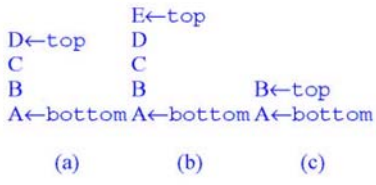
Στοίβες - Ουρές

Γιάννης Θεοδωρίδης, Νίκος Πελέκης, Άγγελος Πικράκης
Τμήμα Πληροφορικής

Στοίβα (stack)

- Δομή τύπου LIFO: Last In - First Out (τελευταία εισαγωγή – πρώτη εξαγωγή)
- Περιορισμένος τύπος γραμμικής λίστας: Εισαγωγή και διαγραφή μόνο στο ένα άκρο της λίστας (στην αρχή)



AbstractDataType *Stack* {
  instances
    ordered list of elements; one end is called the *bottom*; the other is the *top*;
  operations
    Create (): create an empty stack;
    IsEmpty (): return `true` if stack is empty, return `false` otherwise;
    Top (): return top element of stack;
    Add (*x*): add element *x* to the stack;
    Delete (*x*): delete top element from stack and put it in *x*;
}

Κλάση Stack βασισμένη σε τύπο (υλοποίηση με πίνακα)

```
class Stack {
// LIFO objects
   public:
      Stack(int MaxStackSize = 10);
      ~Stack() {delete [] stack;}
      bool IsEmpty() const {return top == -1;}
      bool IsFull() const {return top == MaxTop;}
      T Top() const;
      Stack<T>& Add(const T& x);
      Stack<T>& Delete(T& x);
   private:
      int top;    // current top of stack
      int MaxTop; // max value for top
      T *stack;   // element array
}
```


Συνάρτηση κατασκευής

```
Stack<T>::Stack(int MaxStackSize)
{// Stack constructor.
   MaxTop = MaxStackSize - 1;
   stack = new T[MaxStackSize];
   top = -1;
}
```

Στοιχείο κορυφής (top)

```
T Stack<T>::Top() const
{// Return top element.
   if (IsEmpty()) throw OutOfBounds();
      // Top fails
   else return stack[top];
}
```

Προσθήκη στη στοίβα (add)

```
Stack<T>& Stack<T>::Add(const T& x)
{// Add x to stack.
   if (IsFull()) throw NoMem(); // add fails
   stack[++top] = x;
   return *this;
}
```

Διαγραφή από τη στοίβα (delete)

```
Stack<T>& Stack<T>::Delete(T& x)
{// Delete top element and put in x.
   if (IsEmpty()) throw OutOfBounds(); // delete fails
   x = stack[top--];
   return *this;
}
```


Συνδεδεμένη στοίβα (υλοποίηση με αλυσίδα)

```
class LinkedStack {
   public:
      LinkedStack() {top = 0;}
      ~LinkedStack();
      bool IsEmpty() const {return top == 0;}
      bool IsFull() const;
      T Top() const;
      LinkedStack<T>& Add(const T& x);
      LinkedStack<T>& Delete(T& x);
   private:
      Node<T> *top; // pointer to top node
}
```

Κορυφαίο στοιχείο (top)

```
T LinkedStack<T>::Top() const
{// Return top element.
   if (IsEmpty()) throw OutOfBounds();
   return top->data;
}
```

Προσθήκη στη στοίβα (add)

```
LinkedStack<T>& LinkedStack<T>::Add(const T& x)
{// Add x to stack.
   Node<T> *p = new Node<T>;
   p->data = x; p->link = top; top = p;
   return *this;
}
```


Διαγραφή από τη στοίβα (delete)

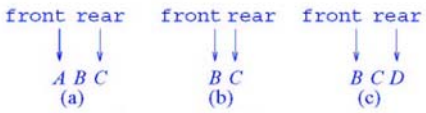
```
LinkedStack<T>& LinkedStack<T>::Delete(T& x)
{// Delete top element and put it in x.
   if (IsEmpty()) throw OutOfBounds();
   x = top->data; Node<T> *p = top; top = top->link;
   delete p; return *this;
}
```

Συνάρτηση καταστροφής

```
LinkedStack<T>::~LinkedStack()
{// Stack destructor..
   Node<T> *next;
   while (top) {
      next = top->link; delete top; top = next;
      }
}
```

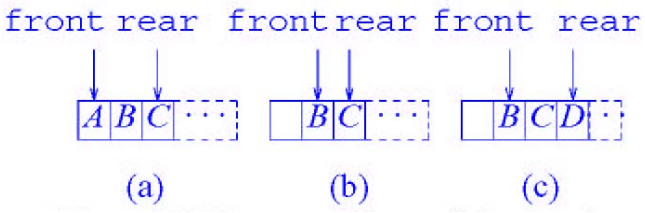
Ουρά (queue)

- Δομή τύπου FIFO:
  First In - First Out
  (πρώτη εισαγωγή – πρώτη εξαγωγή)
- Περιορισμένος τύπος γραμμικής λίστας: Εισαγωγή στο ένα άκρο (στο τέλος) και διαγραφή από το άλλο (την αρχή)



AbstractDataType *Queue* {
instances
ordered list of elements; one end is called the *front*; the other is the *rear*;
operations
Create (): create an empty queue;
IsEmpty (): return `true` if queue is empty, return `false` otherwise;
First (): return first (front) element of queue;
Last (): return last (rear) element of queue;
Add (*x*): add element *x* to the queue;
Delete (*x*): delete front element from queue and put it in *x*;
}

Υλοποίηση ουράς με πίνακα

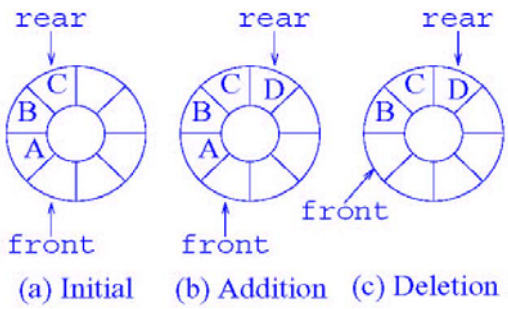


`location(i)=front+i-1`

Ερωτήσεις:

- Γιατί να μη ξεκινά η ουρά πάντα από την αρχή του πίνακα;
- Τι πρόβλημα δημιουργείται με την 'ολίσθηση' της ουράς προς τα δεξιά;

Υλοποίηση κυκλικής ουράς (με πίνακα)

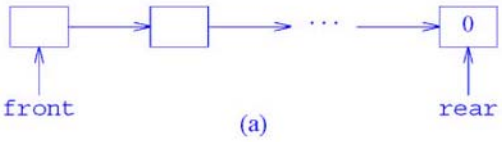


`location(i)=(front+i)%MaxSize`

Ερωτήσεις:

- Πού δείχνουν οι δείκτες **`front`** και **`rear`**;
- Πώς διακρίνουμε μεταξύ μιας άδειας και μιας γεμάτης ουράς;

Συνδεδεμένη ουρά




```
class LinkedQueue {
// FIFO objects
   public:
      LinkedQueue()
          {front = rear = 0;}; // constructor
      ~LinkedQueue(); // destructor
      int IsEmpty() {return ((front) ? 0 : 1);}
      int IsFull();
      int First(type& x); // return first element of queue
      int Last(type& x);  // return last element of queue
      int operator +(type x); // add x to queue
      int operator -(type& x); // delete x from queue
          // First,+,- return 0 on failure, 1 on success
   private:
      Node<type> *front, *rear;
}
```

Προσθήκη (ως τελευταίο στοιχείο) σε συνδεδεμένη ουρά

```
int LinkedQueue<type>::operator+(type x)
//add x to queue
{
Node<type> *i;
i = new Node<type>;
if (i) {
     i->data = x; i->link = 0;
     if (front) rear->link = i;
     else front = i;
     rear = i; return 1;
     };
return 0;
   // add fails
}
```

Διαγραφή (του πρώτου στοιχείου) από συνδεδεμένη ουρά

```
int LinkedQueue<type>::operator-(type& x)
//delete first element and return in x
{
if (IsEmpty()) return 0; //delete fails
x = front->data;
Node<type> *i = front;
front = front->link;
delete i;
return 1;
}
```