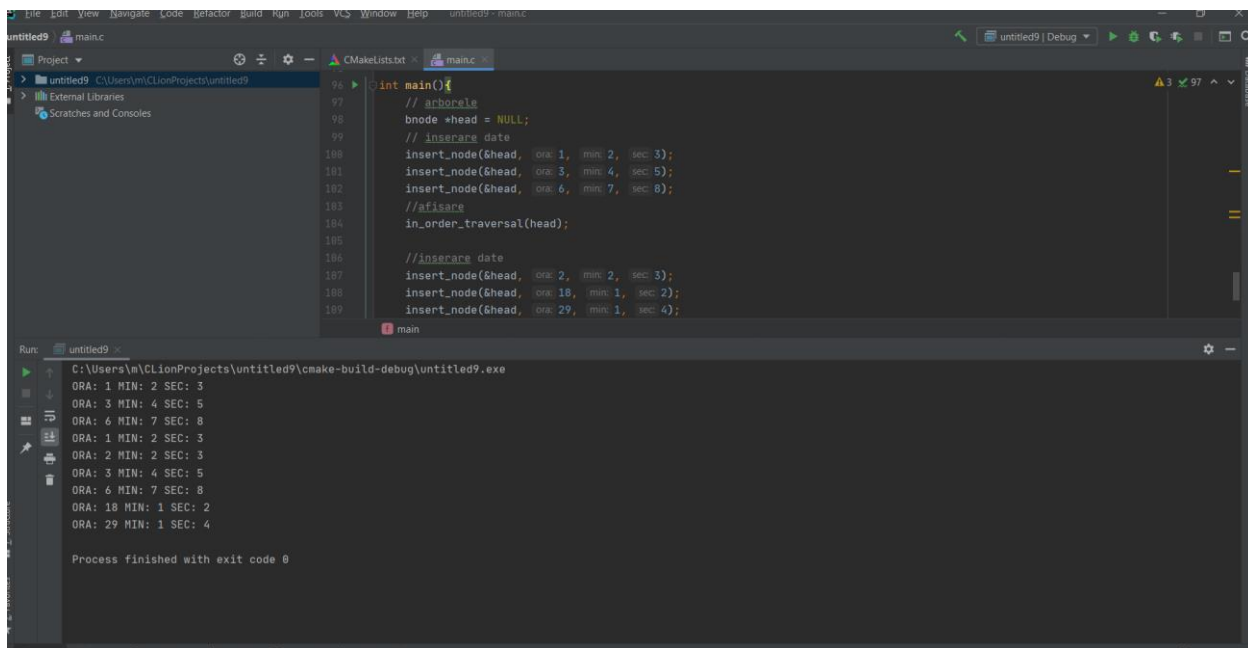
    // afisare

    in_order_traversal(head);

    return 0;

}
```

Testez functionalitatea programului:



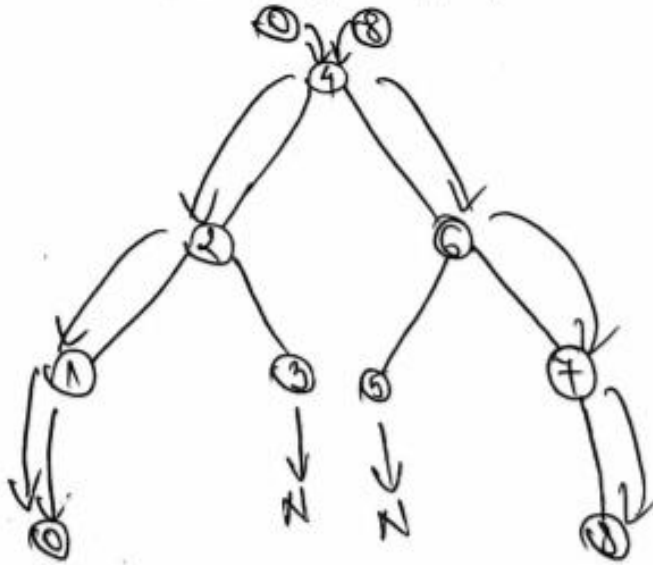
The screenshot shows a C++ IDE with a project named 'untitled9'. The main.cpp file contains the following code:

```
96 int main() {
97     // arborele
98     bnode *head = NULL;
99     // insereaza date
100     insert_node(&head, ora: 1, min: 2, sec: 3);
101     insert_node(&head, ora: 3, min: 4, sec: 5);
102     insert_node(&head, ora: 6, min: 7, sec: 8);
103     // afisare
104     in_order_traversal(head);
105
106     //insereaza date
107     insert_node(&head, ora: 2, min: 2, sec: 3);
108     insert_node(&head, ora: 18, min: 1, sec: 2);
109     insert_node(&head, ora: 29, min: 1, sec: 4);
110 }
```

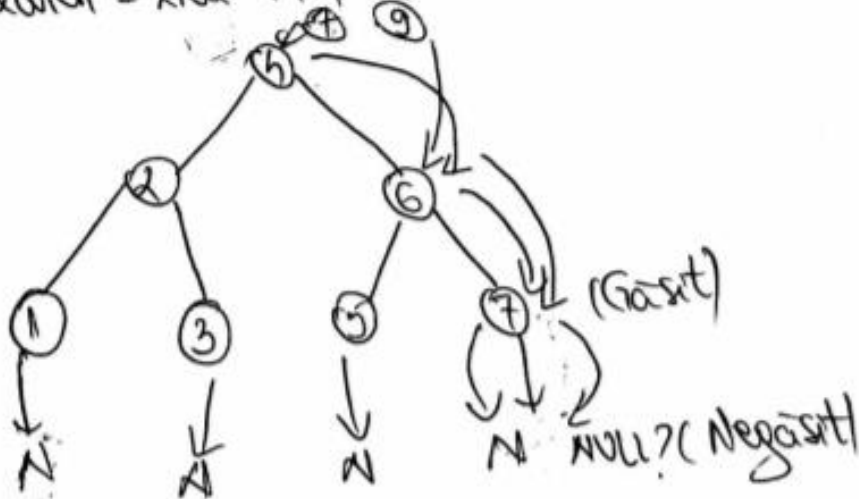
The Run window shows the output of the program:

```
C:\Users\m\CLionProjects\untitled9\cmake-build-debug\untitled9.exe
ORA: 1 MIN: 2 SEC: 3
ORA: 3 MIN: 4 SEC: 5
ORA: 6 MIN: 7 SEC: 8
ORA: 1 MIN: 2 SEC: 3
ORA: 2 MIN: 2 SEC: 3
ORA: 3 MIN: 4 SEC: 5
ORA: 6 MIN: 7 SEC: 8
ORA: 18 MIN: 1 SEC: 2
ORA: 29 MIN: 1 SEC: 4
Process finished with exit code 0
```

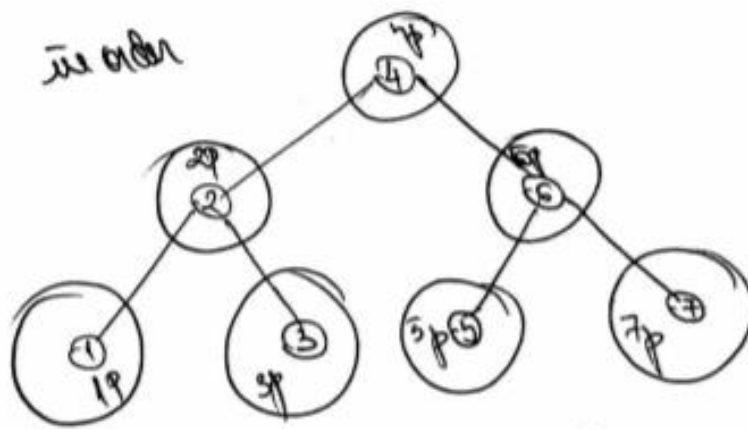
1) Insert Node (8), (0)



2) Search - tree (7), (9)



in order



1 2 3 4 5 6 7 → după sortare

3.

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

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

```
#include <assert.h>
```

```
/*
```

```
* =====
```

```
* IMPORTANT
```

```
* MODALITATE DE IMPLEMENTARE:
```

```
* COADA CARE CONTINE CA SI DATA NUMARUL DE TELEFON, SI MAI RETINEM PE LANGA
```

```
* UN ABC IN CARE VOM ADAUGA APELURILE IN CARE RETINEM NUMARUL DE TELEFON, TIPUL
```

```
* APELULUI DAR SI ORA/MIN/SEC LA CARE AU FOST TRIMITE/PRIMITE
```

```
* NE FOLOSIM DE ACEASTA IMPLEMENTARE PENTRU A RETINE DATELE INTR-UN MOD EFICIENT
```

```
* =====
```

```
*/
```

```
/*
```

```
* Structura unui nod care contine data noastra insemnand
```

```
* datele din zi precum ora, minutul si secunda, iar acum
```

```
* am adaugat si tip-ul apelului si numarul de telefon
```

```
* ideea e ca fiecare numar de telefon va avea apeluri
```

```
* inregistrare, primite si trimise dupa un interval de ora
```

```
* iar eu o sa retin acest lucru in arbore sub aceasta
```

```
* forma
```

```
*/
```

```
typedef struct bnode{
```

```
    int ora;
```

```
    int min;
```

```
    int sec;
```

```
    int tip_apel;
```

```
    int numar_telefon;
```

```

    struct bnode *left;

    struct bnode *right;

}bnode;

/*
* Functie de inseare
* Am explicat la ex2 modul de functionare,
* Diferenta este ca acum ne folosim si de ora tipul apelului
* dar si de numarul de telefon insemnand ca le vom adauga si pe acestea
*/
void insert_node(bnode **l, int ora, int min, int sec, int tip_apel, int numar_telefon){
    if(*l == NULL){
        *l = malloc(sizeof(bnode));

        (*l)->ora = ora;

        (*l)->min = min;

        (*l)->sec = sec;

        (*l)->tip_apel = tip_apel;

        (*l)->numar_telefon = numar_telefon;

        (*l) -> left = NULL;

        (*l) -> right = NULL;

    }

    else if(ora > (*l)->ora){
        insert_node(&(*l) -> right, ora, min, sec, tip_apel, numar_telefon);

    } else if (min > (*l)->min) {
        insert_node(&(*l) -> right, ora, min, sec, tip_apel, numar_telefon);

    } else if (sec > (*l)->sec) {
        insert_node(&(*l) -> right, ora, min, sec, tip_apel, numar_telefon);

    } else {
        insert_node(&(*l) -> left, ora, min, sec, tip_apel, numar_telefon);

    }
}

```

```

}

/*
* Functie cu care cautam in arbore, dupa ora, min, sec, tip-ul apelului dar si numarul
* de telefon
*
*/

bnode *search_tree(bnode *l, int ora, int min, int sec, int numar_telefon, int tip_apel){

    if(l == NULL)

        return NULL;

    if(l->ora == min && l->ora == min && l->sec == sec && l->tip_apel == tip_apel &&
    l->numar_telefon == numar_telefon)

        return l;

    if(ora > l->ora){

        return search_tree(l -> right, ora, min ,sec, numar_telefon, tip_apel);

    } else if (min > l->min) {

        return search_tree(l -> right, ora, min ,sec, numar_telefon, tip_apel);

    } else if (sec > l->sec) {

        return search_tree(l -> right, ora, min ,sec, numar_telefon, tip_apel);

    } else {

        return search_tree(l -> left, ora, min, sec, numar_telefon, tip_apel);

    }

}

/*
* Stanga-radacina-dreapta afisare, cum i se mai zice
* Cat timp arborele NU ESTE NULL, afisam cum am mentionat
* mai sus

```



\* caz contrar iesim.

\*/

```
void in_order_traversal(bnode *l){
```

```
    if(l != NULL){
```

```
        if(l -> left != NULL)
```

```
            in_order_traversal(l -> left);
```

```
        printf("ORA: %d MIN: %d SEC:%d TIP_APEL:%d NUMAR_TELEFON:%d\n",l->ora, l->min, l->sec, l->tip_apel,
```

```
            l->numar_telefon);
```

```
        if(l -> right != NULL)
```

```
            in_order_traversal(l -> right);
```

```
    }
```

```
}
```

/\*

\* Ne definim structurile pentru coada si pentru fiecare nod

\* care va fi un element al listei

\*/

```
typedef struct Node Node;
```

```
typedef struct Queue Queue;
```

/\*

\* Acesta este nodul din lista care contine informatia

\* adica valoarea din noi si pointerul de next, care arata spre

\* elementul din fata si pointerul de prev care arata spre elementul

\* din spate

\* Retinem numarul de telefon pentru care vom avea un arbore binar

\* in care vom retine tipuriel de apeluri, la ora la care au fost

\* primite/trimise dar si numarul de telefon care l-a apelat/pe care

\* l-a apelat

\*/

```
struct Node {
```

```
    int numar_telefon;
```

```
    bnode *l;
```

```
    Node *next;
```

```
    Node *prev;
```

```
};
```

```
/*
```

\* Queue este coada propriu zisa cu cei doi pointeri de head si tail

\* care pointeaza catre capetele cozilor, spre primul element(head)

\* si spre ultimul element(tail)

\*/

```
struct Queue {
```

```
    Node * head;
```

```
    Node * tail;
```

```
};
```

```
/*
```

\* Aceasta functie intai alocu spatiu pentru coada noastra

\* Malloc(sizeof(queue)) ii spune compilatorului sa aloce

\* pentru variabila qu memorie pe heap cat pentru structura queue

\* iar mai apoi setam cei doi pointeri de head si de tail

\* pe NULL

\*/

```
Queue * create_queue()
```

```
{
```

```
    Queue * qu = malloc(sizeof(Queue));
```

```

qu -> head = NULL;

qu -> tail = NULL;

return qu;
}

```

```

/*
* Functie care m-a ajutat pe mine la implementare
* Verifica daca nodul de head este NULL SI cel de tail
* este NULL, caz in care coada este goala
*/

```

```

unsigned is_empty(Queue * qu)
{
    return qu -> head == NULL && qu -> tail == NULL;
}

```

```

/*
* Functie de adaugare in coada
* Aloca memorie pentru nd, nodul nostru pe care vrem sa-l
* adaugam cu valoarea key, si next si prev, cei doi pointeri
* pentru fiecare element din lista
* Daca coada este goala adaugam nodul la inceput, deci ne folosim
* de cei doi pointeri head si tail
* dar daca coada nu este goala, va trebui sa mutam pointerii de next si de prev
*
* UPDATE:
* avem un arbore binar caruia trebuie sa ii dam valoarea din parametrii
* Pentru acest lucru apelam functia de adaugare cu parametrii trimisi mai sus
*
* ACEASTA FUNCTIE SE FOLOSESTE PENTRU UN NUMAR DE TELEFON NOU
*/

```

```
void enqueue(Queue * qu, int numar_telefon_adaugare, int ora, int minut, int secunda, int tip_apel
```

```
,int numar_telefon)
```

```
{
    Node * nd = malloc(sizeof(Node));

    insert_node(&(nd->l), ora, minut, secunda, tip_apel, numar_telefon_adaugare);

    nd->numar_telefon = numar_telefon;

    nd -> next = NULL;

    nd -> prev = NULL;

    if (is_empty(qu)) {
        qu -> head = nd;
        qu -> tail = nd;
    } else {
        qu->tail->prev = qu -> tail;
        qu -> tail -> next = nd;
        qu -> tail = nd;
    }
}
```

```
/*
```

```
* Aceasta functie realizeaza adaugarea in arborele binar pentru un numar de telefon
```

```
* Pe scurt ceea ce se intampla este cautam in coada pana cand gasim elementul iar mai apoi
```

```
* adaugam la abc-ul nodului
```

```
*/
```

```
void enqueue_existent(Queue * qu, int ora, int minut, int secunda, int tip_apel
```

```
,int numar_telefon)
```

```
{
    Node *tmp_node = qu->head;

    do {
        if (tmp_node->numar_telefon == numar_telefon) {
            insert_node(&(tmp_node->l), ora, minut, secunda, tip_apel, numar_telefon);
        }
    } while (tmp_node->next != NULL);
}
```

```

        return;

    }

    tmp_node = tmp_node->next;
} while(tmp_node != qu->tail);
}

/*
 * Functie care sterge din coada un element
 * Daca coada este goala nu o sa facem nimic si iesim din functie
 * Dar daca coada nu este goala, mutam pointerul de head la nextul
 * lui insusi, caz ce va face ca primul element sa nu mai existe in
 * lista, iar apoi il stergem din memorie cat inca mai avem acces direct
 * la el
 */
void dequeue(Queue * qu)
{
    assert(!is_empty(qu));

    Node * tmp_node = qu -> head;

    qu -> head = tmp_node -> next;

    qu -> head->prev = NULL;

    free(tmp_node);

    if (qu -> head == NULL)

        qu -> tail = NULL;
}

/*
 * Acum cautam dupa numarul de telefon
 */
void search(Queue *qu, int key)
{

```

```

Node *tmp_node = qu->head;

do {

    if (tmp_node->numar_telefon == key) {

        printf("NUMAR TELEFON: %d\n", key);

        in_order_traversal(tmp_node->l);

        return;

    }

    tmp_node = tmp_node->next;

} while(tmp_node != qu->tail);

printf("element negasit\n");

}

/*

* Functie cu care parcurg coada si o afisez

* element cu element

* Mergem pana cand head-ul este diferit de tail si afisam

*

* ACUM afisam numarul de telefon

*/

void print(Queue *qu)

{

    Node *tmp_node = qu->head;

    do {

        printf("NR TELEFON :%d ", tmp_node->numar_telefon);

        in_order_traversal(tmp_node->l);

        tmp_node = tmp_node->next;

    } while(tmp_node != qu->tail);

    printf("\n");

}

```

```

int main(){

    Queue *my_queue = create_queue();

    for (int i = 1; i < 10; i++) {

        enqueue(my_queue, 1234 + i, 1 + i, 2 + i, 3 + i, 4 + i, 123 + i);

    }

    // cautam 5

    search(my_queue, 124);

    // afisam coada

    print(my_queue);

    // scoatem 3 numere

    for (int i = 0; i < 3; i++) {

        dequeue(my_queue);

    }

    // o afisam din nou

    print(my_queue);

    return 0;

}

```

The screenshot shows a C++ IDE with the following code in `main.c`:

```

271 int main(){
272     Queue *my_queue = create_queue();
273     for (int i = 1; i < 10; i++) {
274         enqueue(my_queue, 1234 + i, 1 + i, 2 + i, 3 + i, 4 + i, 123 + i);
275     }
276 }

```

The Run window shows the output of the program:

```

C:\Users\m\CLionProjects\untitled9\cmake-build-debug\untitled9.exe
NUMAR TELEFON: 124
ORA: 2 MIN: 3 SEC:4 TIP_APEL:5 NUMAR TELEFON:1235
NR TELEFON :124 ORA: 2 MIN: 3 SEC:4 TIP_APEL:5 NUMAR TELEFON:1235
NR TELEFON :125 ORA: 3 MIN: 4 SEC:5 TIP_APEL:6 NUMAR TELEFON:1236
NR TELEFON :126 ORA: 4 MIN: 5 SEC:6 TIP_APEL:7 NUMAR TELEFON:1237
NR TELEFON :127 ORA: 5 MIN: 6 SEC:7 TIP_APEL:8 NUMAR TELEFON:1238
NR TELEFON :128 ORA: 6 MIN: 7 SEC:8 TIP_APEL:9 NUMAR TELEFON:1239
NR TELEFON :129 ORA: 7 MIN: 8 SEC:9 TIP_APEL:10 NUMAR TELEFON:1240
NR TELEFON :130 ORA: 8 MIN: 9 SEC:10 TIP_APEL:11 NUMAR TELEFON:1241
NR TELEFON :131 ORA: 9 MIN: 10 SEC:11 TIP_APEL:12 NUMAR TELEFON:1242

NR TELEFON :127 ORA: 5 MIN: 6 SEC:7 TIP_APEL:8 NUMAR TELEFON:1238
NR TELEFON :128 ORA: 6 MIN: 7 SEC:8 TIP_APEL:9 NUMAR TELEFON:1239
NR TELEFON :129 ORA: 7 MIN: 8 SEC:9 TIP_APEL:10 NUMAR TELEFON:1240
NR TELEFON :130 ORA: 8 MIN: 9 SEC:10 TIP_APEL:11 NUMAR TELEFON:1241
NR TELEFON :131 ORA: 9 MIN: 10 SEC:11 TIP_APEL:12 NUMAR TELEFON:1242

Process finished with exit code 0

```