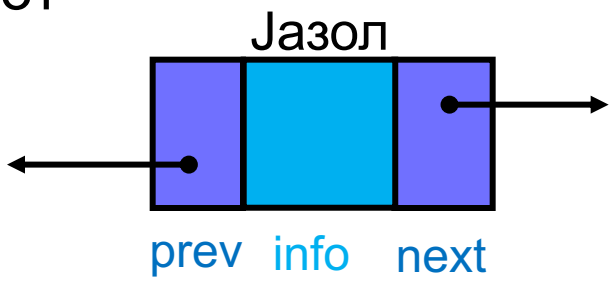
ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Двојно поврзани листи – динамички структури

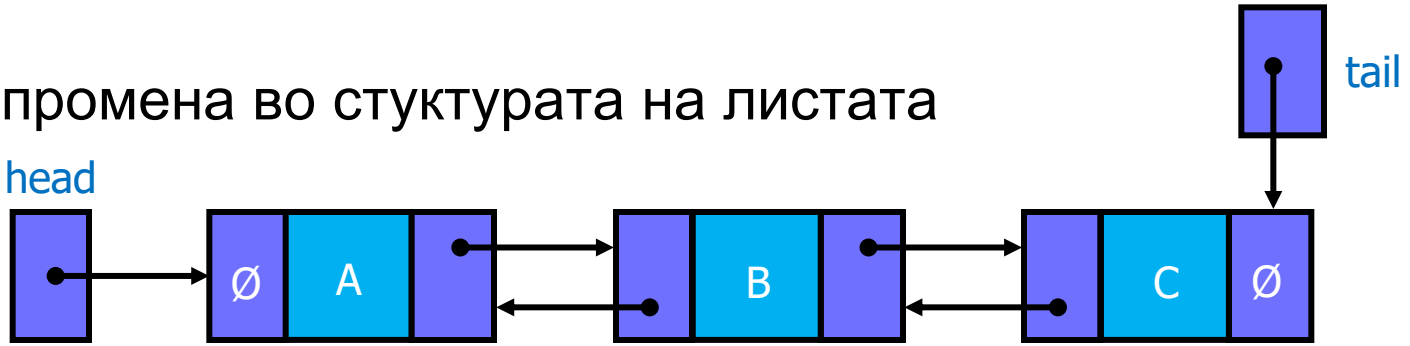
– Податочни структури и програмирање –

Двојно поврзани листи

- Има промена во структурата на јазолот
- Секој јазол содржи
  - Податок (info)
  - Покажувач кон претходниот јазол од листата (prev)
  - Покажувач кон следниот јазол од листата (next)



- Има промена во стуктурата на листата



- Може да **се итерира во двете насоки**

Декларирање

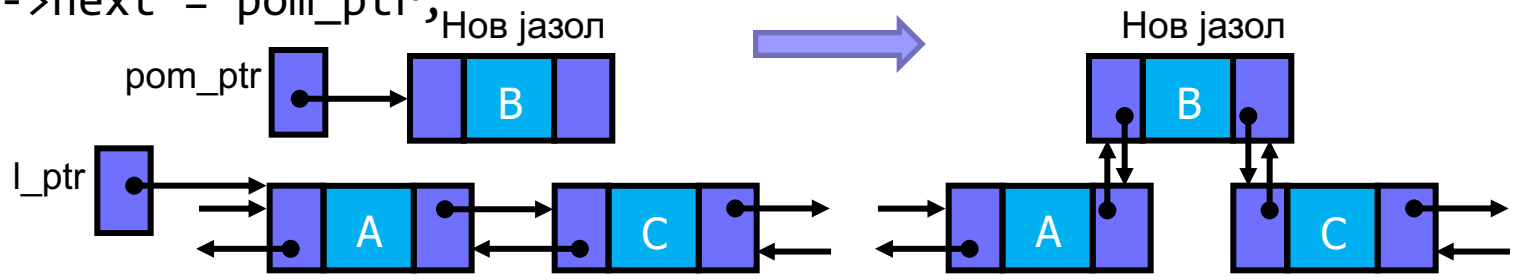
```
#include <iostream>

using namespace std;

typedef int info_t;
struct node
{
        info_t info;
        node *next;
        node *prev;
};
```

```
struct DLList {
        node *head;
        node *tail;
        void init();
        void ins_after(node *l_ptr,info_t x);
        void ins_before(node *l_ptr,info_t x);
        void ins_first(info_t x);
        node *node_on_position(int n);
        void mk_link_list(int n);
        void del_first();
        void del_last();
        node *find_first(info_t x);
        node *find_last(info_t x);
        void printlist();
        void free_link_list();
        void ins_in_order(info_t x);
};
void DLList::init(){
    head = tail = NULL;
}
```

Додавање на јазол после даден јазол

- **Вметнување на јазол во средина** на листа **веднаш после јазолот на кој покажува** `l_ptr`
- **Потребно е да се ажурираат 4 линка**

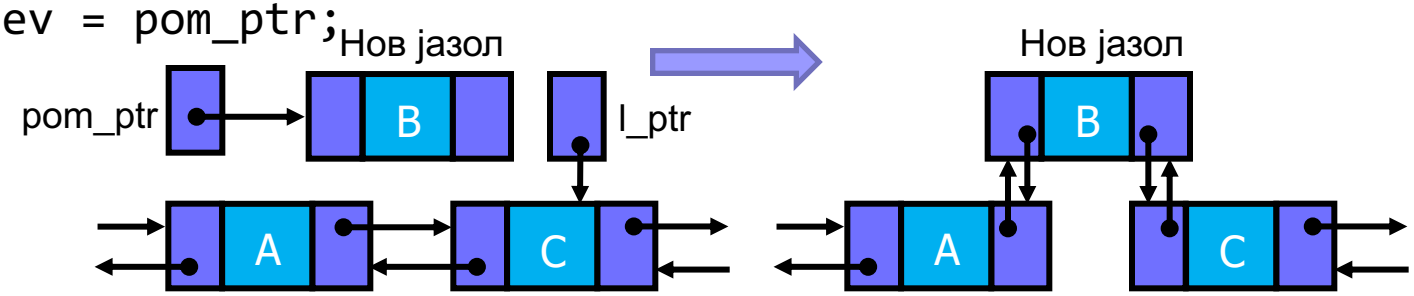
```
void DLList::ins_after(node *l_ptr, info_t x){
    node *pom_ptr = new node;
    pom_ptr->info = x;
    pom_ptr->prev = l_ptr;
    pom_ptr->next = l_ptr->next;
    if(l_ptr->next == NULL)
        tail = pom_ptr;
    else
        l_ptr->next->prev = pom_ptr;
    l_ptr->next = pom_ptr;
}
```



Додавање на јазол пред даден јазол

- **Вметнување на јазол во средина** на листа **веднаш пред јазолот на кој покажува** `l_ptr`
- **Потребно е да се ажурираат 4 линка**

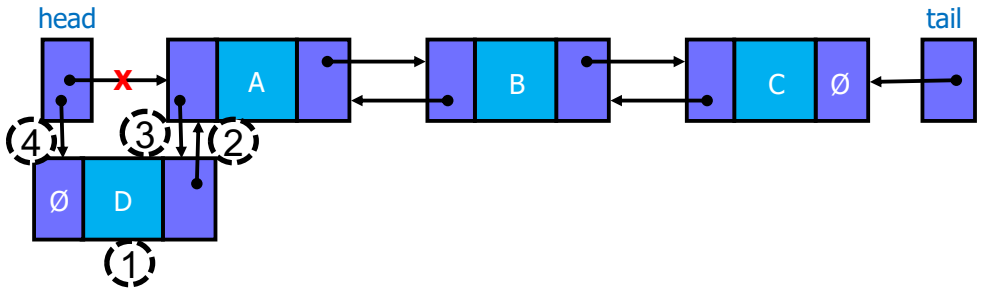
```
void DLList::ins_before(node *l_ptr, info_t x){
    node *pom_ptr = new node;
    pom_ptr->info = x;
    pom_ptr->next = l_ptr;
    pom_ptr->prev = l_ptr->prev;
    if(l_ptr->prev == NULL)
        head = pom_ptr;
    else
        l_ptr->prev->next = pom_ptr;
    l_ptr->prev = pom_ptr;
}
```



Додавање на јазол на почеток од листа

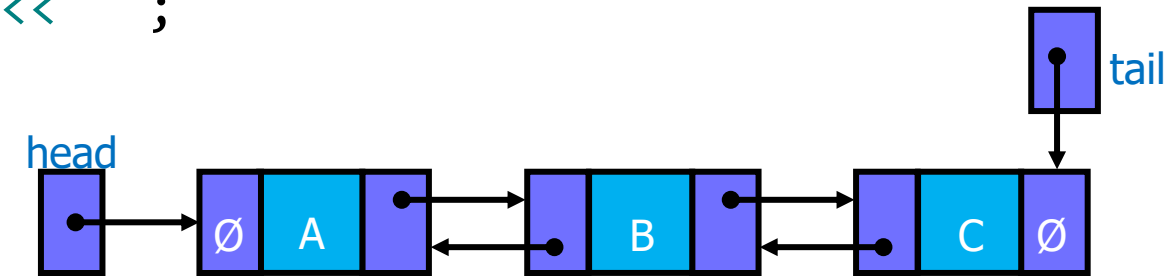
```
void DLList::ins_first(info_t data){
    node *pom_ptr = new node;
    pom_ptr->info = data;
    pom_ptr->prev = NULL;
    pom_ptr->next = head;
    if(head == NULL)
        tail = pom_ptr;
    else
        head->prev = pom_ptr;
    head = pom_ptr;
}
```

```
//Повик во main():


DLList L1;
L1.init();
L1.ins_first(5);
L1.ins_first(7);
L1.ins_first(3);
```



Печатење на двојно поврзана листа

```
void DLList::printlist()
{
        void printnode(node *p);
        node *p;
        for (p = head; p != NULL; p = p->next)
                printnode(p);
        cout<<endl;
}

void printnode(node *p)
{
        cout<<p->info<<" ";
}
```



Враќање на адреса на јазол на позиција во листата

```
node* DLList::node_on_position(int n) {
        node* pom_ptr = head;
        for (int i = 0; i < n && pom_ptr != NULL; i++)
                pom_ptr = pom_ptr->next;

        return pom_ptr;
}
```

```
//Повик во main():
DLList D;
D.init();
D.ins_first(10);D.ins_first(9);
D.ins_first(8);D.ins_first(6);
D.printlist();
node* pom = D.node_on_position(2);
D.ins_before(pom, 4);
D.printlist();
D.ins_after(pom, 12);
D.printlist();
```

Што ќе се испечати?

6 8 9 10
6 8 4 9 10
6 8 4 9 12 10

Внесување на двојно поврзана листа

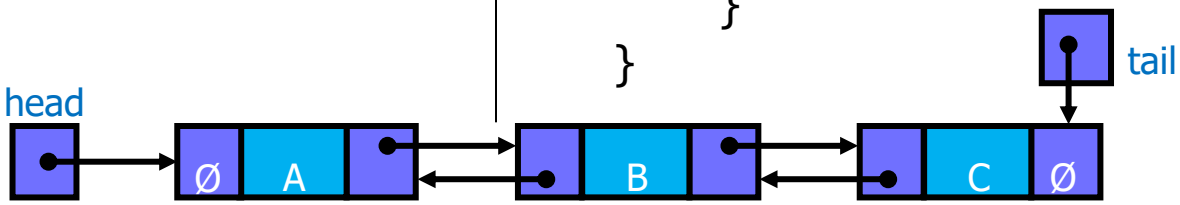
- `int n` – број на елементи во листата што се креира

```
void DLList::mk_link_list(int n)
{
    info_t data;
    init();

    while (n--)
    {
        cin>>data;
        ins_first(data);
    }
}
```

Бришење на прв и последен јазол од листата

```
void DLList::del_first(){
    if (head != NULL){
        if (head->next == NULL)
        {
            delete head;
            head = tail = NULL;
        }
        else
        {
            node *pom = head;
            head = head->next;
            head->prev = NULL;
            delete pom;
        }
    }
}
```

```
void DLList::del_last(){
    if (head != NULL) {
        if (head->next == NULL)
        {
            delete head;
            head = tail = NULL;
        }
        else
        {
            node *pom = tail;
            tail = tail->prev;
            tail->next = NULL;
            delete pom;
        }
    }
}
```



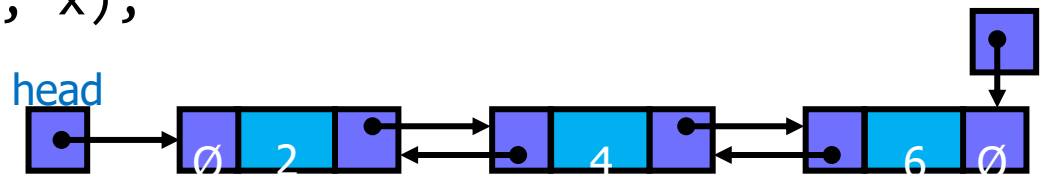
Наоѓање на елемент во листата

- Враќа **покажувач кон јазолот во кој е пронајдена вредноста** или **NULL** ако не е пронајдена

```
node *DLList::find_first(info_t x)
{
        node *l;
        for (l = head; l != NULL && l->info != x; l = l->next);
        return (l != NULL ? l : NULL);
}

node *DLList::find_last(info_t x)
{
        node *l;
        for (l = tail; l != NULL && l->info != x; l = l->prev);
        return (l != NULL ? l : NULL);
}
```

Вметнување елемент во веќе подредена листа на соодветно место

```
void DLList::ins_in_order(info_t x)
{
    node *p = head;

    if (x <= p->info) /* vmetni na chelo */
        ins_first(x);
    else /* najdi mu go mestoto */
    {
        while (p != NULL && p->info < x)
            p = p->next;
        if(p == NULL)
            ins_after(tail, x);
        else
            ins_before(p, x);
    }
}
```



Бришење на двојно поврзана листа

```
void DLList::free_link_list()
{
    while (head != NULL)
        del_first();
}
```

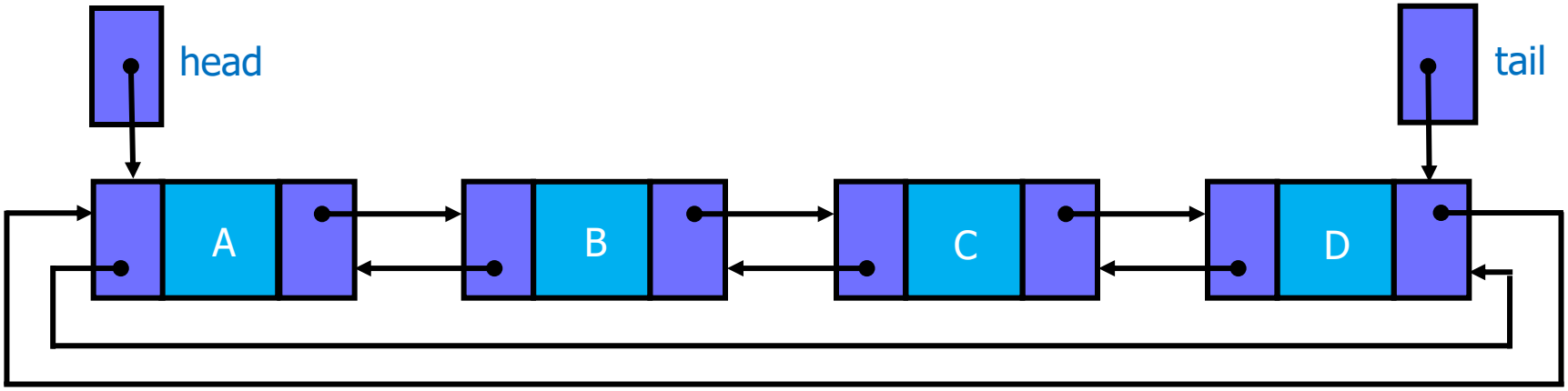
```
int main(){
    DLList D;
    D.init();
    D.ins_first(10); D.ins_first(9);
    D.ins_first(8); D.ins_first(6);
    D.printlist();
    D.ins_in_order(7); D.printlist();
    D.del_first(); D.del_last();
    D.printlist();
    D.free_link_list();
}
```

Што ќе се испечати?

```
6 8 9 10
6 7 8 9 10
7 8 9
```

Двојно поврзани кружни листи

- Листата **нема крај (NULL)** (завртена е во јамка)
- Може да **се итерира во двете насоки**
- **Структурата на јазолот останува иста како кај двојната поврзана листа**
- Има **„два краја“** (подеднакво лесно може да се додаваат и бришат елементи и од почетокот и од крајот на листата) но мора да се внимава дека крајот има линк поле кое води до почениот јазол



Декларирање и креирање

```
#include <iostream>

using namespace std;

typedef int info_t;
struct node
{
        info_t info;
        node *next;
        node *prev;
};
```

```
struct CDLList {
    node *head;
    node *tail;
    void init();
    void ins_first(info_t x);
    void ins_last(info_t x);
    void mk_link_list(int n);
    void del_first();
    void del_last();
    node *find_first(info_t x);
    void printlist();
    void free_link_list();
};
void CDLList::init(){
     head = tail = NULL;
}
```

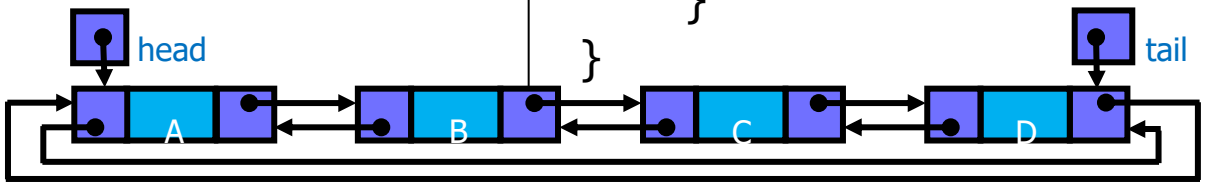
Печатење на двојно поврзана кружна листа

```
void CDLList::printlist(){
        void printnode(node *p);
        node *p;
        for (p = head; p != tail; p = p->next) {
                printnode(p);
                cout << "<->";
        }
         printnode(p);
        cout<<endl;
}

void printnode(node *p)
{
        cout<<p->info;
}
```

Додавање на јазол на почеток и на крај од двојно поврзана кружна листа

```
void CDLList::ins_first(info_t
data){
    node *p_ptr = new node;
    p_ptr->info = data;
    if(head == NULL){
        tail = head = p_ptr;
        p_ptr->prev =
          p_ptr->next = p_ptr;
    }else{
        p_ptr->next = head;
        p_ptr->prev = tail;
        tail->next = p_ptr;
        head->prev = p_ptr;
        head = p_ptr;
    }
}
```

```
void CDLList::ins_last(info_t
        data){
    node* p_ptr = new node;
    p_ptr->info = data;
    if (head == NULL) {
        tail = head = p_ptr;
        p_ptr->prev =
          p_ptr->next = p_ptr;
    }else {
        p_ptr->prev = tail;
        p_ptr->next = head;
        tail->next = p_ptr;
        head->prev = p_ptr;
        tail = p_ptr;
    }
}
```



Креирање на двојно поврзана кружна листа

```
void CDLList::mk_link_list(int n)
{
    info_t data;
    init();

    while (n--)
    {
        cin>>data;
        ins_first(data);
    }
}
```

Што ќе се испечати за внес
10 5 6 9 1

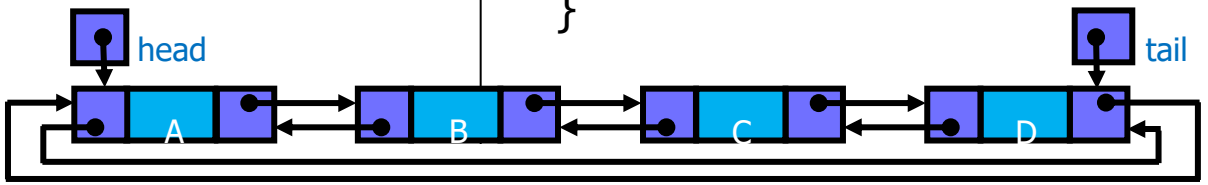
```
//Повик во main():

CDLList D;
D.mk_link_list(5);
D.printlist();
```

```
1<->9<->6<->5<->10
```

Бришење на прв и последен јазол од двојно поврзана кружна листа

```
void CDLList::del_last(){
    if (head != NULL){
        if (head == tail){
            delete head;
            head = tail = NULL;
        }
        else{
            node *pom = tail;
            tail = tail->prev;
            tail->next = head;
            head->prev = tail;

            delete pom;
        }
    }
}
```

```
void CDLList::del_first() {
    if (head != NULL){
        if (head->next == head){
            delete head;
            head = tail = NULL;
        }
        else{
            node *pom = head;
            tail->next = head->next;
            head = head->next;
            head->prev = tail;

            delete pom;
        }
    }
}
```



Наоѓање на елемент во двојно поврзана кружна листа и бришење на истата

- При пронаоѓање на елементот функцијата враќа **покажувач кон јазолот во кој е пронајдена вредноста** или **NULL** ако не е пронајдена

```
node *CDLList::find_first(info_t x)
{
        node *l;
        for (l = head; l != tail && l->info != x; l = l->next);
        if(l==tail && l->info !=x)
                    l = NULL;
        return (l != NULL ? l : NULL);
}
void CDLList::free_link_list()
{
    while (head != NULL)
        del_first();
}
```

Употреба на двојно поврзана кружна листа

```
int main(){
    CDLList D;
    D.init();
    D.ins_first(10); D.ins_last(9);
    D.ins_last(8); D.ins_first(6);
    D.printlist();

    D.ins_last(11);
    D.printlist();

    D.del_first();
    D.del_last();
    D.printlist();
    D.free_link_list();
    return 0;
}
```

Што ќе се испечати?

```
6<->10<->9<->8
6<->10<->9<->8<->11
10<->9<->8
```