ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

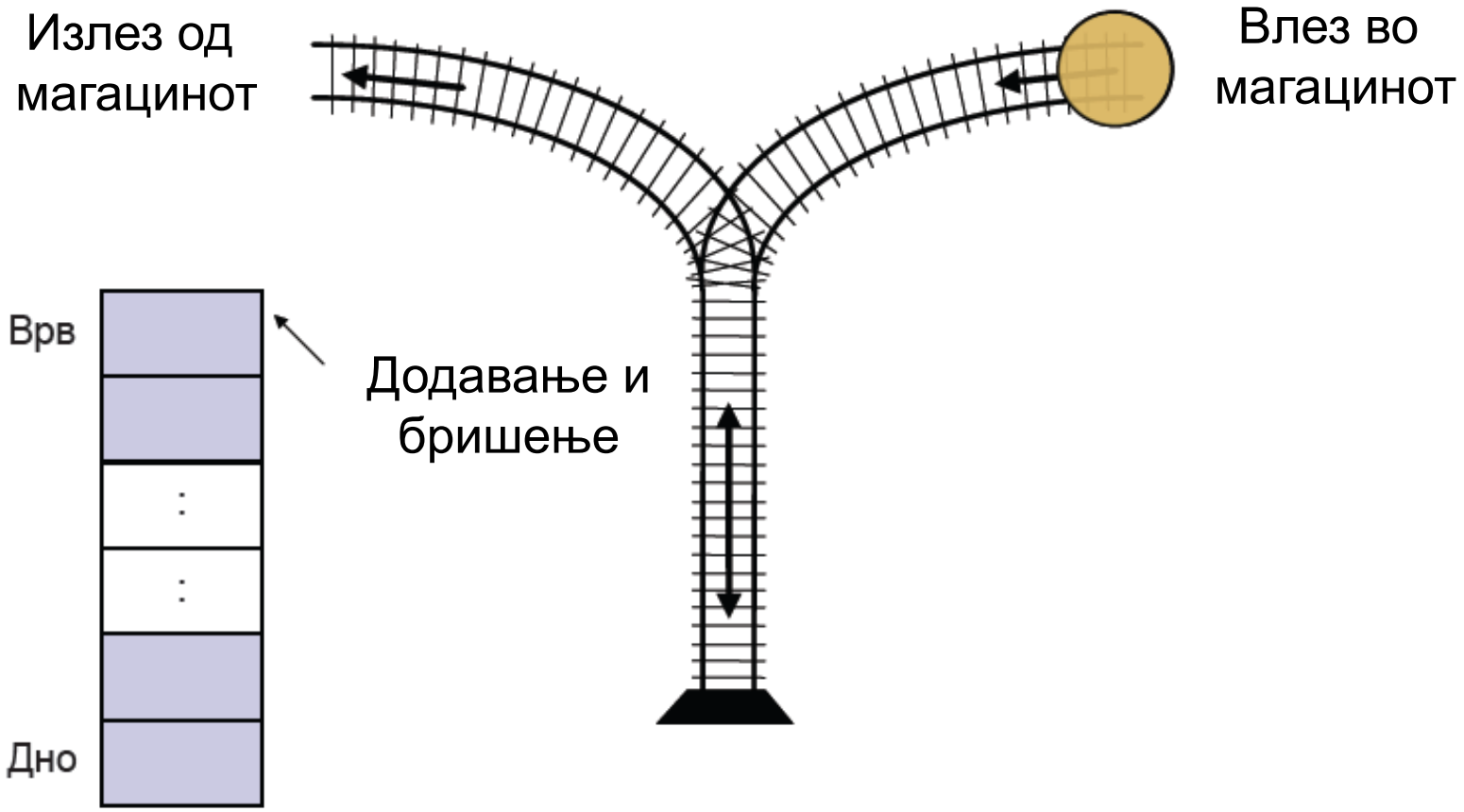
Магацини (stacks) и редови (queues)

– Податочни структури и програмирање –

Магацин (Stack) – Стек

- Дефиниција: **Хомогена подредена линеарна структура** со еден крај
- **LIFO (Last-In-First-Out) структура** – „куп“ на чиј што врв се додава и од чиј што врв се вади (не е дозволено вметнување или извлекување од средината или од дното)
- Секој **нов елемент** за магацинот се додава **на неговиот врв**
- **Секој елемент** што се отстранува од магацинот **се вади од неговиот врв**
- => **Последица**: елементите се вадат од магацинот во обратен редослед од оној во кој биле ставени на него

Магацин (стек)



Примена

- Во некои програмски јазици (или стандардни библиотеки) е имплементиран како **вграден тип**
- Голем број алгоритми **интерно го користат** за да ја извршат својата задача
- Различни **форми на вгнездување (загради)**
- Пресметување на **аритметички и други изрази**
- Имплементација на **функциски повици**
- Водење **евиденција за претходните избори** (backtracking)
- Водење **евиденција за следните избори** (креирање на лавиринт)

Функции подржани од структурата магацин

- **Основни**:
  - Додавање на елемент на магацинот
  - Вадење на елемент од магацинот
  - Проверка: дали магацинот е празен?
- **Дополнителни**:
  - Читање на елементот кој се наоѓа на врвот (без негово отстранување)
  - Проверка: дали магацинот е полн? (имплементациско ограничување)

Реализација на магацин како еднодимензионална низа

- **Максималната длабочина** на магацинот (однапред се задава) **STACKSIZE** – во магацинот ќе може да се сместат најмногу STACKSIZE елементи

- За чување на елементите се користи **еднодимензионална низа** и **дополнителна променлива** (целобројна или покажувач) која го означува **врвот на магацинот**

Функција 1: Додавање на елемент на магацин

- **Псевдокод**:

$$S \Leftarrow x\text{: if } v \geq M\text{-}1 \text{ then overflow (полн стек)}$$
$$\text{else } v \leftarrow v+1,\ S_v \leftarrow x$$

S – the stack
x – the element to be added
v – the top of the Stack
M – the maximum sze of the stack

Функција 2: Вадење на елемент од магацин

- **Псевдокод**:

$$x \Leftarrow S: \quad \text{if } v = -1 \text{ then underflow (празен стек)}$$
$$\text{else } x \leftarrow S_v,\ v \leftarrow v\text{-}1$$

S – the stack
x – the element to be added
v – the top of the Stack
M – the maximum sze of the stack

Дефинирање на магацин (1)

```
#include <iostream>
using namespace std;
const int STACKSIZE = 80;
typedef char info_t;
struct stack {
    info_t S[STACKSIZE];
    int top;
    void Init();
    void StackOverflow();
    void StackUnderflow();
    void Push(info_t x);
    info_t Pop();
    info_t Peek();
    bool isEmpty();
};
```

```
void stack::Init(){
    top = -1;
}

void stack::StackOverflow() {
    cout<<"ERROR: StackOverflow"<<endl;
    exit(-1);
}

void stack::StackUnderflow(){
    cout<<"ERROR:StackUnderflow"<<endl;
    exit(-1);
}
```

Дефинирање на магацин (2)

```
void stack::Push(info_t x){
    if(top>=STACKSIZE - 1)
        StackOverflow();
    top++;
    S[top] = x;
}

info_t stack::Peek(){
    if(top<0)
        StackUnderflow();

    return S[top];
}
```

```
info_t stack::Pop(){
    if(top<0)
        StackUnderflow();
    int temp = top;
    top--;
    return S[temp];
}

bool stack::isEmpty(){
    return top<0;
}
```

Пример за употреба на магацин

- **Проверка дали даден израз е правилно форматиран** во однос на отворени и затворени загради – дали се балансирани заградите { } , [ ] и ( )?

- **Потсетување**: Едноставно пребројување не го решава проблемот (…[…(…]…)…)

Алгоритам за проверка на балансираност на загради

1. Создај празен магацин
2. Читај знаци сè до крајот на внесувањето
3. Ако знакот е отворена заграда {,[,(постави го на (во) магацинот
4. Ако знакот е затворена заграда },],), тогаш ако магацинот е празен пријави грешка, инаку извади знак од магацинот.
5. Ако симболот изваден од магацинот не е соодветната отворена заграда, пријави грешка.
6. Кога ќе заврши влезот, доколку магацинот не е празен пријави грешка

Пример 4.1

```
#include <iostream>
void ExprError(char * n, char * s);
int main(){
    stack mag, *m= &mag;
    char niza[80], *s=niza;
    m->Init();
    cout<<"->";
    cin>>niza;

    while(*s){
        switch (*s) {
        case '(':
        case '[':
        case '{':m->Push(*s); break;
        case ')':if (m->isEmpty() || m->Pop() != '(')
                    ExprError(niza, s); break;
        case ']':if (m->isEmpty() || m->Pop() != '[')
                    ExprError(niza, s); break;
        case '}':if (m->isEmpty() || m->Pop() != '{')
                    ExprError(niza, s); break;
        } // case
        s++;
    } // while
```

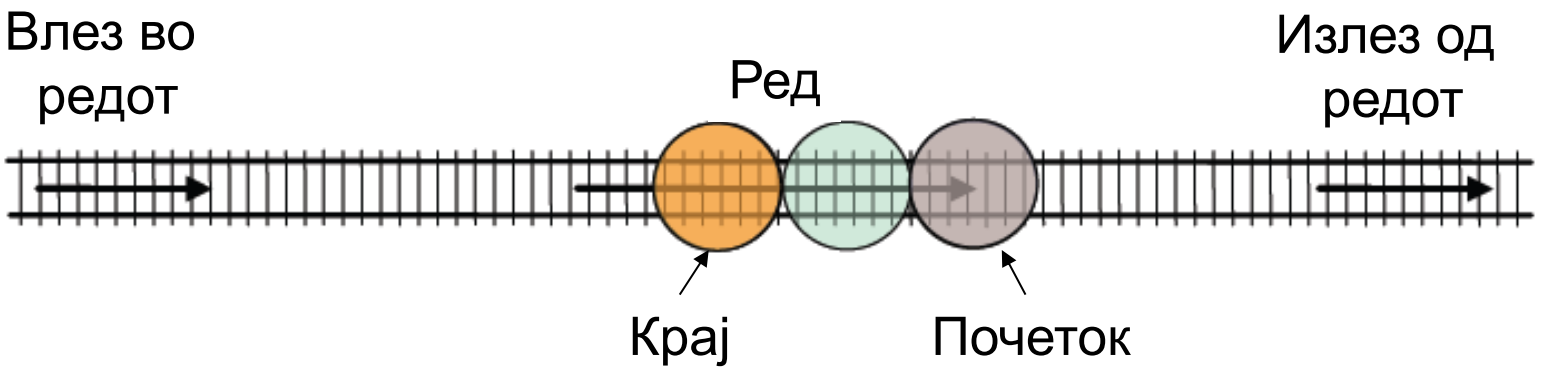
Пример 4.1 (продолжува)

```
    if (!m->isEmpty())
        ExprError(niza, s);
    else
        cout<<"Expression OK\n";

    return(0);
} //end main

void ExprError(char * n, char * s) {
    cout<<"Error in expresion: "<<endl;
    cout<<n<<endl;
    for (int i = 0; i<(s - n); i++)
        cout<<' ';
    cout<<'^';
    exit(-1);
} /* ()()[](([{()()}()[{()()}()]])) */
```

Ред на чекање (Queue) (1)

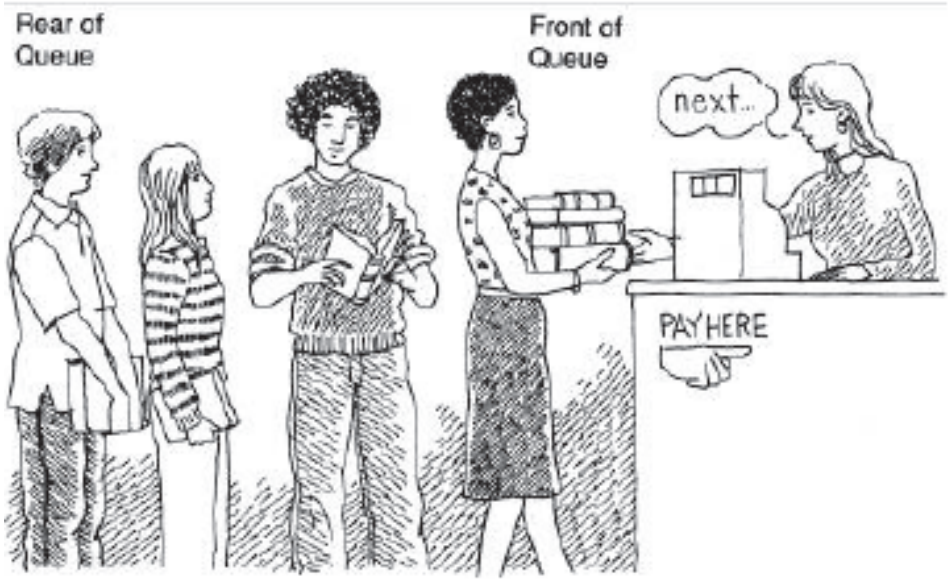
- Дефиниција: **Хомогена подредена линеарна структура со два краја** (почеток и крај)
- **FIFO (First In, First Out) структура** – структура во која елементот кој бил прв поставен во редот ќе биде и првиот кој ќе биде изваден од него
- **Елементите се додаваат на едниот крај** од редот (крај)
- **Елементите се вадат** од редот на **спротивниот крај** (почеток)
- => **Последица**: елементите се вадат од редот по истиот редослед во кој биле ставени во него

Ред на чекање (Queue) (2)



Примена

- **Секаде каде се опслужуваат клиенти** од било кој вид



- **За комуникација на два уреди** со различни брзини

Функции подржани од структурата ред на чекање

- **Основни**:
  - Додавање на елемент во редот на чекање
  - Вадење на елемент од редот на чекање
  - Проверка: дали редот е празен?

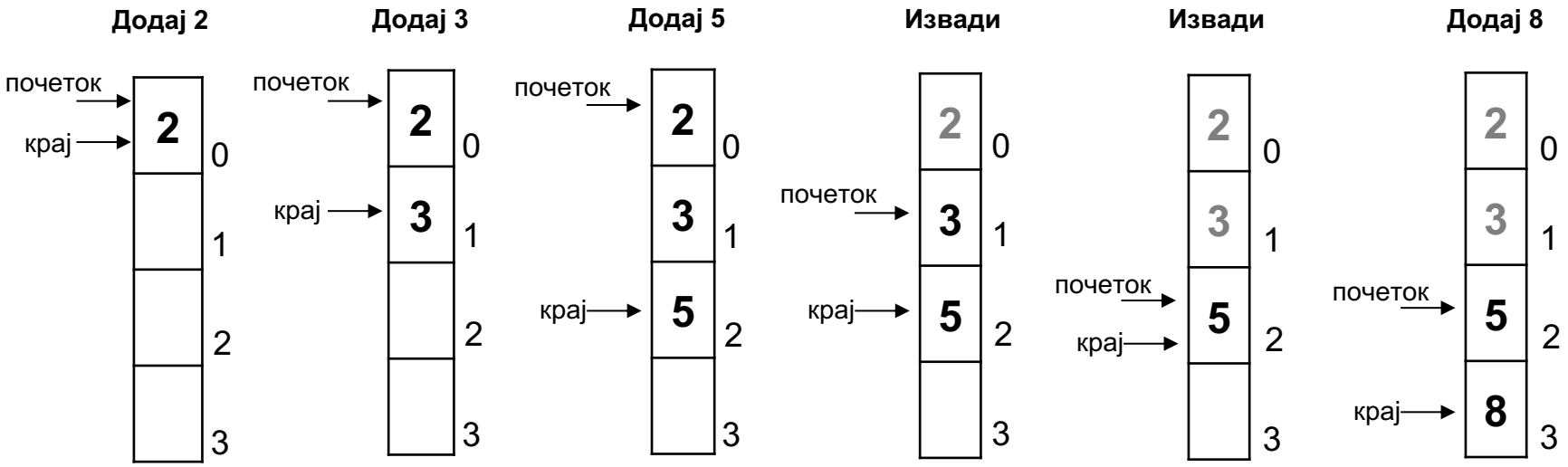
- **Дополнителни**:
  - Читање на елементот кој се наоѓа на крајот за вадење (без негово отстранување)
  - Проверка: дали редот е полн? (имплементациско ограничување)

Реализација на редот на чекање како еднодимензионална низа

- **Максималната должина** на редот (однапред се задава) **QUEUESIZE** – во редот ќе може да чекаат најмногу QUEUESIZE елементи

- За чување на елементите се користи **еднодимензионална низа** и **дополнителни променливи** (целобројни или покажувачи) кои ги означуваат **позициите за вадење и поставување на елементи (позиција на последно внесен елемент)** во редот (соодветно: почеток и крај)

Илустрација за линеарен ред



Дефинирање на ред на чекање (1)

```
#include <iostream>
using namespace std;
const int QUEUESIZE = 20;

typedef int info_t;
struct queue {
    info_t Q[QUEUESIZE];
    int f, r; /*r - do kade imash
zapishano, f – kade treba da chitas*/
    void QueueOverflow();
    void QueueUnderflow();
    void Put(info_t x);
    info_t Pull();
    info_t Peek();
    void Init();
    bool isEmpty();
    int length();
};
```

```
void queue::QueueOverflow(){
    cout<<"ERROR:
QueueOverflow"<<endl;
    exit(-1);
}

void queue::QueueUnderflow(){
    cout<<"ERROR:
QueueUnderflow"<<endl;
    exit(-1);
}

void queue::Init(){
    r = f = -1;
}
```

Линеарен ред

```
void queue::Put(info_t x){
    if (r >= QUEUESIZE-1)
        QueueOverflow();
    else {
        if (f == -1)
            f = 0;
        Q[++(r)] = x;
    }
}
```

```
info_t queue::Pull(){
    info_t x;
    if (f == -1)
        QueueUnderflow();
    else {
        x = Q[f];
        if (f == r)
            f = r = -1;
        else f++;
    }
    return x;
}
```

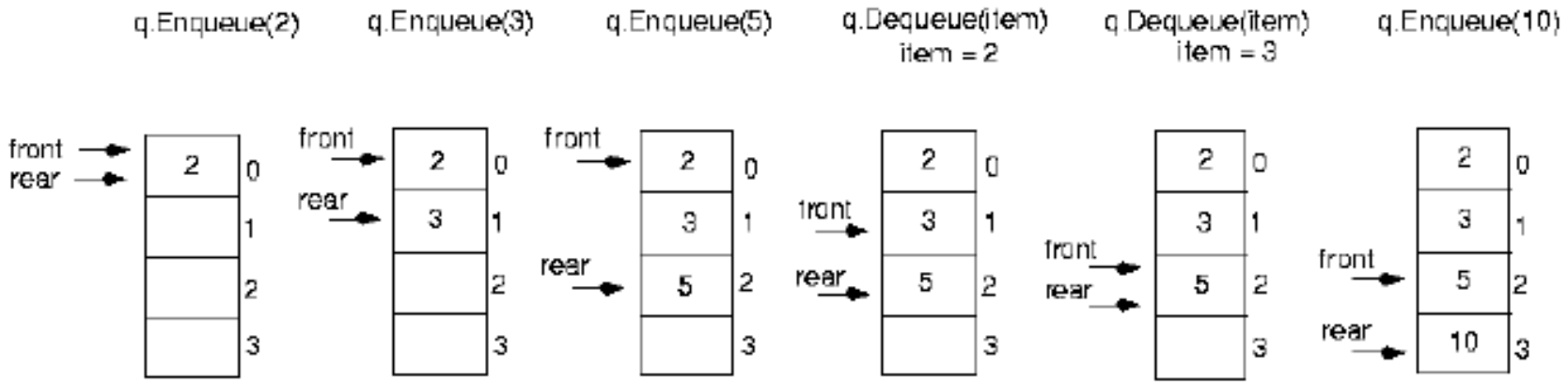
Линеарен ред

```
bool Queue::isFull()
{
        return (rear == MAX - 1);
}
```

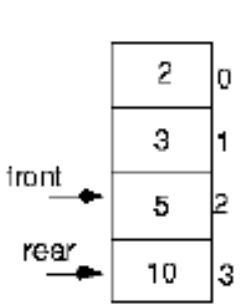
```
bool Queue::isEmpty() {
    return (rear == -1);
}
```

```
int Queue::length(){
    return rear - front + 1;
}
```

Потреба од прстенести редови



q.Enqueue(20) ???

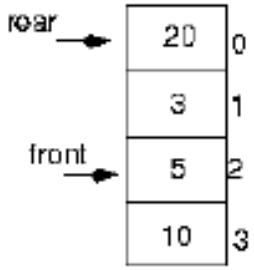


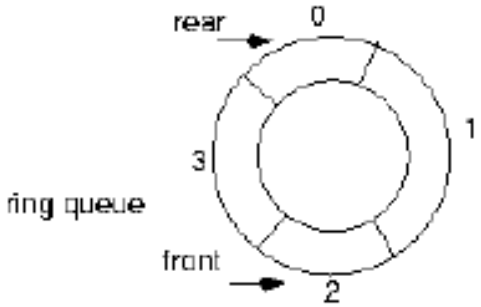
Let the queue elements "wrap around"

```
if(rear == maxQue -1)
  rear = 0;
else
  rear = rear + 1;
```

or

```
rear = (rear + 1) % maxQue;
```





Функција 1: Додавање на елемент во редот на чекање

- **Псевдокод**:

$$
\begin{array}{lllll}
Q \Leftarrow x: & \text{if} & r == \text{M-1} & \text{then} & r \leftarrow 0 \\
 & & & \text{else} & r \leftarrow r+1 \\
 & \text{if} & r == \text{f} & \text{then} & \text{преполнување} \\
 & & & \text{else} & S_r \leftarrow x
\end{array}
$$

Функција 2: Вадење на елемент од редот на чекање

- **Псевдокод**:

$$x \Leftarrow Q : \text{if } f == \text{-1 then} \quad \text{празен ред}$$

$$\text{else} \begin{cases} x \leftarrow Q_f \\ \text{if } f == M-1 \text{ then } f \leftarrow 0 \\ \text{else } f \leftarrow f+1 \end{cases}$$

Прстенест ред

```
void queue::Put(info_t x){
    //ako svrtil cel krug
    if (r >= QUEUESIZE-1)
        r = 0;
    else
        r++;
    //ako e poln
    if (r == f)
        QueueOverflow();
    else {
        Q[r] = x;
        //ako e prv element
        if (f == -1)
           f = r;
    }
}
```

```
info_t queue::Pull(){
    info_t x;
    //ako e prazen
    if (f == -1)
        QueueUnderflow();
    else {
        x = Q[f];
        //ako e samo eden element
        if (f == r)
            f = r = -1;
        else
            //ako front e na kraj
            if (f >= QUEUESIZE-1)
                f = 0;
            else
                f++;
    }
    return x;
}
```

Дефинирање на ред на чекање (прод.)

```
info_t queue::Peek(){
    if (f == -1)
        QueueUnderflow();
    return Q[f];
}

void queue::Init(){
    r = f = -1;
}

bool queue::isEmpty(){
    return f<0;
}

int queue::QueueLen() {
    if (r == -1)
        return 0;
    else if (r >= f)
        return r - f + 1;
    else
        return (QUEUESIZE - (f - r)+1);
}
```