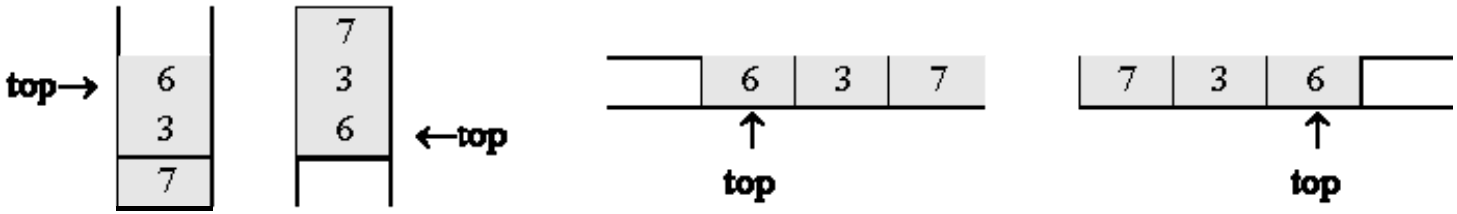
Magacin, red i dvostrani red

Magacin

- *Magacin (engl. stack) je linearna struktura podataka kod koje se može pristupiti samo poslednje dodatom podataku.To je struktura koja radi po principu LIFO (last-in, first-out).*

Organizacuja magacina

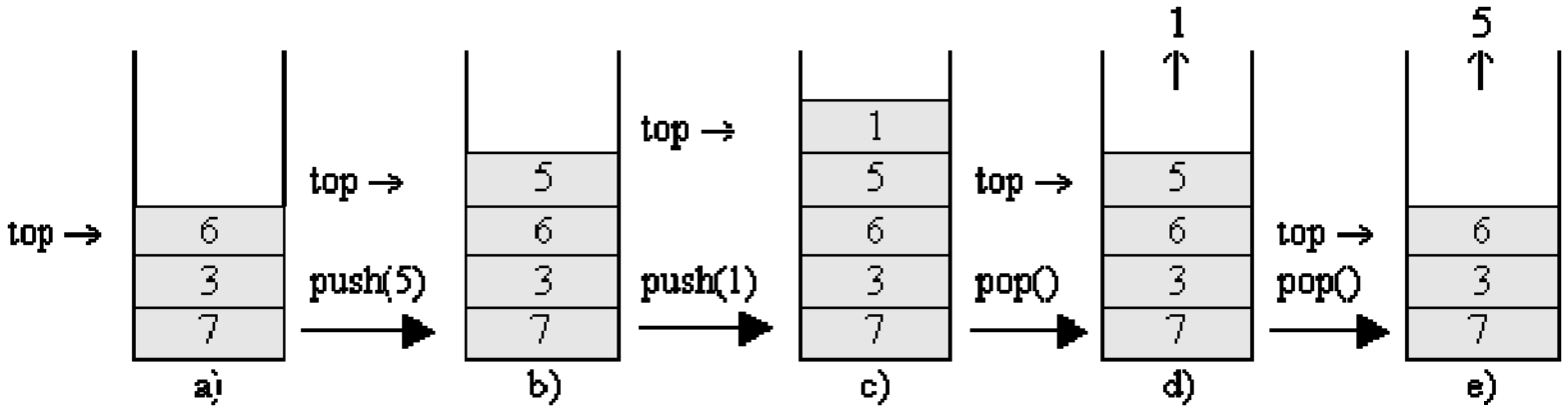
Magacin je specijalan slučaj linearne liste kod koje se operacije dodavanja i brisanja vrše samo na jednom kraju, koji se naziva **vrh** (engl. *top*) magacina.



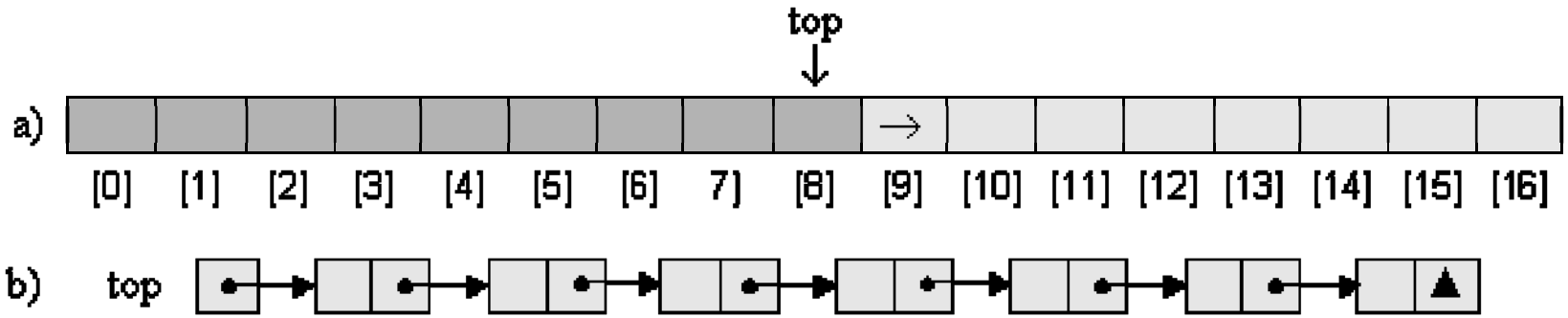
Osnovne operacije

- **push** - dodaje element na vrh magacina (*OverflowException*),
- **pop** - čita i uklanjanja element sa vrha magacina (*UnderflowException*),
- **getTop** - čita element sa vrha magacina, ali ga ne uklanjanja (*UnderflowException*),
- **isEmpty** - proverava da li je magacin prazan i
- **numberOfElements** - vraća broj elemenata smeštenih u magacinu.

Primer operacija

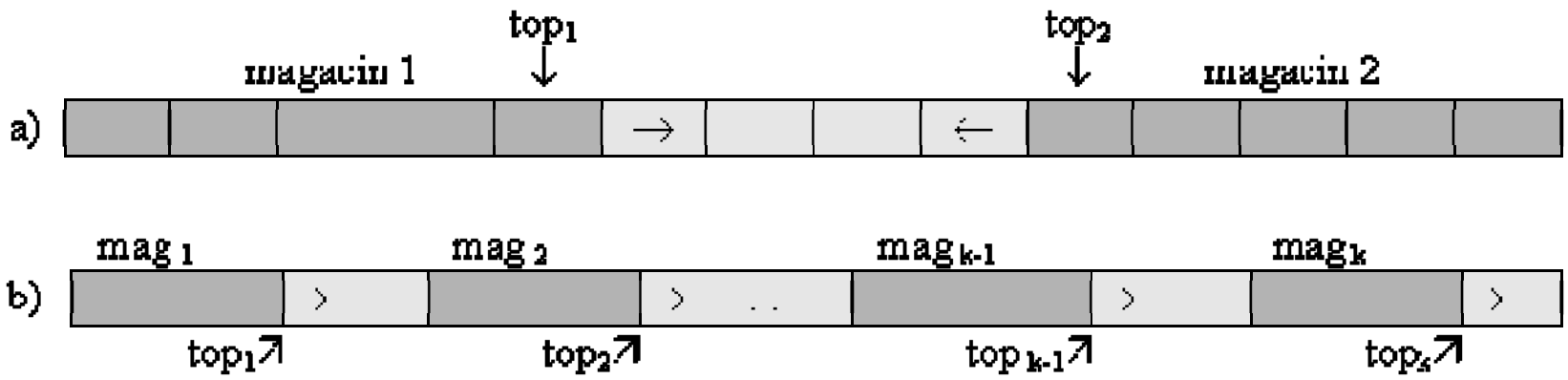


Memorijska reperezentacija

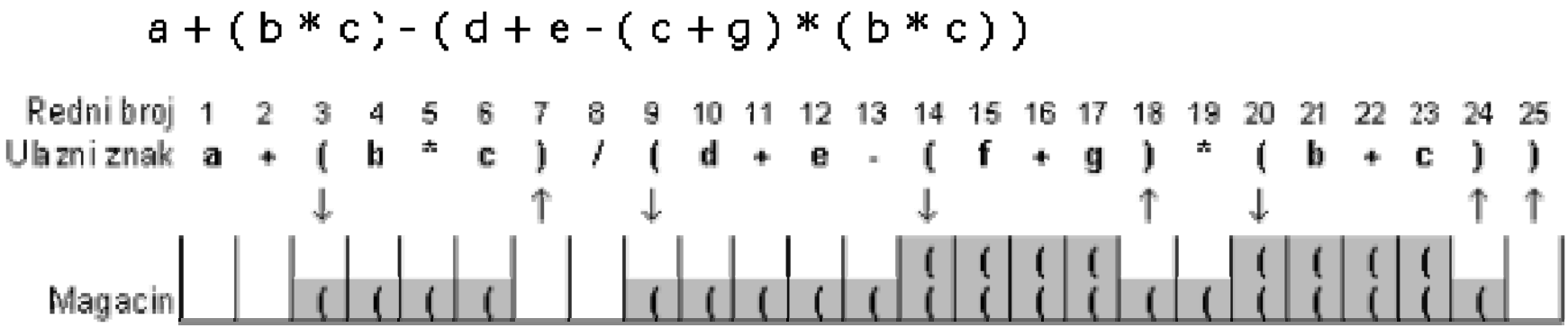


Slika 1.3 Memorijska reprezentacija magacina: a) statička b) dinamička

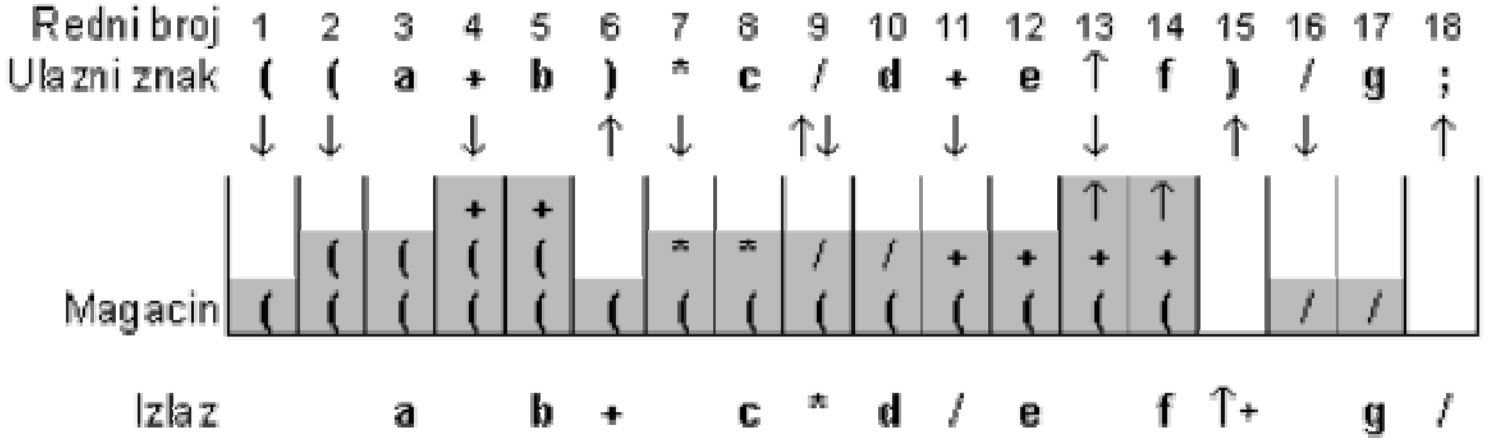
Statička implementacija više magacina koji dele zajednički memorijski prostor



Primena - uparivanje zagrada



Primer - prevođenje aritmetičkih izraza iz infix u postfix notaciju

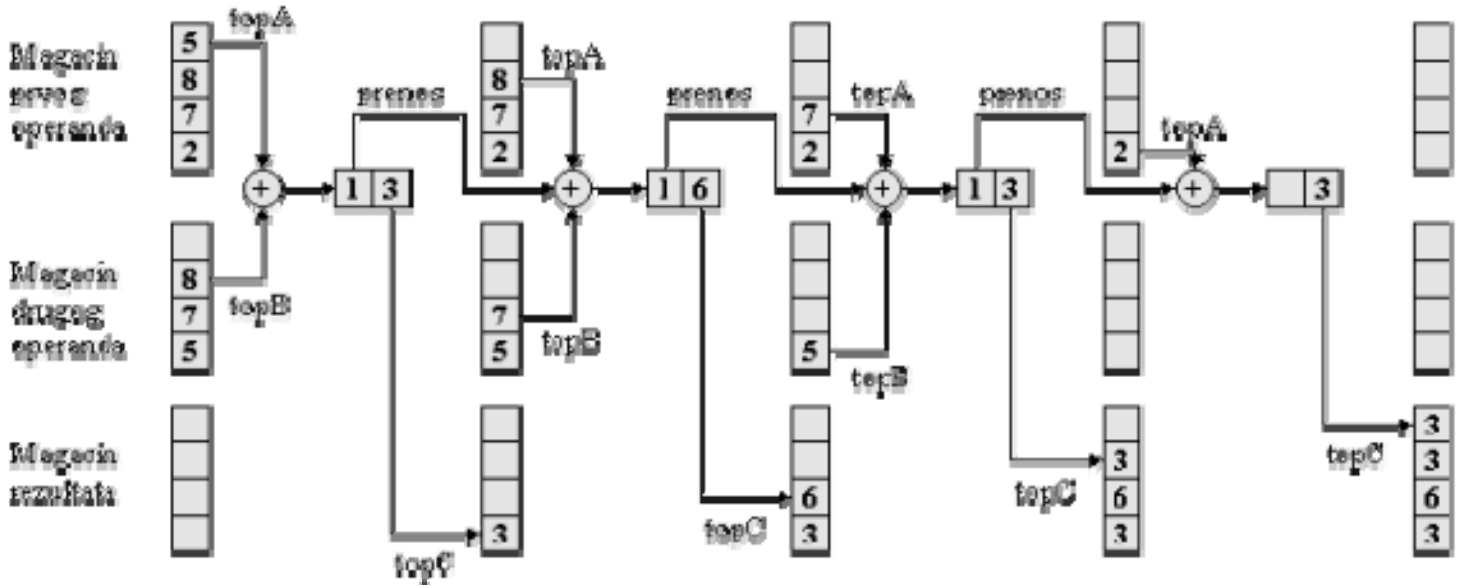


((a + b) * c / d + e ↑ f) / g

a b + c * d / e f ↑ + g /

Primer - sabiranje velikih celih brojeva

npr. 234 524 876 023 227 + 1 243 808 276



Primer sabiranja brojeva 2785 i 578 korišćenjem magacina

Virtuelna klasa magacina

```
template <class T>
class Stack
{
public:
virtual T getTop() {
        throw new SBPException("Virtual function call!");}
virtual void push(T object) {
        throw new SBPException("Virtual function call!");}
virtual T pop() {throw new SBPException("Virtual function call!");}
virtual bool isEmpty() {return true;}
virtual long numberOfElements() {return 0;}
};
```

Statička implementacija magacina

```
template <class T>
class StackAsArray : public Stack<T>
{
protected:
        T* array;       // polje elemenata
        long size;      // veličina polja
        long top;       // indeks vršnog elementa
public:
StackAsArray(long nsize){
                size = nsize;
                array = new T[size];
                top = -1;
};
bool isEmpty () { return (top == -1); }
long numberOfElements () { return (top + 1); }
~StackAsArray() { delete [] array; }
```

Statička implementacija magacina

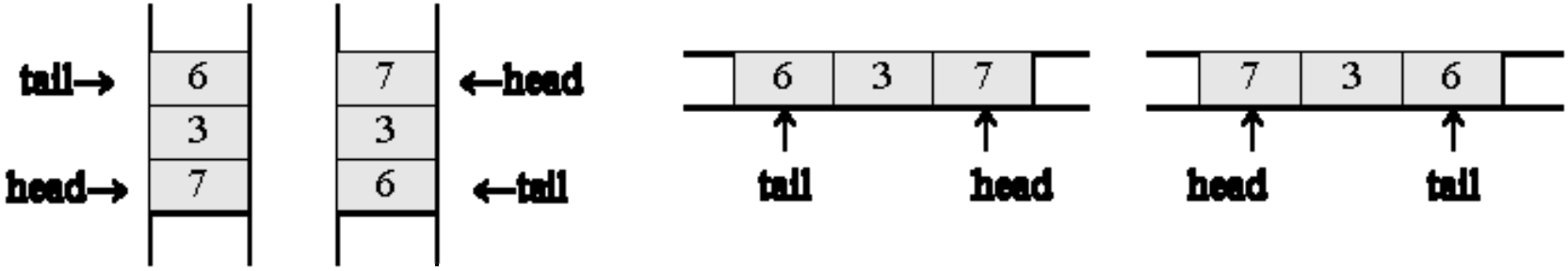
```
void push(T object){
        if (top == (size - 1) )
                throw new SBPException("Stack overflow!");
        array [++top] = object;
}

T pop(){
        if (top == -1)
                throw new SBPException("Stack underflow!");
        T result = array [top--];
        return result;
}

T getTop(){
        if (top == -1)
                throw new SBPException("Stack underflow!");
        return array [top];
}
```

Red

- *Red (engl. queue) je linearna struktura podataka koja omogućuje pristup samo najranije dodatom podatku, odnosno podatak prvi dodat u red se prvi i čita iz reda. Red radi po principu FIFO (first-in, first-out).*

Organizacija reda

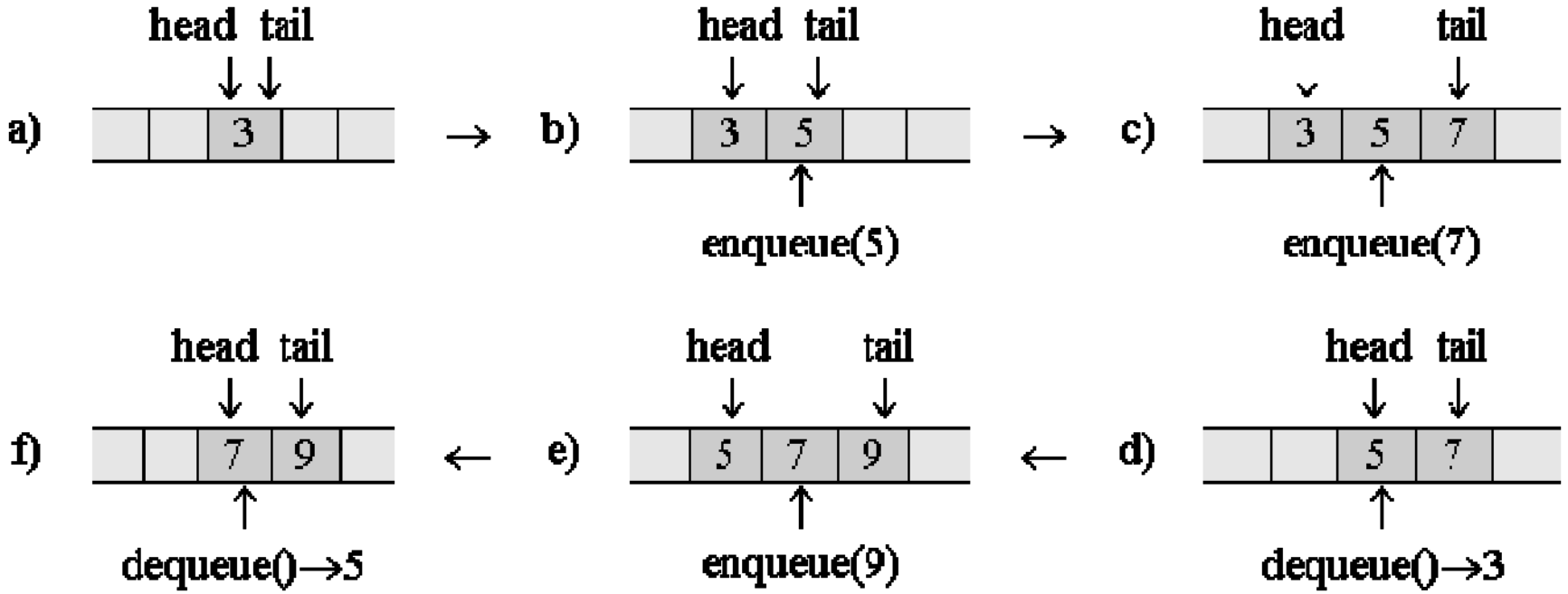
Red je specijalni slučaj linearne liste kod koje se novi element dodaje na jednom kraju liste, koji se naziva **kraj** ili **rep** (engl. *tail*) liste, a briše sa drugog kraja liste, koji se naziva **početak** ili **glava** (engl. *head*) liste.



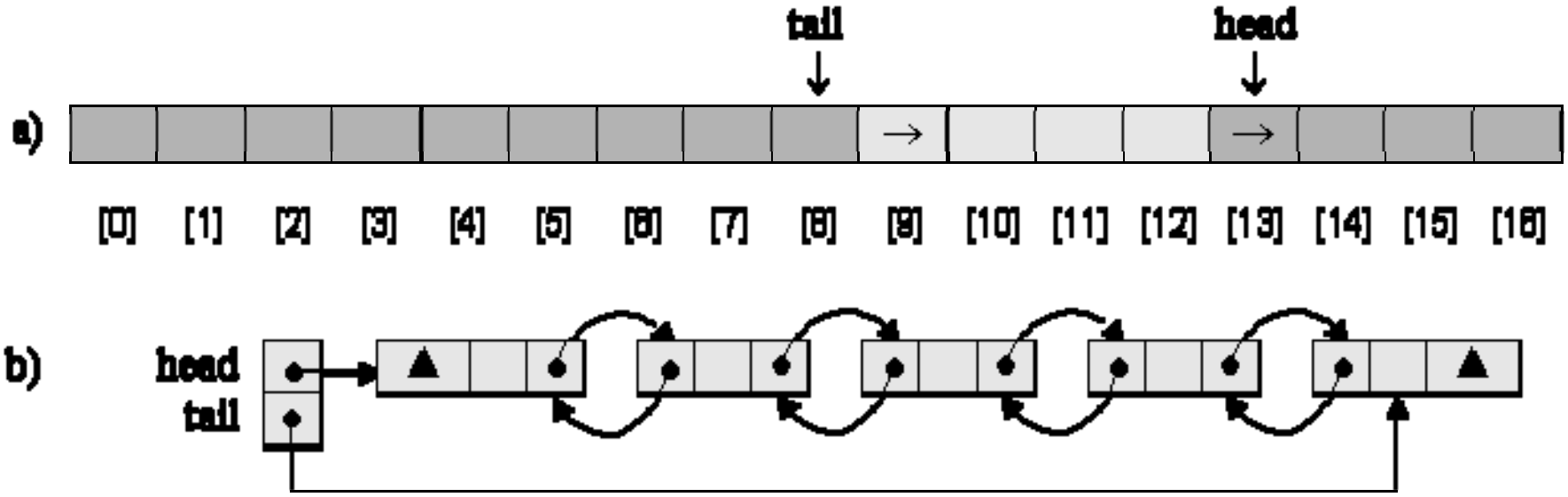
Osnovne operacije

- **enqueue** - dodaje element na kraj reda (*OverflowException*),
- **dequeue** - čita i uklanjanja element sa početka reda (*UnderflowException*),
- **getHead** - čita element sa početka reda, ali ga ne uklanjanja (*UnderflowException*),
- **isEmpty** - proverava da li je red prazan i
- **numberOfElements** - vraća broj elemenata smeštenih u redu.

Primer dodavanja i brisanja elemenata



Memorijska reprezentacija



Virtuelna klasa reda

```
template <class T>
class Queue
{
public:
virtual T getHead() {
        throw  new SBPException("Virtual function call!");}
virtual void enqueue(T object){
        throw new SBPException("Virtual function call!");}
virtual T dequeue() {
        throw  new SBPException("Virtual function call!");}
virtual bool isEmpty() {
        throw  new SBPException("Virtual function call!");}
virtual long numberOfElements(){
        throw new SBPException("Virtual function call!");}
};
```

Statička implementacija reda

```
template <class T>
class QueueAsArray : public Queue<T>
{
protected:
    T* array;                       // polje elemenata
    long size;                      // veličina polja
    long head;                      // indeks početnog elementa
    long tail;                      // indeks krajnjeg elementa
    long numOfElements;             // broj elemenata u redu

public:
   QueueAsArray(long nsize){
                    size = nsize;
                    array = new T[size];
                    head = tail = -1;
                    numOfElements = 0;
    }
    bool isEmpty() { return (numOfElements == 0); }
    long numberOfElements() { return numOfElements; }
    ~QueueAsArray(){ delete [] array; }
```

Statička implementacija reda

```
T getHead(){
                if (numOfElements == 0)
                        throw new SBPException("Queue underflow!");
                return array [head];
}
void enqueue(T object){
                if (numOfElements == size)
                        throw new SBPException("Queue overflow!");
                if (++tail == size) tail = 0;
                array [tail] = object;
                if(numOfElements == 0) head = tail;
                numOfElements++;
}
T dequeue(){
                if (numOfElements == 0)
                        throw new SBPException("Queue underflow!");
                T result = array [head];
                if (++head == size) head = 0;
                numOfElements--;
                if(numOfElements == 0) head = tail = -1;
                return result;
}
```

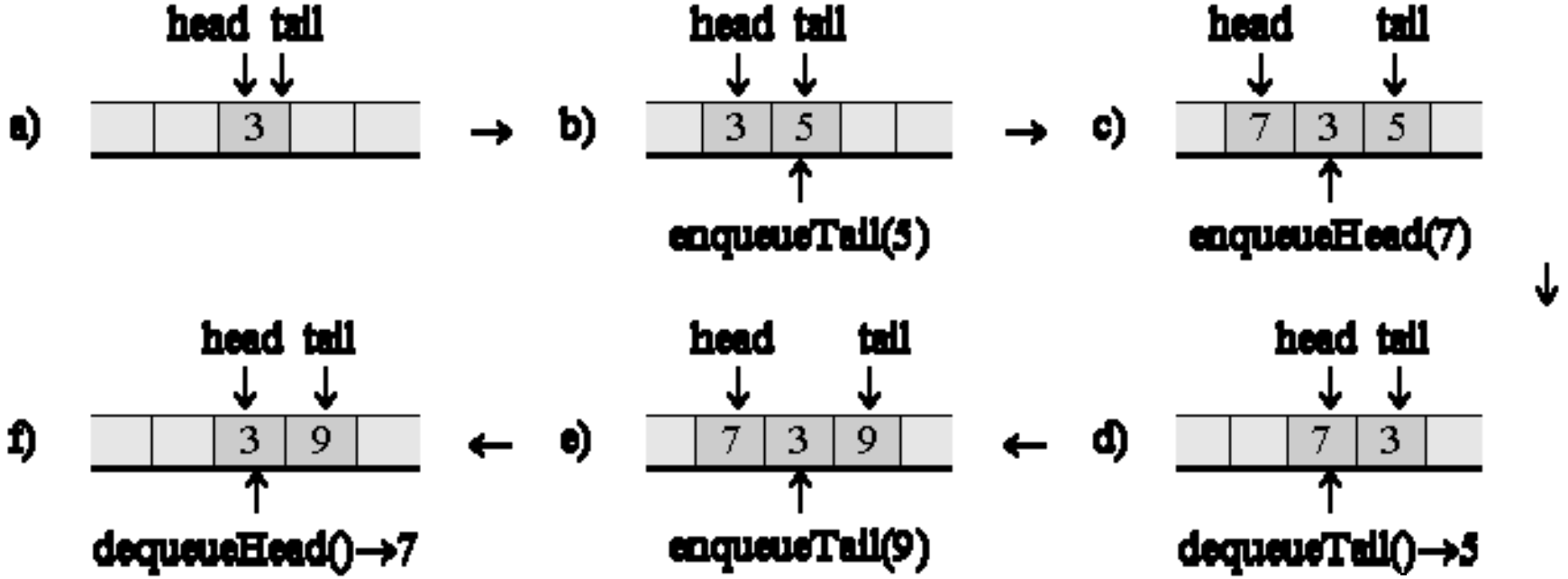
Dvostrani red

- *Dvostrani red (engl. deque) je linearna struktura podataka koja omogućuje dodavanje i brisanje podataka sa oba kraja.*

Osnovne operacije

- **enqueueHead** - dodaje element na početak dvostranog reda (*OverflowException*),
- **enqueueTail** - dodaje element na kraj dvostranog reda (*OverflowException*),
- **dequeueHead** - čita i uklanja element sa početka dvostranog reda (*UnderflowException*),
- **dequeueTail** - čita i uklanja element sa kraja dvostranog reda (*UnderflowException*),
- **getHead** - čita element sa početka dvostranog reda, ali ga ne uklanja (*UnderflowException*),
- **getTail** - čita element sa kraja dvostranog reda, ali ga ne uklanja (*UnderflowException*),
- **isEmpty** - proverava da li je dvostrani red prazan i
- **numberOfElements** - vraća broj elemenata smeštenih u dvostranom redu.

Primer dodavanja i brisanja elemenata



Virtuelna klasa dvostranog reda

```
template <class T>
class Deque
{
public:
virtual T getHead() {
        throw new SBPException("Virtual function call!");}
virtual T getTail() {
        throw new SBPException("Virtual function call!");}
virtual void enqueueHead(T object){
        throw new SBPException("Virtual function call!");}
virtual void enqueueTail(T object){
        throw new SBPException("Virtual function call!");}
virtual T dequeueHead() {
        throw new SBPException("Virtual function call!");}
virtual T dequeueTail() {
        throw new SBPException("Virtual function call!");}
virtual bool isEmpty() {return true;}
virtual long numberOfElements() {return 0;}
};
```

Statička implementacija dvostranog reda

```
template <class T>
class DequeAsArray : public QueueAsArray<T>, Deque<T>
{
public:
        DequeAsArray(long nsize) : QueueAsArray<T>(nsize){ };

        void enqueueHead(T object)
        {
                if (numOfElements == size)
                        throw new SBPException("Deque overflow!");
                if (numOfElements == 0)
                        head = tail = 0;
                else if (head-- == 0)
                        head = size - 1;
                array [head] = object;
                ++numOfElements;
};

T dequeueHead() { return dequeue(); }
```

Statička mplementacija dvostranog reda

```
T getTail(){
                if (numOfElements == 0)
                        throw new SBPException("Deque underflow!");
                return array[tail];
}

void enqueueTail(T object) { enqueue (object); }

T dequeueTail(){
                if (numOfElements == 0)
                        throw new SBPException("Deque underflow!");
                T result = array [tail];
                if (tail-- == 0)
                        tail = size - 1;
                --numOfElements;
                return result;
}
```

Zadaci za proveru prihvatanja gradiva

- Projektovati klasu SStack i implementirati funkciju za prevođenje aritmetičkih izraza iz infix u postfix notaciju. Izraz se zadaje u obliku stringa (niza karaktera) i može sadržati zagrade i blanko znake. Smatrati da postoje sledeći operatori: +, -, * i /, a da se operandi zadaju jednim slovom alfabeta.
- Predložiti implementaciju (tj. pobrojati potrebne atribute klase i nacrtati algoritam realokacije) *n* magacina koji dele zajednički memorijski prostor veličine *m* memorijskih lokacija, tako da ukoliko dodje do prekoračenja u nekom od magacina treba preraspodeliti slobodni prostor M, tako da:
  - αM lokacija bude dodeljeno na osnovu trenutne veličine magacina,
  - βM lokacija bude dodeljeno proporcionalno porastu magacina od prethodne realokacije i
  - γM lokacija dodeliti ravnopravno svim magacinima.

  Parametri α, β i γ su konstante koje se definišu unutar klase koja implementira n-tostruki magacin.