http://rain.ifmo.ru/cat/



# ADS-1 <3> Линейные динамические ADT

© С. Е. Столяр, 2007
ses@mail.ifmo.ru

© СПбГУ ИТМО, 2007
Кафедра компьютерных технологий





# Классификация

*Списком* называется конечное множество элементов с заданным на нем линейным порядком.

В отличие от лекции о «машинозависимых» структурах данных, где рассматривались статические массивы, здесь речь пойдет о *динамических структурах*. Все они являются разновидностями списков.

## Линейные динамические структуры данных

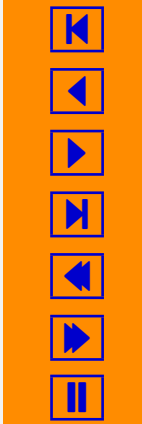
- *Списки* (List)
- *Стеки* (Stack)
- *Очереди* (Queue)
- *Деки* (Deque)



# ADT List: организация, доступ ...

Компонентами (item) множества *Список* (list) являются записи, включающие не менее двух полей. Поле *ключ* (key) идентифицирует элемент списка, поле *указатель* (pointer) — указывает на соседний элемент списка. Остальные (информационные) поля обычно не несут алгоритмической нагрузки.


Каждый элемент «видит» только «соседа» (двух «соседей»), наличие указателей делает список *связным* (linked). Благодаря указателям поддерживается линейный порядок на множестве $L$.





# . . . ADT List: организация, доступ . . .

Наличие линейного порядка на непустом множестве гарантирует существование минимального в заданном порядке элемента, *головы списка* (head).

Доступ к множеству $L$ — через внешний указатель $p$ на головной элемент.

Элемент, не имеющий следующего за ним, называется *хвостом* (tail).

Иногда список создается с постоянным головным элементом, *заголовком* (header).

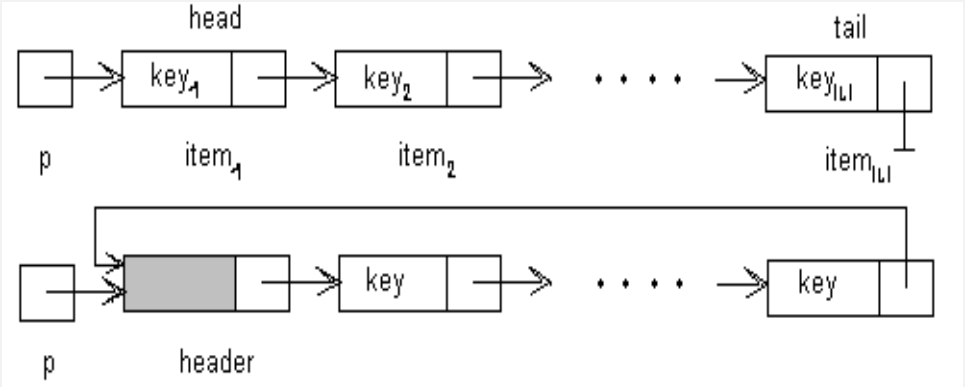
Обычно заголовком оснащают *кольцевой* список — для поддержания линейного порядка.



# . . . ADT List: организация, доступ . . .

## Односвязный список

Односвязный (однонаправленный, односторонний) список (singly linked list) содержит компоненты с одним указателем, который «смотрит» на *следующий* (next) элемент.



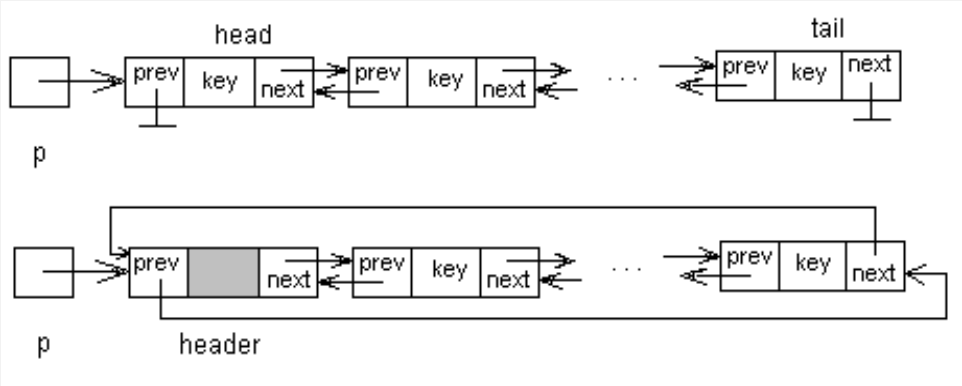

В *кольцевом* списке указатель хвоста «видит» заголовок.

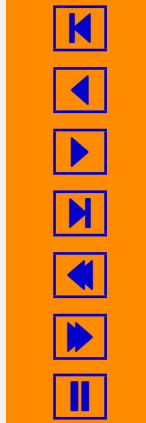


# . . . ADT List: организация, доступ . . .

## Двусвязный список

Компоненты двусвязного (двунаправленного, двустороннего) списка (doubly linked list) включают два указателя — на *следующий* (next) и на *предыдущий* (previous) элементы.

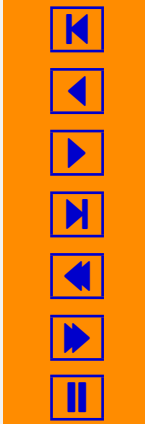




# ADT List: операции ...

- MakeList($L$)
  Функция: создает пустой список, возвращая указатель на него. Трудоемкость $O(1)$.
- ListNewItem($key, q$)
  Конструктор: создает новый элемент с ключом $key$, возвращая указатель на него $q$. Трудоемкость $O(1)$.
- ListInsert($L, q$)
  Процедура: берет элемент по указателю $q$ и вставляет его в список в качестве головного. Трудоемкость $O(1)$.
- ListSearch($L, key$)
  Функция: возвращает указатель на первый из элементов с ключом $key$. Трудоемкость $O(|L|)$.





# . . . ADT List: операции . . .

- ListSearchPredecessor$(L, q)$

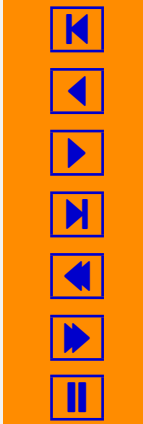
  Функция: возвращает для элемента списка, доступного по указателю $q$, адрес его предшественника. Трудоемкость $O(|L|)$ для односвязного списка и $O(1)$ для двусвязного.

- ListDelete$(L, q)$

  Процедура: удаляет из списка элемент, адресуемый непустым указателем $q$. Трудоемкость $O(|L|)$ для односвязного списка и $O(1)$ для двусвязного.

- ListSearchEnd$(L)$

  Функция: возвращает указатель на последний элемент списка. Трудоемкость $O(|L|)$; для двусвязного кольцевого — $O(1)$.

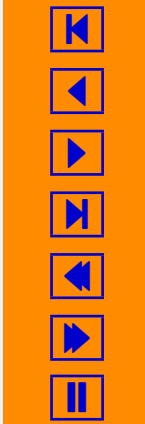




# . . . ADT List: операции

- ListConcatenation$(L, L_1, L_2)$
  Функция: возвращает список $L$, образованный *сцеплением* списков $L_1$ и $L_2$. Трудоемкость $O(|L_1|)$ для односвязного списка и $O(1)$ для двусвязного кольцевого.

- ListMerge$(L, L_1, L_2)$
  Функция: сливает *упорядоченные* списки $L_1$ и $L_2$ в результирующий упорядоченный список $L$; исходные списки перестают существовать. Трудоемкость $O(|L_1| + |L_2|)$.
  Реализация: компоненты списков не переписываются, только переставляются указатели.



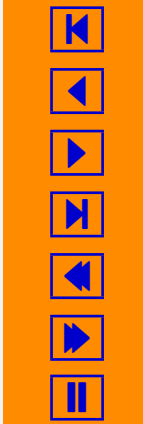


# ADT Stack: организация и операции

*Стеком* называется разновидность списка, где все операции выполняются только на одном его конце, называемом *вершиной* ($top$ или $head$) стека.

Дисциплина обслуживания 'Last In, First Out' (*LIFO*).

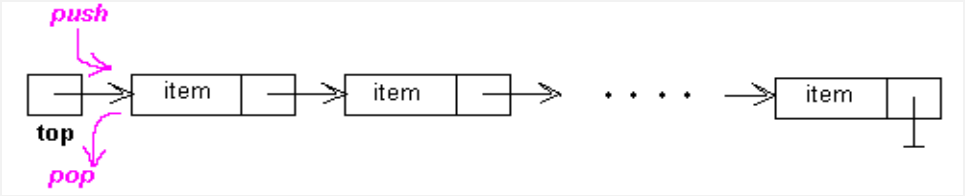
- $\mathrm{Push}(S, item)$
  Процедура: вставляет в стек элемент $item$. Трудоемкость $O(1)$.
- $\mathrm{Pop}(S, item)$
  Процедура: извлекает из непустого стека элемент $item$. Трудоемкость $O(1)$.
- $\mathrm{isStackEmpty}(S)$
  Функция: возвращает булевское значение $\mathrm{true}$ ($\mathrm{false}$) для пустого (непустого) стека. Трудоемкость $O(1)$.



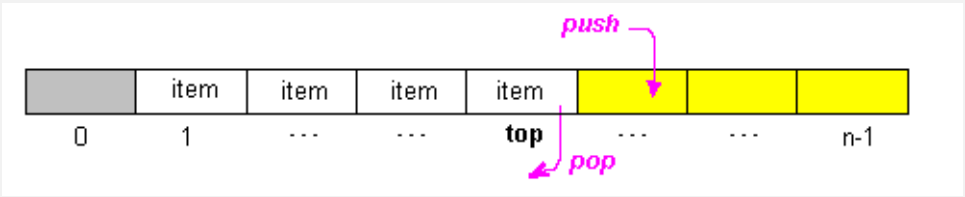


# ADT Stack: реализация

## На основе однонаправленного списка




## На основе кольцевого вектора






# ADT Stack: приложения

- Механизм рекурсии
- Механизмы распределения оперативной памяти
- Синтаксический разбор
- Стековая арифметика
- Стековые языки программирования (Forth, Postscript)
- Обход (поиск) в глубину
- Обработка линейных записей (см. *визуализатор*)
- Алгоритмы на графах
- ...

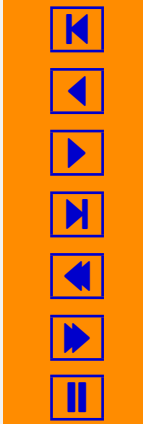


# ADT Queue: организация и операции

*Очередью* называется разновидность списка с двумя поддерживаемыми операциями: вставкой со стороны хвоста и удалением со стороны головы.

Дисциплина обслуживания 'First In, First Out' (*FIFO*).

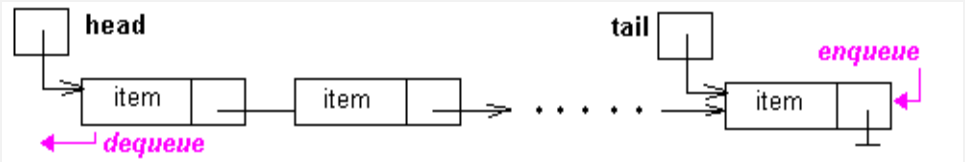
- $\mathrm{Enqueue}(Q, item)$
  Процедура: вставляет в очередь элемент $item$. Трудоемкость $O(1)$.
- $\mathrm{Dequeue}(Q, item)$
  Процедура: извлекает из непустой очереди элемент $item$. Трудоемкость $O(1)$.
- $\mathrm{isQueueEmpty}(S)$
  Функция: возвращает булевское значение $\mathrm{true}$ ($\mathrm{false}$) для пустой (непустой) очереди. Трудоемкость $O(1)$.



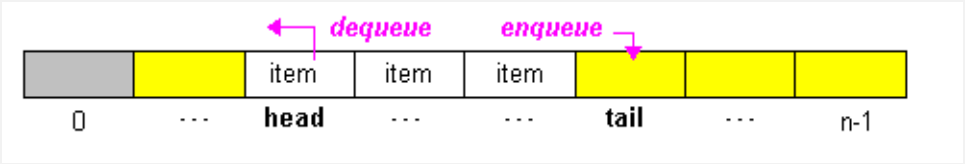


# ADT Queue: реализация

## На основе однонаправленного списка




## На основе кольцевого вектора






# ADT Queue: приложения

- Механизмы распределения оперативной памяти
- Конвейерная обработка
- Обход (поиск) в ширину
- Обработка линейных записей (см. *визуализатор*)
- Алгоритмы на графах
- . . .



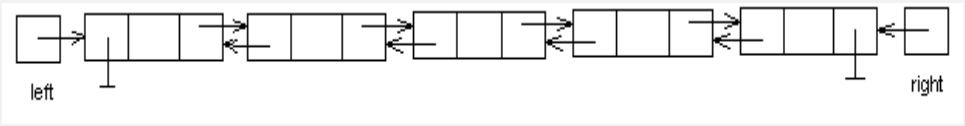






# ADT Deque

*Дек* (двусторонняя очередь; *d*ouble-*e*nded *que*ue) — это линейный двунаправленный список с операциями вставки и удаления, которые разрешены на обоих его концах, левом (left) и правом (right).




## Ограниченные деки

Дек с *ограниченным входом*: операция вставки элемента поддерживается только на одном конце.

Дек с *ограниченным выходом*: операция удаления элемента поддерживается только на одном конце.