

Stuparu Elena Natalia, grupa 412D

Lucrarea 1 SDA (9.04.2021)

Problema 1)

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

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

```
#define SIZE    100
```

```
typedef int T;
```

```
typedef struct ListNode {
```

```
    T value;
```

```
    struct ListNode* next;
```

```
    struct ListNode* prev;
```

```
} ListNode;
```

```
typedef struct List {
```

```
    ListNode* first;
```

```
    ListNode* last;
```

```
} List;
```

```
ListNode *createListNode(T value);
```

```
List createList(T value);
```

```
List nill(void);
```

```
int isEmpty(List list);
```

```
List enqueue(List list, T value);
```

```
int contains(List list, T value);
```

```
List dequeue(List list);
```

```
int top(List list);
```

```
int length(List list);
```

```
List destroyList(List list);
```

```
void print(List list);
```

```
int main() {
```

```
    List l = nill();
```

```

l = createList(3);
l = enqueue(l, 4);
l = enqueue(l, 5);
l = enqueue(l, 6);
print(l);
printf("Va iesi un elementul %d din coada\n", top(l));
print(l);
dequeue(l);
if(contains(l, 4))
    printf("coada contine elementul 4\n");
destroyList(l);
return 0;
}

```

```

ListNode *createListNode(T value) {
    ListNode node = (ListNode) malloc(sizeof(ListNode));
    node->value = value;
    node->prev = NULL;
    node->next = NULL;
    return node;
}

```

```

List nill(void) {
    List list;
    list.first = list.last = NULL;
    return list;
}

```

```

List createList(T value) {
    List list;
    list = nill();
    list.first = createListNode(value);
    list.last = list.first;
    return list;
}

```

```

}

int isEmpty(List list) {
    if(list.first == NULL && list.last == NULL)
        return 1;
    else
        return 0;
}

List enqueue(List list, T value) {
    if(list.first == NULL){
        list = createList(value);
        return list;
    }
    else{
        ListNode *temp;
        ListNode *p = createListNode(value);
        temp = list.last;
        list.last = p;
        temp->next = p;
        p->prev = temp;
        //list.first = p;
        return list;
    }
}

List dequeue(List list) {
    if(isEmpty(list))
        return list;
    else{
        ListNode *temp = list.first;
        list.first->next->prev = NULL;
        list.first = list.first->next;
        free(temp);
    }
}

```

```

        return list;
    }
}

int top(List list) {
    if(isEmpty(list))
        return -1;
    else
        return list.first->value;
}

int contains(List list, T value) {
    int valid = 0;
    if(list.first == NULL)
        return 0;
    else{
        ListNode *temp = list.first;
        while(temp!= list.last){
            if(temp->value == value)
                valid = 1;
            temp = temp->next;
        }
        if(temp == list.last && temp->value == value)
            valid = 1;
        if(valid)
            return 1;
        else
            return 0;
    }
}

void print(List list) {
    ListNode *temp;
    if(!isEmpty(list)){

```

```
temp = list.first;
while(temp != NULL) {
    printf("%d ", temp->value);
    temp = temp->next;
}
printf("\n");
}
```

```
List destroyList(List list) {
    ListNode *temp, *aux;
    temp = list.first;
    while(temp != list.last){
        aux = temp;
        temp = temp->next;
        list.first = temp;
        temp->prev = NULL;
        free(aux);
    }
    list.first = NULL;
    list.last = NULL;
    free(temp);
    return list;
}
```

```

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

#define SIZE 100

typedef int T;
typedef struct ListNode {
    T value;
    struct ListNode* next;
    struct ListNode* prev;
} ListNode;

typedef struct List {
    ListNode* first;
    ListNode* last;
} List;

ListNode *createListNode(T value);
List createList(T value);
List nill(void);
int isEmpty(List list);
List enqueue(List list, T value);
int contains(List list, T value);
List dequeue(List list);
int top(List list);
int length(List list);
List destroyList(List list);
void print(List list);

int main() {
    List l = nill();
    l = createList( value: 3);
    l = enqueue(l, value: 4);
    l = enqueue(l, value: 5);
    l = enqueue(l, value: 6);
    print(l);
    printf( "Format: "Va iesi un elementul %d din coada\n", top(l));
    print(l);
    dequeue(l);
    if(contains(l, value: 4))
        printf( "Format: este elementul %d din coada\n", 4);
}

```

```

ListNode *createListNode(T value) {
    ListNode *node = (ListNode*) malloc(sizeof(ListNode));
    node->value = value;
    node->prev = NULL;
    node->next = NULL;
    return node;
}

List nill(void) {
    List list;
    list.first = list.last = NULL;
    return list;
}

List createList(T value) {
    List list;
    list = nill();
    list.first = createListNode(value);
    list.last = list.first;
    return list;
}

int isEmpty(List list) {
    if(list.first == NULL && list.last == NULL)
        return 1;
    else
        return 0;
}

List enqueue(List list, T value) {
    if(list.first == NULL){
        list = createList(value);
        return list;
    }
    else{
        ListNode *temp;
        ListNode *p = createListNode(value);
        temp = list.last;
        temp->next = p;
        p->prev = temp;
        list.last = p;
    }
}

```

```

List enqueue(List list, T value) {
    if(list.first == NULL){
        list = createList(value);
        return list;
    }
    else{
        ListNode *temp;
        ListNode *p = createListNode(value);
        temp = list.last;
        list.last = p;
        temp->next = p;
        p->prev = temp;
        //list.first = p;
        return list;
    }
}

List dequeue(List list) {
    if(isEmpty(list))
        return list;
    else{
        ListNode *temp = list.first;
        list.first->next->prev = NULL;
        list.first = list.first->next;
        free(temp);
        return list;
    }
}

int top(List list) {
    if(isEmpty(list))
        return -1;
    else
        return list.first->value;
}

```

```

int contains(List list, T value) {
    int valid = 0;
    if(list.first == NULL)
        return 0;
    else{
        ListNode *temp = list.first;
        while(temp != list.last){
            if(temp->value == value)
                valid = 1;
            temp = temp->next;
        }
        if(temp == list.last && temp->value == value)
            valid = 1;
        if(valid)
            return 1;
        else
            return 0;
    }
}

void print(List list) {
    ListNode *temp;
    if(!isEmpty(list)){
        temp = list.first;
        while(temp != NULL) {
            printf(_Format: "%d ", temp->value);
            temp = temp->next;
        }
        printf(_Format: "\n");
    }
}

List destroyList(List list) {
    ListNode *temp, *aux;
    temp = list.first;
    while(temp != list.last){

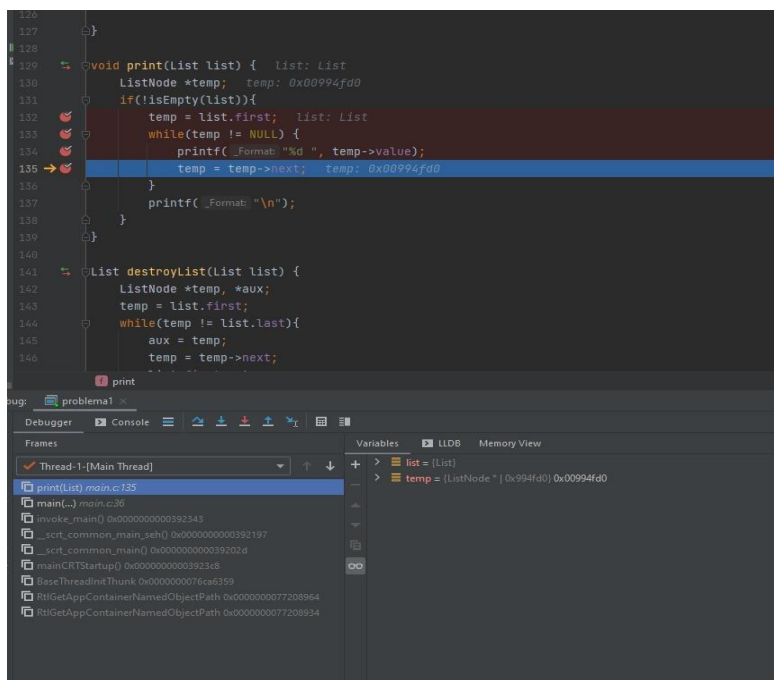
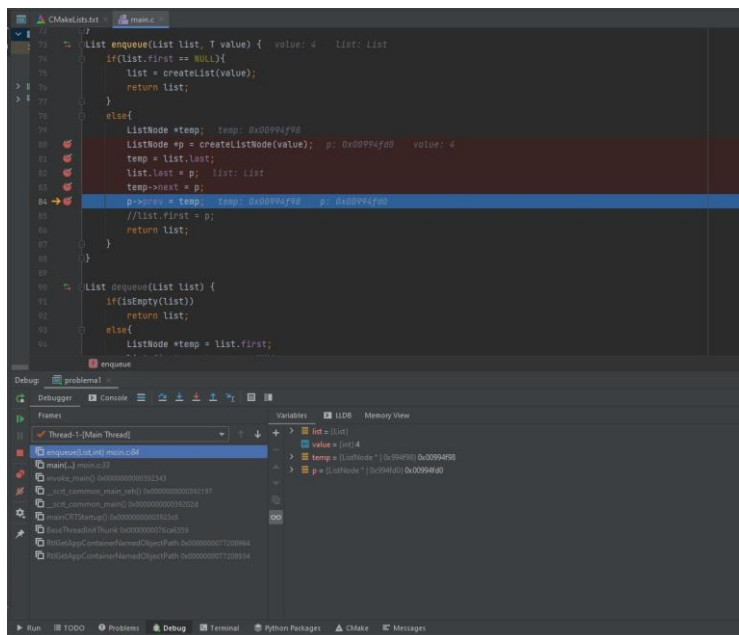
```

```

List destroyList(List list) {
    ListNode *temp, *aux;
    temp = list.first;
    while(temp != list.last){
        aux = temp;
        temp = temp->next;
        list.first = temp;
        temp->prev = NULL;
        free(aux);
    }
    list.first = NULL;
    list.last = NULL;
    free(temp);
    return list;
}

```

Listare functii:




```

87 }
88     return list;
89 }
90
91 List dequeue(List list) { List: List
92     if(isEmpty(list))
93         return list;
94     else{
95         ListNode *temp = list.first; temp: 0x012664e8
96         list.first->next->prev = NULL;
97         list.first = list.first->next; list: List
98         free(temp); temp: 0x012664e8
99     }
100     return list;
101 }
102
103 int top(List list) {
104     if(isEmpty(list))
105         return -1;
106     else
107         return list.first->value;
108 }
109
110 int contains(List list, T value) {
111     dequeue

```

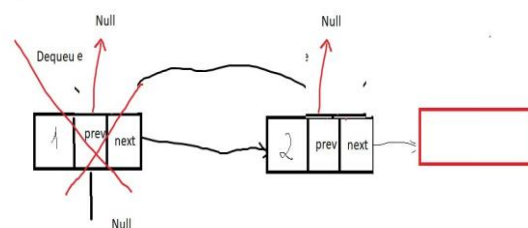
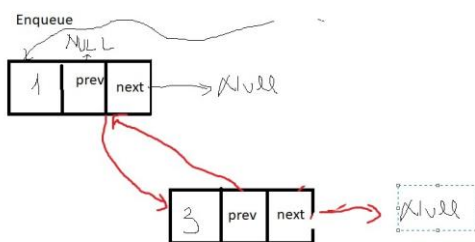
Debugger: problema1

Frames: Thread-1 (Main Thread)

- dequeue(List) main.c:98
- main(...) main.c:39
- invoke_main() 0x00000000000212343
- _start_common_main_seh() 0x00000000000212197
- _start_common_main() 0x0000000000021202d
- mainCRTStartup() 0x000000000002123c8
- BaseThreadInitThunk 0x000000000076ca639
- RtlGetAppContainerNamedObjectPath 0x000000000077208964
- RtlGetAppContainerNamedObjectPath 0x000000000077208934

Variables: LLDB Memory View

- list = [List]
- temp = (ListNode * | 0x12664e8) 0x012664e8



In momentul crearii liste vom pune primul element in lista (createListNode) si vom atribui valoare null pentru urmatorul si predecesorul. Primul si ultimul element sunt nule. In functia de adugare in coada (enqueue) vom intrebati daca lista e goala. In acest caz, return o noua lista cu elementul pe primul loc; Altfel vom folosi variabila temp pentru a ne ajuta sa tinem minte pointerul spre ultimul element,. Apoi adaugam in coada pe ultima pozitie elementul dorit si refacem legaturile cu variabila temp. Pentru functia top aceasta doar intoarce primul element din coada. In cadrul functiei contains am folosit o parcurgere [prin lista pana la elementul dorit si in cazul in care l-am gasit (temp->value == value) ne oprim si vom intoarce 1. De asemenea, si functia de listare (print) are o parcurgere prin lista folosind variabila temp si afisind fiecare element in parte.

Problema 2)

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

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

```
struct bnode {
```

```
    int ora;
```

```
    int min;
```

```

    int sec;

    struct bnode* left;

    struct bnode* right;
};

struct bnode* new_tree_node(int ora, int min, int sec);

int compare(struct bnode*r, int ora, int min, int sec);

struct bnode* build_abe(int n, int ora[], int min[], int sec[]);


struct bnode* build_abc(struct bnode*r, int ora, int min, int sec);

struct bnode* search_abc(struct bnode*r, int ora, int min, int sec);


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

int sir[10]={1,2,3,4,5,6,7,8,9,10};
int sir1[10]={3,5,10,7,9,8,2,6,1,4};
int sir2[10]={3,4,10,10,9,8,3,6,5,4};

int main() {
    int i;

    struct bnode* roote = NULL;

    struct bnode* rootc = NULL;


    roote=build_abe(10,sir1, sir, sir2);

    for (i=0; i < 10; i++)

        rootc=build_abc(rootc,sir1[i], sir[i], sir2[i]);

    ldr(rootc);

    printf("\n");

    return 0;
}

struct bnode* new_tree_node(int ora, int min, int sec)
{

```

```

    struct bnode* p;
    p= (struct bnode*) malloc(sizeof(struct bnode));
    p->min=min;
    p->ora = ora;
    p->sec = sec;
    p->left=NULL;
    p->right=NULL;
}

struct bnode* build_abe(int n, int ora[], int min[], int sec[])
{
    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(ora[i], min[i], sec[i]);
        i++;
        p->left = build_abe(nl, ora, min, sec);
        p->right = build_abe(nr, ora, min, sec);
        return p;
    }
}

struct bnode* build_abc(struct bnode*r, int ora, int min, int sec)
{
    if (r==NULL) r= new_tree_node(ora, min, sec);
    else
    {
        if (compare(r, ora, min, sec) < 0) r->left=build_abc(r->left,ora, min, sec);
    }
}

```

```

        if (compare(r, ora, min, sec) < 0) r->right=build_abc(r->right,ora, min, sec);
    }
    return r;
}

int compare(struct bnode*r, int ora, int min, int sec) {
    printf("%d %d %d", r->ora, r->min, r->sec);
    if (r->min < min && r->ora < ora && r->sec < sec)
        return 1;
    else
        if (r->min < min && r->ora == ora && r->sec < sec)
            return 1;
        else
            if (r->min == min && r->ora == ora && r->sec < sec)
                return 1;
            else
                return -1;
}

struct bnode* search_abc(struct bnode*r, int ora, int min, int sec)
{
    if (r == NULL) return NULL;
    if (r->min == min && r->ora == ora && r->sec == sec) return r;
    if (compare(r, ora, min, sec) < 0) return(search_abc(r->left, ora, min, sec));
    if (compare(r, ora, min, sec) > 0) return (search_abc(r->right, ora, min, sec));
}

void ldr(struct bnode* r)
{
    if(r!=NULL)
    {
        ldr(r->left);
        printf("%d %d %d, ", r->ora, r->min, r->sec);
        ldr(r->right);
    }
}

```

```

    }
}
void dlr(struct bnode* r)
{
    if(r!=NULL)
    {
        printf("%d %d %d, ", r->ora, r->min, r->sec);
        dlr(r->left);
        dlr(r->right);
    }
}
void lrd(struct bnode* r)
{
    if(r!=NULL)
    {
        lrd(r->left);
        lrd(r->right);
        printf("%d %d %d, ", r->ora, r->min, r->sec);
    }
}

```

```

57
58
59 struct bnode* build_abe(int n, int ora[], int min[], int sec[])  n: 10   ora: 0x0047a070   min: 0x0047a048   sec: 0x0047a098
60 {
61     struct bnode* p;  p: 0xffffffff
62     static int i=0;  i: 0
63     int nl, nr;  nl: 5   nr: 4
64
65     if (n == 0) return NULL;
66     else
67     {
68         nl=n/2;
69         nr=n-nl-1;  n: 10   nl: 5   nr: 4
70         p = new_tree_node(ora[i], min[i], sec[i]);  sec: 0x0047a098   ora: 0x0047a070   p: 0xffffffff   i: 0
71         i++;
72         p->left = build_abe(nl, ora, min, sec);
73         p->right = build_abe(nr, ora, min, sec);
74         return p;
75     }
76 }
77
78 struct bnode* build_abe(struct bnode* p, int n, int min, int sec)
79 build_abe

```

Console

```

Thread]
*int*int*main.c:70
57
0x0000000000471d73
main_seh0 0x0000000000471a67
main0 0x000000000047195d
0 0x0000000000471d48
rank 0x0000000076a6359
UserNamedObjectPath 0x0000000077208964
UserNamedObjectPath 0x0000000077208934

```

Variables

```

n = (int) 10
ora = (int *) [0x47a070] 0x0047a070
min = (int *) [0x47a048] 0x0047a048
sec = (int *) [0x47a098] 0x0047a098
nl = (int) 5
i = (int) 0
p = [bnode *] 0xffffffff 0xffffffff
nr = (int) 4

```

```
76     }  
77  
78     struct bnode* build_abc(struct bnode*r, int ora, int min, int sec)   ora: 5    min: 2    r: 0x010ac488    sec: 4  
79     {  
80         if (r==NULL) r= new_tree_node(ora, min, sec);  
81         else  
82         {  
83             if (compare(r, ora, min, sec) < 0) r->left=build_abc(r->left,ora, min, sec);   ora: 5    r: 0x010ac488    sec: 4  
84             if (compare(r, ora, min, sec) < 0) r->right=build_abc(r->right,ora, min, sec);  
85         }  
86  
87         return r;  
88     }  
89     int compare(struct bnode*r, int ora, int min, int sec) {  
90         if (r->min < min && r->ora < ora && r->sec < sec)  
91             return -1;  
92         else  
93             if (r->min < min && r->ora == ora && r->sec < sec)  
94                 return -1;  
95         return 1;  
96     }  
97  
98     void build_abc
```

Console

[hread]

int,int,int) main.c:63

0000000000471c73
main_seh! 0x0000000000471ac7
main! 0x000000000047195d
winapi!0x0000000000000000

Variables LLDB Memory View

+> x = (bnode *) 0x010ac488
- ora = (int) 5
- min = (int) 2
- sec = (int) 4

The screenshot shows a C program with a `compare` function and a `search_abc` function. The `compare` function takes a `bnode*` and three integers (`ora`, `min`, `sec`) and returns `-1`, `0`, or `1` based on comparisons. The `search_abc` function uses `compare` to find a node in a linked list.

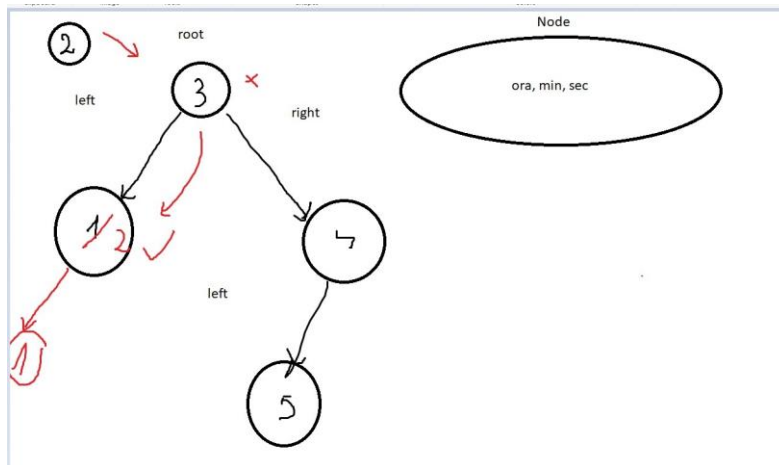
```

1  compare(struct bnode*r, int ora, int min, int sec) {
2      printf("Format: %d %d %d", r->ora, r->min, r->sec);
3      if (r->min < min && r->ora < ora && r->sec < sec)
4          return -1;
5      else
6          if (r->min < min && r->ora == ora && r->sec < sec)
7              return -1;
8          else
9              if (r->min == min && r->ora == ora && r->sec < sec)
10                 return -1;
11             else
12                 return 1;
13      }
14
15  struct bnode* search_abc(struct bnode*r, int ora, int min, int sec)
16  {
17      compare
18  }

```

The LLDB Variables window shows the following values:

| Variables | LLDB | Memory View |
|-------------------------|-----------|-------------|
| r = [bnode *] 0x0fe95e8 | 0x0fe95e8 | 0x0fe95e8 |
| ora = (int) 10 | 10 | |
| min = (int) 3 | 3 | |
| sec = (int) 10 | 10 | |



Fiecare nod este descris prin ora, min, sec. Dacă dorim să adăugăm elementul nodul 2 (notatie specială), vom începe de la root, vom compara valorile din root cu cele din nodul 2 folosind funcția compare. În momentul acesta funcția compare va întoarce -1 și ne îndreptăm spre left. Ajungem în nodul 1, iar răspunsul funcției compare va fi 1. Atunci vom adăuga aici nodul 2 și vom pune nodul 1 ca fiind copilul stâng al acestuia. Pe același principiu este construită și funcția search_abc așa cum am descris în diagrama. Funcția de compare va returna 1 dacă rootul este mai mic și astfel se va merge pe partea dreapta. Altfel va întoarce -1.