1) Programul incepe prin definirea bibloteciilor necesare: stdio.h, conio.h si stdlib.h

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

struct node
{
int data;
struct node *link;
}*front, *rear;

// am construit functia de search astfel incat sa cautam un element dupa pozitia sa
void search(int position)
{
    struct node *temp;
    temp = front;
    int i = 0;
    // verific daca coada este goala
    if (front == NULL)
    {
        printf("Queue underflow\n");
    }
    // daca nu este goala, o parcurg si afisez pozitia si valoarea ce se afla la acea pozitie
    else {
        while (i != position) {
            // temp = temp->link;
            i++;
        }
        printf("His position is: %d \n", i);
        printf("The value of the searched element is: %d \n", temp->data);
        }
```

```
}

int get(int position) {
   struct node *temp;
   temp = front;
   int i = 0;
   // verificam daca exista elemente in coada
   if (front == NULL)
   {
      printf("Queue underflow\n");
   }
   // parcurgem nodurile pana la pozitia nodului dorit
   else {
      while (i != position) {
         temp = temp->link;
         i++;
      }
      return temp->data;
   }
}

void insert()
{
   // am luat un nod pentru a-l adauga in coada
   struct node *temp;
   temp = (struct node*)malloc(sizeof(struct node));
   printf("Enter the element to be inserted in the queue: ");
   scanf("%d", &temp->data);
   // nodul nu este legat la coada
   temp->link = NULL;
   // daca coada nu exista, atunci adaug temp la ea
```

```
        if (rear == NULL)
        {
            front = rear = temp;
        }
    // daca avem elemente in coada, adaugam temp in continuarea ei
        else
        {
            rear->link = temp;
        rear = temp;
        }
}

void delete() {
    struct node *temp;
    temp = front;
    // verific prima data daca coada are elemente in ea
    if (front == NULL) {
        printf("Queue underflow\n");
        front = rear = NULL;
    }
    // daca exista elemente in coada, stergem elementul ce are data = x
    // (stergem nodul ce il contine pe x)
    else {
        printf("The deleted element from the queue is: %d\n", front->data);
        front = front->link;
        free(temp);
    }
}

void display() {
    struct node *temp;
```

```
   temp = front;
   int cnt = 0;
   // verific daca coada este goala
   if (front == NULL)
   {
      printf("Queue underflow\n");
   }
   // daca nu este goala, o parcurg si afisez elementele pe rand
   else {
      printf("The elements of the stack are:\n");
      while (temp) {
         printf("%d\n", temp->data);
         temp = temp->link;
         cnt++;
      }
   }
}

   int main()
   {
   int choice, position;
   printf ("LINKED LIST IMPLEMENTATION OF QUEUES\n\n");
   // am folosit un switch pentru a ne alege optiunea pe care dorim sa o executam
   // optiunile sunt : inserare, eliminare, afisare
   do
      {
      printf("1. Insert\n2. Delete\n3. Display\n4. Search\n5. Get\n6. Exit\n\n");
      printf("Enter your choice:");
      scanf("%d",&choice);
      switch(choice) {
         case 1:
```

```
    insert();
    break;
    case 2:
    delete();
    break;
    case 3:
    display();
    break;
    case 4:
    printf("Say the position of the number we want to find: ");
    scanf("%d", &position);
    search(position);
    break;
    case 5:
    printf("The result from extraction is: ");
    int result = get(position);
    printf("The number is:%d ", result);
    break;
    case 6:
    exit(0);
    break;
    default:
    printf("Sorry, invalid choice!\n");
    break;
  }
} while(choice!=5);
return 0;
}
```

2) Variabile utilizate:

- ora, min, sec= variabila de tip int, care margheaza cifra din campul de informatie al nodului;

- left = pointer de tip struct nod, care marcheaza campul de legatura la stanga al nodului

- right= pointer de tip struct nod, care marcheaza campul de legatura la dreapta al nodului

Programul incepe prin definirea bibloteciilor necesare: stdio.h, conio.h si stdlib.h

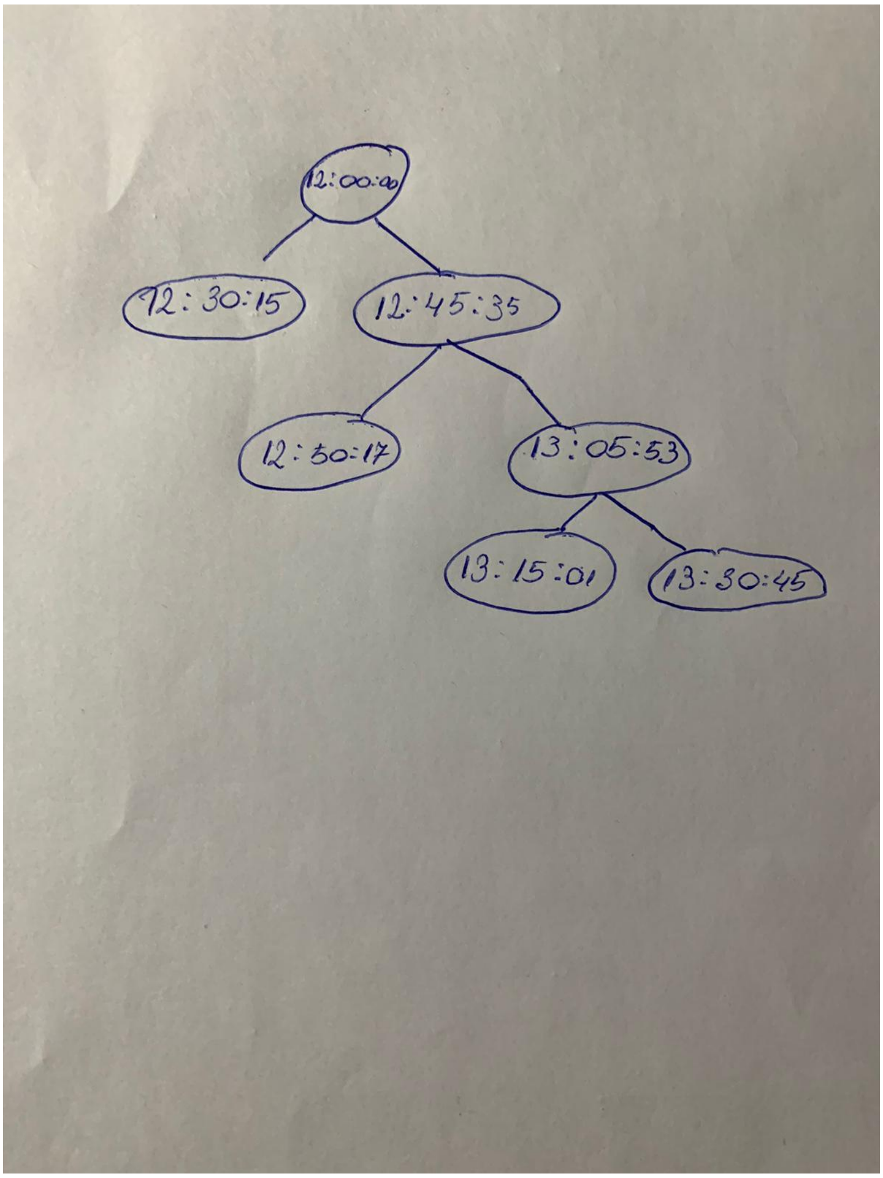
Se predefineste structura unui nod

Se creeaza functia de construire a unui nou nod (se aloca spatiu prin functia malloc a unui element))
{p->data primeste x;

p->left=p->right primesc NULL;}

Se construieste arborele

Se creeza functia de parcurgere a arborilor binari care nu intoarce nimic



```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
// Arbori binari
struct bnode {
int ora;
 int min;
 int sec;
 struct bnode* left;
struct bnode* right; }

struct bnode* new_tree_node(int a);

struct bnode* build_abe(int n, int A[]);

struct bnode* build_abc(struct bnode*r, int a);
struct bnode* search_abc(struct bnode*r, int a);



void ldr(struct bnode* r);
void dlr(struct bnode* r);
void lrd(struct bnode* r);

int main() {

    int i;

    struct bnode* roote = NULL;
    struct bnode* rootc = NULL;

    roote=build_abe(10,sir);
    for (i=0; i < 10; i++)
        rootc=build_abc(rootc,sir1[i]);
    ldr(rootc);
    printf("\n");

    return 0;
}
// Creare unui nod nou in arbore
struct bnode* new_tree_node(int a)
{
    struct bnode* p;

    p= (struct bnode*) malloc(sizeof(struct bnode));
    p->ora=a1;
    p->min=a2;
    p->sec=a3;
    p->left=NULL;
    p->right=NULL;
}
//Construim arborele
// A = {a1, a2, ...., aN} -> multimea de elemente cu care se va construit
// a1 -> radacina
 urmatoarele nl = N/2 elemente (a2, a3, ... ak) -> vor constitui subarborele sting
 urmatoarele nr = N - N/2 -1 elemente (ak, ak+1, ... aN) -> vor constitui subarborele drept
-> se efectueaza recursiv
se iese din recursivitate -> cand nl si/sau nr devin 0


struct bnode* build_abe(int n, int A[])
{
struct bnode* p;
```

```
static int i=0;
int nl, nr;

if (n == 0) return NULL;
else
{
    nl=n/2;
    nr=n-nl-1;
    p = new_tree_node(A[i]);
    i++;
    p->left = build_abe(nl,A);
    p->right = build_abe(nr,A);
    return p;
}
}
//Inserarea in arbori binari
struct bnode* build_abc(struct bnode*r, int a)
{
    if (r==NULL) r= new_tree_node(a);
    else
    {
        if (a < r->data ) r->left=build_abc(r->left,a);
        if (a > r->data ) r->right=build_abc(r->right,a);
    }

    return r;}
//Cautarea in arbori binari
struct bnode* search_abc(struct bnode*r, int a)
{
  if (r == NULL)  return NULL;
  if (r->data == a) return r;
  if (a < r->data) return (search_abc(r->left,a));
```

```
 if (a > r->data) return (search_abc(r->right,a));
}
//Algoritmii de parcurgere -> recursivi
void ldr(struct bnode* r)
{
  if(r!=NULL)
  {
    ldr(r->left);
    printf("%d, ", r->data);
    ldr(r->right);
  } }
void dlr(struct bnode* r)
{
  if(r!=NULL)
  {
    printf("%d, ", r->data);
    dlr(r->left);
    dlr(r->right);
  } }
void lrd(struct bnode* r)
{
  if(r!=NULL)
  {
    lrd(r->left);
    lrd(r->right);
    printf("%d, ", r->data);  }}
```