ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Еднодимензионални податочни структури

Алгоритми и податочни структури

Аудиториска вежба 5

Еднодимензионални податочни структури

- Со помош на основните податочни структури, можат да се креираат некои апстрактни податочни структури со специфична намена
  - стек(stack)
  - редица (queue)
  - приоритетна редица (priority queue)

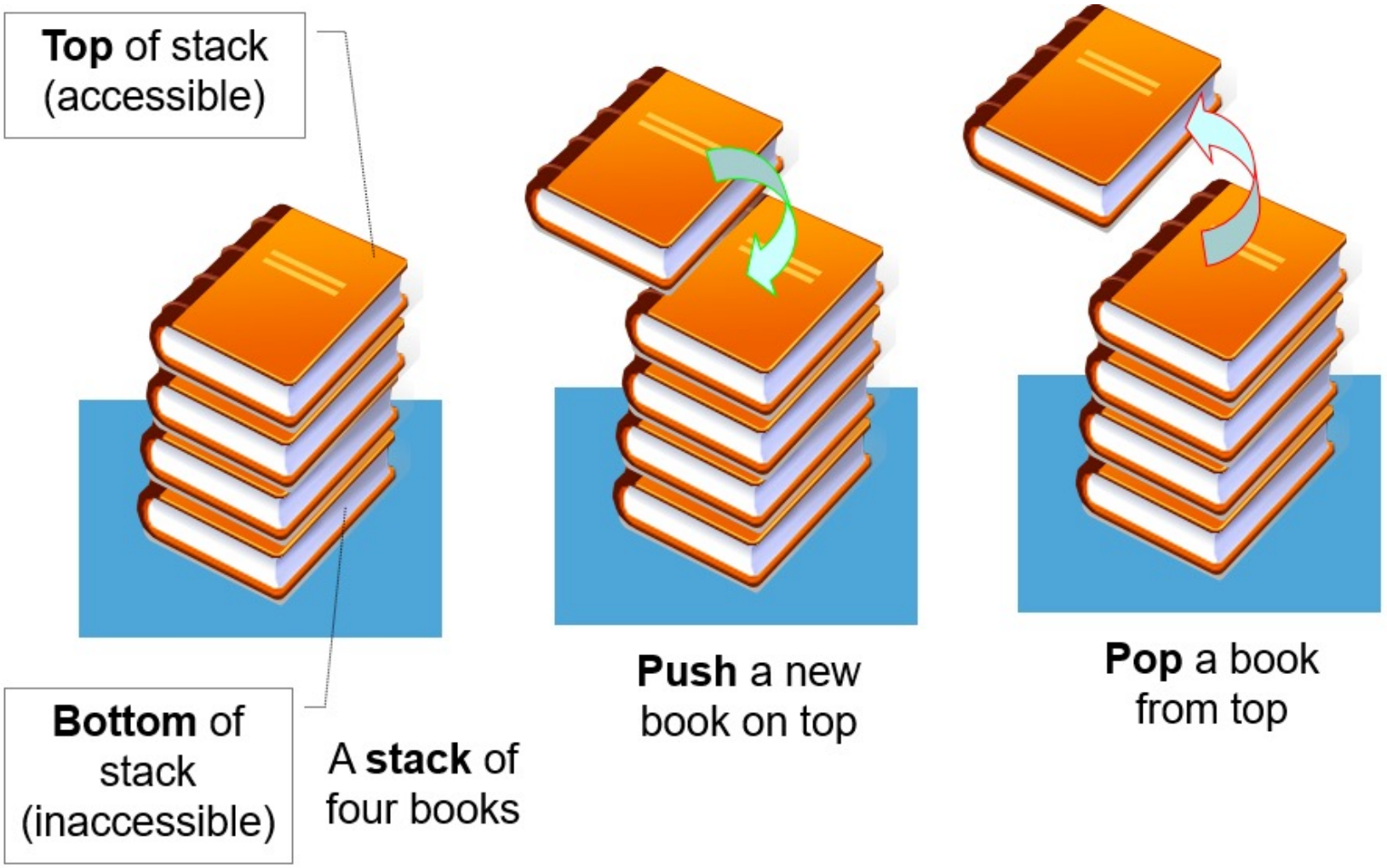
Стек

- **Стекот** работи на принципот LIFO (Last In – First Out).
- Во македонската литература се сретнува и со називот **Магацин**
- Операции:
  - за вметнување елемент на врвот на стекот – push
  - за бришење на елемент од врвот на стекот – pop
  - дополнително (во кодовите за Java) за да се прочита елементот на врвот од стекот, без притоа да се избрише - peek

Стек

- Како една од фундаменталните податочни структури стекот се користи во многу алгоритми и апликации
- Примери за употреба на стек
  - Backtracking алгоритми
  - Back/Forward функционалности во прелистувачи
  - Undo/Redo функционалности
  - Превртување на поврзана листа
  - Евалуација на аритметички изрази
  - Проверка на коректност на загради во израз
  - Парсери
  - Имплицитно во рекурзија
  - Повици на функции

Примена на стек



Имплементации на стек

- Постојат две генерални имплементации на стек:
  - Ограничен стек – кој се имплементира со користење на низа
  - Неограничен стек – кој се имплементира со користење на листа

Интерфејс за стек - Java

```
public interface Stack<E> {
    // Elementi na stekot se objekti od proizvolen tip.
    // Metodi za pristap:

    public boolean isEmpty();
    // Vrakja true ako i samo ako stekot e prazen.

    public E peek();
    // Go vrakja elementot na vrvot od stekot.

    // Metodi za transformacija:
    public void clear();
    // Go prazni stekot.

    public void push(E x);
    // Go dodava x na vrvot na stekot.

    public E pop();
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
}
```

Се користи и во двете имплементации на стек

Стек со низа - Java

```
public class ArrayStack<E> implements Stack<E> {
        private E[] elems; //elems[0...depth-1] se negovite elementi.
        private int depth; //depth e dlabochinata na stekot

        public ArrayStack(int maxDepth) {
            // Konstrukcija na nov, prazen stek.
        }

        public boolean isEmpty() {
            // Vrakja true ako i samo ako stekot e prazen.
        }
        public E peek() {
            // Go vrakja elementot na vrvot od stekot.
        }

        public void clear() {
            // Go prazni stekot.
        }

        public void push(E x) {
            // Go dodava x na vrvot na stekot.
        }

        public int size() {
            // Ja vrakja dolzinata na stack-ot.
        }
        public E pop() {
            // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
        }
    }
```

Стек со низа - Java

```
public boolean isEmpty() {
    // Vrakja true ako i samo ako stekot e prazen.
    return (depth == 0);
}

public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (depth == 0)
        throw new NoSuchElementException();
    return elems[depth - 1];
}

public void clear() {
    // Go prazni stekot.
    for (int i = 0; i < depth; i++) elems[i] = null;
    depth = 0;
}
```

Стек со низа - Java

```
public void push(E x) {
    // Go dodava x na vrvot na stekot.
    elems[depth++] = x;
}

public int size() {
    // Ja vrakja dolzinata na stack-ot.
    return depth;
}

public E pop() {
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
    if (depth == 0)
        throw new NoSuchElementException();
    E topmost = elems[--depth];
    elems[depth] = null;
    return topmost;
}
```

Стек со листа - Java

```
public class LinkedStack<E> implements Stack<E> {
        // top e link do prviot jazol ednostrano-povrzanata
        // lista koja sodrzi gi elementite na stekot .
        private SLLNode<E> top;
        int size;

        public LinkedStack() {    }

        public boolean isEmpty() {}

        public E peek() {}

        public void clear() {}

        public void push(E x) {}

        public int size() {}

        public E pop() {}
    }
```

Стек со листа - Java

```
public LinkedStack() {
    // Konstrukcija na nov, prazen stek.
    top = null;
    size = 0;
}

public boolean isEmpty() {
    // Vrakja true ako i samo ako stekot e prazen.
    return (top == null);
}

public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (top == null)
        throw new NoSuchElementException();
    return top.element;
}

public void clear() {
    // Go prazni stekot.
    top = null;
    size = 0;
}
```

Стек со листа - Java

```
public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (top == null)
        throw new NoSuchElementException();
    return top.element;
}

public void push(E x) {
    // Go dodava x na vrvot na stekot.
    top = new SLLNode<E>(x, top);
    size++;
}

public int size() {
    // Ja vrakja dolzinata na stekot.
    return size;
}

public E pop() {
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
    if (top == null)
        throw new NoSuchElementException();
    E topElem = top.element;
    size--;
    top = top.succ;
    return topElem;
}
```

Класата `Stack` во Java

- Класата `Stack` претставува LIFO стек на објекти
- Ја наследува и проширува класата `Vector` со пет операции кои дозволуваат вектор да биде третиран како стек

  `public class` **`Stack<E>`** `extends` **`Vector<E>`**
- Опис на методите:
  - `empty()`
    - враќа: true ако и само ако стекот не содржи елементи; false во останатите случаи
  - `peek()`
    - враќа: објектот кој е најгоре на стекот (последниот елемент од Vector објектот), без притоа да го вади од структурата
  - `pop()`
    - враќа: објектот кој е најгоре на стекот (последниот елемент од Vector објектот и притоа го брише елементот)
  - `push(E item)`
    - враќа: item влезниот аргумент
  - `search(Object o)`
    - враќа: дистанцата до првото појавување на o од почетокот на стекот; -1 доколку објектот не е во стекот

Задача 1.

- Да се провери коректноста на заградите во еден израз.
- Еден израз има коректни загради ако:
  - За секоја лева заграда, подоцна следува соодветна десна заграда
  - За секоја десна заграда претходно постои лева заграда
  - Секој под-израз меѓу пар од две загради содржи коректен број на загради
- Примери на изрази со коректни и некоректни загради:

$s \times (s - a) \times (s - b) \times (s - c)$ — коректни

$(- b + \sqrt{[b^2 - 4ac]}) / 2a$ — коректни

$s \times (s - a) \times (s - b \times (s - c)$ — некоректни

$s \times (s - a) \times s - b) \times (s - c)$ — некоректни

$(- b + \sqrt{[b^2 - 4ac)}] / 2a$ — некоректни

Задача 1. - Java

```
public class Zagradi {
        public static boolean daliKorektni(String phrase) {
            // Test whether phrase is well-bracketed.
            ArrayStack<Character> bracketStack = new ArrayStack<Character>(100);
            for (int i = 0; i < phrase.length(); i++) {
                char cur = phrase.charAt(i);
                if (cur == '(' || cur == '[' || cur == '{')
                    bracketStack.push(cur);
                else if (cur == ')' || cur == ']' || cur == '}') {
                    if (bracketStack.isEmpty()) return false;
                    char left = bracketStack.pop();
                    if (!daliSoodvetni(left, cur)) return false;
                }
            }
            return (bracketStack.isEmpty());
        }
```

Задача 1. - Java

```
    public static boolean daliSoodvetni(char left, char right) {
        // Test whether left and right are matching brackets
        // (assuming that left is a left bracket and right is a right bracket).
        switch (left) {
            case '(':
                return (right == ')');
            case '[':
                return (right == ']');
            case '{':
                return (right == '}');
        }
        return false;
    }

    public static void main(String[] args) {
        String phrase = "s x (s - a) x (s - b) x (s - c)";
//        String phrase = "s x (s - a) x s - b) x (s - c)";
        System.out.println(phrase + " ima "
                + (daliZagraditeSePravilni(phrase) ? "korektni" : "nekorektni")
                + " zagradi.");
    }
}
```

Задача 2.

- ЗА ДОМА. Да се напише алгоритам кој ќе врши евалуација на израз во постфикс нотација.
- Пр. 5 9 + 2 * 6 5 * + изразот е во постфикс нотација

Редица

- **Редицата** работи на принципот FIFO (First In – First Out).
- Операции:
  - за вметнување на елемент на едниот крај на редицата т.е. на опашката на редицата – enqueue
  - за бришење на елемент на другиот крај на редицата, т.е. од главата (почетокот) на редицата– dequeue
  - дополнително (во кодовите за Java) за да се добие елементот на почетокот (главата) од редицата, без притоа да се избрише - peek

Редица - примена

- Кај мрежни рутери
- Плејлисти
- Кај мрежните печатачи, каде повеќе корисници испраќаат документи за печатење на делен печатач
- Кај хард дисковите, односно во нивните драјвери, каде повеќе процеси бараат пристап до податоците од дискот
- Во оперативните системи, каде повеќе процеси чекаат на ред да бидат опслужени од процесорот

Имплементации на редица

- Постојат две генерални имплементации на редица:
  - Ограничена редица – која се имплементира со користење на низа
  - Неограничена редица – кој се имплементира со користење на листа

Интерфејс за редица- Java

Се користи и во двете имплементации на редица

```
public interface Queue<E> {
        // Elementi na redicata se objekti od proizvolen tip.
        // Metodi za pristap:
        public boolean isEmpty();
        // Vrakja true ako i samo ako redicata e prazena.

        public int size();
        // Ja vrakja dolzinata na redicata.

        public E peek();
        // Go vrakja elementot na vrvot t.e. pocetokot od redicata.

        // Metodi za transformacija:

        public void clear();
        // Ja prazni redicata.

        public void enqueue(E x);
        // Go dodava x na kraj od redicata.

        public E dequeue();
        // Go otstranuva i vrakja pochetniot element na redicata.
    }
```

Редица со низа - Java

```
class ArrayQueue<E> {
        // Redicata e pretstavena na sledniot nacin:
        // length go sodrzi brojot na elementi.
        // Ako length > 0, togash elementite na redicata se zachuvani vo
    elems[front...rear-1]
        // Ako rear > front, togash vo  elems[front...maxlength-1] i elems[0...rear-1]
        E[] elems;
        int length, front, rear;

        // Konstruktor ...
        public ArrayQueue(int maxlength) {
            elems = (E[]) new Object[maxlength];
            clear();
        }

        public boolean isEmpty() {
            // Vrakja true ako i samo ako redicata e prazena.
            return (length == 0);
        }

        public int size() {
            // Ja vrakja dolzinata na redicata.
            return length;
        }
        public void clear() {
            // Ja prazni redicata.
            length = 0;
            front = rear = 0;  // arbitrary
        }
```

Редица со низа - Java

```
public E peek() {
        // Go vrakja elementot na vrvot t.e. pocetokot od redicata.
        if (length > 0)
            return elems[front];
        else
            throw new NoSuchElementException();
    }

    public void enqueue(E x) {
        // Go dodava x na kraj od redicata.
        if (length == elems.length)
            throw new NoSuchElementException();
        elems[rear++] = x;
        if (rear == elems.length) rear = 0;
        length++;
    }

    public E dequeue() {
        // Go otstranuva i vrakja pochetniot element na redicata.
        if (length > 0) {
            E frontmost = elems[front];
            elems[front++] = null;
            if (front == elems.length) front = 0;
            length--;
            return frontmost;
        } else
            throw new NoSuchElementException();
    }
}
```

Редица со листа - Java

```
public class LinkedQueue<E> implements Queue<E> {
       // Redicata e pretstavena na sledniot nacin:
        // length go sodrzi brojot na elementi.
        // Elementite se zachuvuvaat vo jazli dod SLL
        // front i rear se linkovi do prviot i posledniot jazel soodvetno.
        SLLNode<E> front, rear;
        int length;

        // Konstruktor ...
        public LinkedQueue () {
            clear();
        }

        public boolean isEmpty () {
               // Vrakja true ako i samo ako redicata e prazena.
            return (length == 0);
        }

        public int size () {
           // Ja vrakja dolzinata na redicata.
            return length;
        }
        public E peek () {
               // Go vrakja elementot na vrvot t.e. pocetokot od redicata.
            if (front == null)
                throw new NoSuchElementException();
            return front.element;
        }
```

Редица со листа - Java

```
public void clear () {
        // Ja prazni redicata.
         front = rear = null;
         length = 0;
    }
    public void enqueue (E x) {
        // Go dodava x na kraj od redicata.
         SLLNode<E> latest = new SLLNode<E>(x, null);
         if (rear != null) {
             rear.succ = latest;
             rear = latest;
         } else
             front = rear = latest;
         length++;
    }

    public E dequeue () {
        // Go otstranuva i vrakja pochetniot element na redicata.
         if (front != null) {
             E frontmost = front.element;
             front = front.succ;
             if (front == null)  rear = null;
             length--;
             return frontmost;
         } else
             throw new NoSuchElementException();
    }
}
```

Интерфејсот Queue во Java

- Нема готова класа за редица во Java, само интерфејс.
- Ова е `Queue` интерфејсот во Java:

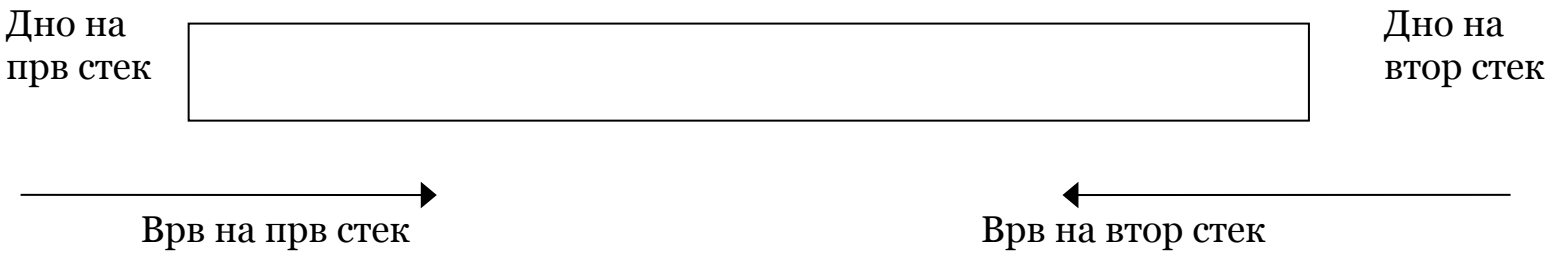
```
public interface Queue<E> extends Collection<E>
{
        E element();
        boolean offer(E e);
        E peek();
        E poll();
        E remove();
}
```

Queue во Java

- Класи кои го имплементираат Queue интерфејсот се:
  - `AbstractCollection`
  - `LinkedList`
  - `PriorityQueue`
  - `LinkedBlockingQueue`
  - `BlockingQueue`
  - `ArrayBlockingQueue`
  - `LinkedBlockingQueue`
  - `PriorityBlockingQueue`

Задача 3.

- Да се изменат функциите за стек (магацин) така да овозможат користење на два стека кои делат заеднички простор, односно поле. Следната слика ја покажува логичката структура на ваквите стекови.



- Да се напишат функции за додавање и бришење на елемент од стековите, така што стекот кој се користи се одредува со променлива која се пренесува како аргумент во соодветните функции.

Задача 3. - Java

```
public class DoubleArrayStack<E> {
        // Stekot e pretstaven na sledniot nacin:
        // depth1 e dlabochinata na prviot stekot,
        // depth2 e dlabochinata na prviot stekot,
        // elems[maxDepth-depth2...maxDepth-1] se elementi na vtoriot stek,
        private E[] elems; // elems[0...depth-1] se elementi na prviot stek,
        private int depth1;
        private int depth2;

        public DoubleArrayStack(int maxDepth) {
            // Konstrukcija na nov, prazen spodelen stek.
            elems = (E[]) new Object[maxDepth];
            depth1 = 0;
            depth2 = 0;
        }
        public boolean isEmptyFirst() {
            // Vrakja true ako i samo ako prviot stek e prazen.
            return (depth1 == 0);
        }
        public boolean isEmptySecond() {
            // Vrakja true ako i samo ako vtoriot stek e prazen.
            return (depth2 == 0);
        }
        public boolean isFull() {
            // Vrakja true ako i samo ako celata niza e polna.
            return (depth1 + depth2 == elems.length);
        }
```

Задача 3. - Java

```
public E peekFirst() {
        // Go vrakja elementot na vrvot od prviot stek.
        if (depth2 == 0)
            throw new NoSuchElementException();
        return elems[depth1 - 1];
    }

    public E peekSecond() {
        // Go vrakja elementot na vrvot od vtoriot stek.
        if (depth1 == 0)
            throw new NoSuchElementException();
        return elems[elems.length - depth2];
    }

    public void clearFirst() {
        // Go prazni prviot stek.
        for (int i = 0; i < depth1; i++)
            elems[i] = null;
        depth1 = 0;
    }


    public void clearSecond() {
        // Go prazni vtoriot stek.
        for (int i = elems.length - 1; i >= elems.length - depth2; i--)
            elems[i] = null;
        depth2 = 0;
    }
```

Задача 3. - Java

```
public void pushFirst(E x) {
        // Go dodava x na vrvot na prviot stek.
        if (!this.isFull())
            elems[depth1++] = x;
        else
            System.out.println("Error, the array is full");
    }

    public void pushSecond(E x) {
        // Go dodava x na vrvot na vtoriot stek.
        if (!this.isFull())
            elems[elems.length - (++depth2)] = x;
        else
            System.out.println("Error, the array is full");
    }


    public E popFirst() {
        // Go otstranuva i vrakja elementot shto e na vrvot na prviot stek.
        if (depth1 == 0)
            throw new NoSuchElementException();
        E topmost = elems[--depth1];
        elems[depth1] = null;
        return topmost;
    }
```

Задача 3. - Java

```
    public E popSecond() {
        // Go otstranuva i vrakja elementot shto e na vrvot na vtoriot stek.
        if (depth2 == 0)
            throw new NoSuchElementException();
        E topmost = elems[elems.length - depth2];
        elems[depth2--] = null;
        return topmost;
    }

    public String pecatiNizata() {
        StringBuilder ret = new StringBuilder("Elementite se: ");
        for (E elem : elems) ret.append(elem).append(" ");
        return ret.toString();
    }

    public static void main(String[] args) {
        DoubleArrayStack<Integer> d = new DoubleArrayStack<Integer>(6);
        d.pushFirst(1);
        d.pushFirst(2);
        d.pushFirst(3);
        d.pushSecond(-1);
        d.pushSecond(-2);
        d.pushSecond(-3);
        System.out.println("Vrv na prv: " + d.peekFirst() + ", dolzina na prv: " + d.depth1);
        System.out.println("Vrv na vtor: " + d.peekSecond() + ", dolzina na vtor: " + d.depth2);
        d.pushFirst(4);
        d.popFirst();
        d.pushFirst(4);
        System.out.println("Vrv na prv: " + d.peekFirst() + ", dolzina na prv: " + d.depth1);
        d.pecatiNizata();
    }
}
```

Приоритетна редица

- Еднодимензионална линеарна секвенца од елементи, каде секој елемент има свој приоритет.
- Елементот со најголем приоритет секогаш прв се вади од листата
- Должина на приоритетна редица е бројот на елементи што ги содржи
  - Празната приоритетна редица има должина 0

Приоритетна редица

- Две можности при имплементација
  1. Подредена редица
  2. Неподредена редица
- И двете се независни од фундаменталниот податочен тип што ќе се користи (во однос на перформансите)
  - Со низа – ограничена приоритетна редица со предефиниран капацитет
  - Со поврзана листа – неограничена приоритетна редица