ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

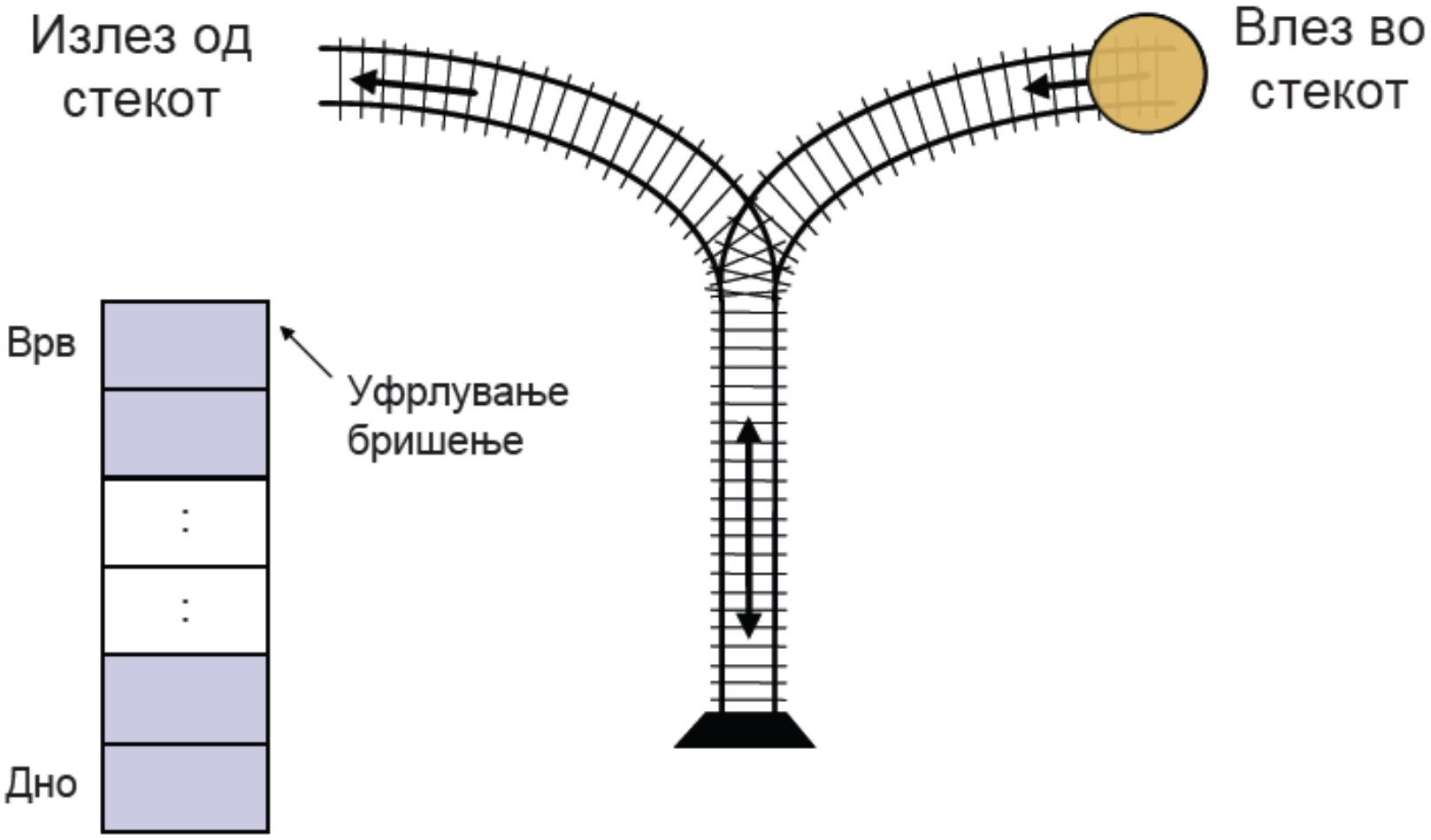
Магацини (stacks) и редови (queues)

– 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** во 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 еднодимензионално поле

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

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

Функција 1: Додавање на елемент на магацин

- **Псевдокод**:

$S \Leftarrow x$: $v \leftarrow v+1$

if $v > M$ then overflow (полн стек)

else $S_v \leftarrow x$

Функција 2: Вадење на елемент од магацин

- **Псевдокод**:

$$x \Leftarrow S: \quad \text{if } v = 0 \text{ then underflow (празен стек)}$$
$$\text{else } x \leftarrow S_v,\ v \leftarrow v\text{-}1$$

Дефинирање на магацин (1)

```
#include <stdio.h>
#include <stdlib.h>
#define STACKSIZE 20

typedef char info_t;

typedef struct s {
        info_t S[STACKSIZE];
        int top;
} stack;

void Init(stack * m) {
        m->top = -1;
}
```

```
void StackOverflow(void) {
        fprintf(stderr, "ERROR:
StackOverflow\n");
        exit(-1);
}

void StackUnderflow(void) {
        fprintf(stderr, "ERROR:
StackUnderflow\n");
        exit(-1);
}
```

Дефинирање на магацин (2)

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

info_t Peek(stack * m) {
        if (m->top<0)
                StackUnderflow();
        return (m->S[(m->top)]);
}
```

```
info_t Pop(stack *m) {
        if (m->top<0)
                StackUnderflow();
        return(m->S[(m->top)--]);
}

int isEmpty(stack *m) {
        return((m->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. Создади 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

```
int main() {
        stack mag, *m = &mag;
        char niza[80], *s = niza;
        Init(m);
        printf("->");
        scanf("%s", niza);

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

Пример 4.1 (продолжува)

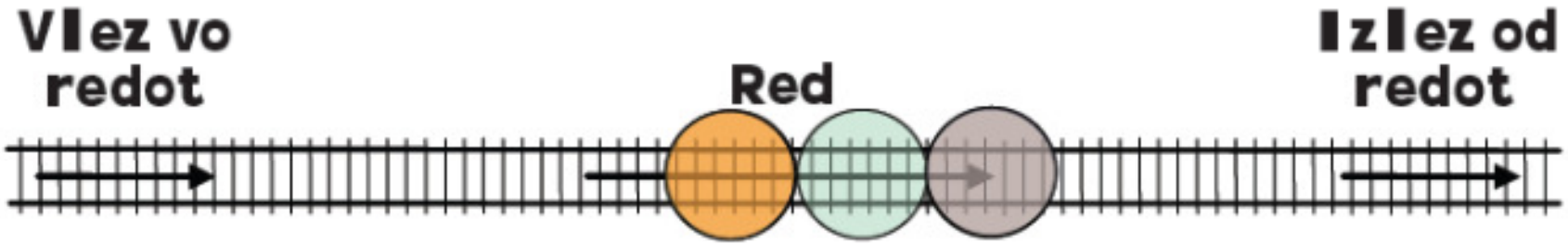
```
        if (!isEmpty(m))
                ExprError(niza, s);
        else
                printf("Expression OK\n");
        return(0);
} //end main

void ExprError(char * n, char * s) {
        int i;
        printf("Error in expresion: %s\n", n);
        printf(" ");
        for (i = 0; i<(s - n); i++)
                putchar(' ');
        putchar('^');
        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?

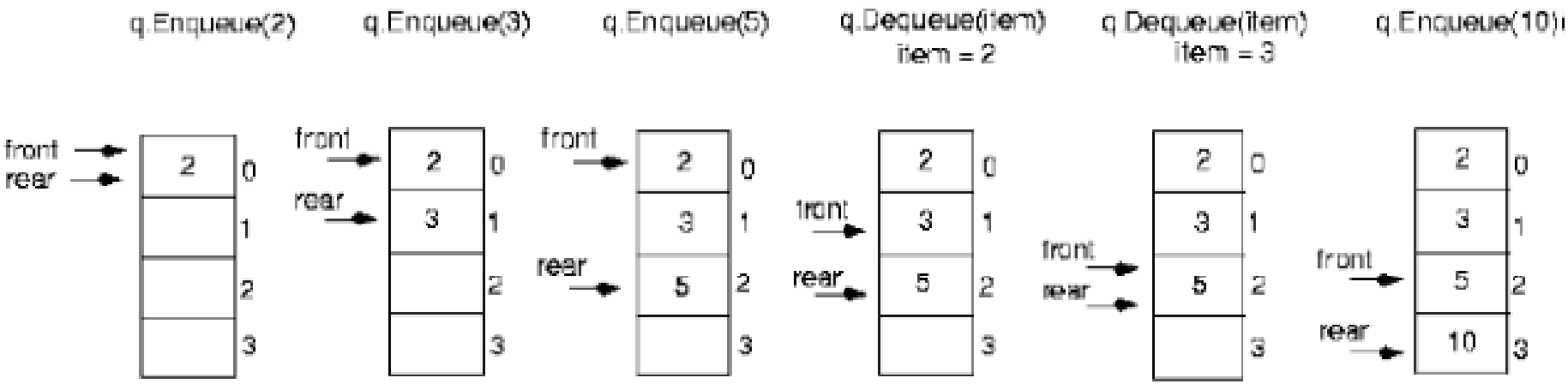
- **Дополнителни**:
  - Читање на елементот кој се наоѓа на крајот за вадење (без негово отстранување)
  - Проверка: дали редот е полн? (имплементациско ограничување)

Реализација на редот на чекање како еднодимензионално поле

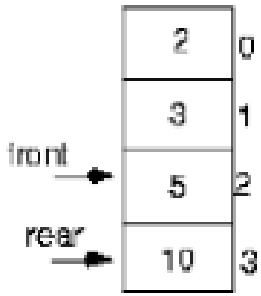
- **Максималната должина** на редот (однапред се задава) **QUEUESIZE** – во редот ќе може да чекаат најмногу QUEUESIZE елементи

- За чување на елементите се користи **еднодимензионално поле** и **дополнителни променливи** (целобројни или покажувачи) кои ги означуваат **позициите за вадење и поставување на елемети** во редот (соодветно: почеток и крај)

Потреба од прстенести редови



q.Enqueue(20) ???

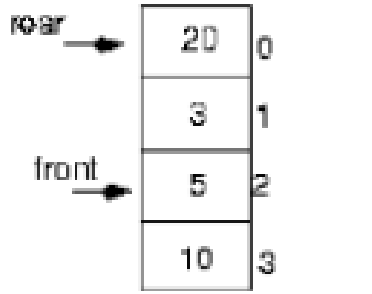


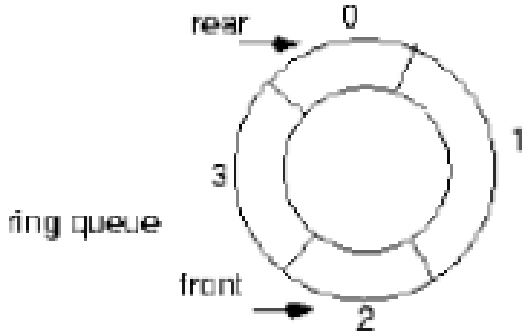
Let the queue elements "wrap around"

```
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**:

$$Q \Leftarrow x: \quad \text{if } q = M \text{ then } q \leftarrow 1$$
$$\text{else } q \leftarrow q + 1$$
$$\text{if } q = p \text{ then преполнување}$$
$$\text{else } Q_q \leftarrow x$$

Функција 2: Вадење на елемент од редот на чекање

- **Псевдокод**:

$$x \Leftarrow Q: \quad \text{if } q = p \text{ then празен ред}$$

$$\text{else} \begin{cases} \text{if } p = M \text{ then } p \leftarrow 1 \\ \qquad\qquad \text{else } p \leftarrow p + 1 \\ x \leftarrow Q_p \end{cases}$$

Дефинирање на ред на чекање (1)

```
#include <stdio.h>
#include <stdlib.h>
#define QUEUESIZE 20
typedef int info_t;

typedef struct s {
        info_t Q[QUEUESIZE];
        int f, r; // info_t f,r;
} queue;

void QueueOverflow(void) {
        fprintf(stderr, "ERROR: QueueOverflow\n");
        exit(-1);
}

void QueueUnderflow(void) {
        fprintf(stderr, "ERROR: QueueUnderflow\n");
        exit(-1);
}
```

Линеарен ред

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

```
info_t Pull(queue * m) {
        int x;
        if (m->f == -1)
                QueueUnderflow();
        else {
                x = m->Q[m->f];
                if (m->f == m->r)
                    m->f = m->r = -1;
                else m->f++;
        }
        return x;
} // end Pull
```

Прстенест ред

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

```
info_t Pull(queue * m) {
        info_t x;
        if (m->f == -1)
                QueueUnderflow();
        else {
                x = m->Q[m->f];
                if (m->f == m->r)
                    m->f = m->r = -1;
                else
                    if (m->f >= QUEUESIZE-1)
                        m->f = 0;
                    else
                        m->f++;
        }
        return(x);
} // Pull
```

Дефинирање на ред на чекање (прод.)

```
info_t Peek(queue * m) {
        if (m->f == -1)
                QueueUnderflow();
        return (m->Q[m->f]);
} // end Peek
void Init(queue * m) {
        m->f = m->r = -1;
} // end Init
int isEmpty(queue * m) {
        return((m->f)<0);
} // end isEmpty
int QueueLen(queue * m) {
        if (m->r == -1)
                return(0);
        else if (m->r >= m->f)
                return m->r - m->f + 1;
        else
                return (QUEUESIZE - (m->f - m->r));
} // end QueueLen
```