

# Структуры данных

# Структурирование данных

- Структурирование данных — разработка структуры данных, состоящая в определении данных как объектов элементарного типа и отношений между ними.
- В математике существует понятие структуры, позволяющее строго описать структуру данных и операции над ними.
- Для различия математических свойств данных и представления их в памяти ЭВМ используются специальные термины :
  - логическая структура данных (или просто структура данных) и
  - физическая структура данных (структура хранения данных, структура представления в памяти).

- Структура данных — множество элементов данных, объединенных и упорядоченных одним из принятых способов. При использовании данных с ЭВМ, данные представляются в памяти машины, на физическом носителе.
- Структура хранения данных — организация и способ размещения элементов логической структуры в запоминающей среде, способ отображения структуры данных в физическую среду.

- Любое представление структуры данных в памяти ЭВМ должно включать в себя как сами данные, так и задаваемые взаимосвязи, которые и определяют структурирование. (Некоторые структуры данных могут быть представлены в памяти ЭВМ различными способами.)
- Представление структуры данных в памяти ЭВМ зависит от предполагаемого использования данных, поскольку для различных типов структур эффективность выполнения тех или иных операций обработки данных различна.

# Основные структуры данных

## Массив

- **Массив** — представляет собой некоторое количество расположенных в определенном порядке элементов одного типа. Индекс предназначен для обеспечения возможности указания на элементы массива. Типичными операциями над данными типа массив являются:
  - Задание начальных значений элементов массива.
  - Выбор элементов массива по заданным значениям индексов.
  - Избирательное обновление массива.

# Последовательности

- **Последовательности** — в отличие от массива, количество элементов (длина) последовательности конечно, но не фиксировано. Это допускает существование последовательности произвольной длины. Основные операции над последовательностями:
  - Создание последовательности
  - Выборка элементов последовательности
  - Включение элементов последовательности
  - Удаление элементов последовательности

# Операции последовательности

Рассмотрим более подробно и формально эти операции.

- **создание:**  $T(l_1, l_2, \dots, l_n)$ ; при  $n = 0$ , последовательность называется пустой.
- **выборка:** если  $x$  последовательность, то ее элементы можно записать следующим образом:
  - $x[i]$  —  $i$  - элемент  $x$
  - $\text{first}(x)$  — начальный элемент  $x$
  - $\text{last}(x)$  — конечный элемент  $x$

- ***удаление:***

- **tail(x)** — последовательность, в которой из **x** исключен начальный элемент.
- **initial(x)** — последовательность, в которой из **x** исключен последний элемент.

- ***включение:***

- appendl(x, l)** — последовательность, в которой добавляется **l** перед **x** (слева).
- **appendr(x, l)** — последовательность, в которой добавляется **l** после **x** (справа).



Если  $x = T(l_1, l_2, \dots, l_n)$ , то

$\text{tail}(x) = T(l_2, \dots, l_n)$

$\text{initial}(x) = T(l_1, l_2, \dots, l_{n-1})$

$\text{appendl}(x, l) = T(l, l_1, l_2, \dots, l_{n-1})$

$\text{appendr}(x, l) = T(l_1, l_2, \dots, l_n, l)$

# ЗАВИСИМОСТИ

- Между указанными функциями существуют следующие зависимости:  
 **$\text{first}(\text{appendl}(x, l)) = l$**   
 **$\text{tail}(\text{appendl}(x, l)) = x$**   
 **$\text{appendl}(\text{tail}(x), \text{first}(x)) = x$ , если  $x \neq T()$**   
 **$\text{last}(\text{appendr}(x, l)) = l$**   
 **$\text{initial}(\text{appendr}(x, l)) = x$**   
 **$\text{appendr}(\text{initial}(x), \text{last}(x)) = x$ , если  $x \neq T()$**

- Определение пустой последовательности

- **empty(x) =**

**true**, если  $x = T()$

{

**false**, если  $x \neq T()$

# Очередь

- Очередью (англ. *queue*) называется структура данных, в которой элементы кладутся в конец, а извлекаются из начала. Таким образом, первым из очереди будет извлечен тот элемент, который будет добавлен раньше других.

- **Очередь** — это последовательность, для которой определены следующие операции
- образование пустой очереди **T()**;
- **last(x)**;
- **initial(x)**;
- **appendl(x, l)**.

- Добавление элемента к очереди осуществляется с ее левого конца с помощью операции **appendl**,
- а извлечение элемента производится с правого конца с помощью оператора **last** и **initial**.
- В связи с этим очередь является памятью типа первым вошел — первым вышел (first in first out, FIFO).
- Очереди используются в случае, когда данные обрабатываются в порядке их поступления или образования.

# Последовательный файл

- **Последовательный файл**, над которым определены следующие 5 операций:
  - формирование пустой последовательности **T()**
  - **first(x)**
  - **tail(x)**
  - **appendr(x, l)**
  - **empty(x)**

- Это последовательность, в которой допускается выборка (доступ) начального элемента последовательности и добавление элемента в конец последовательности. Такие последовательности реализуются на внешней запоминающей среде (ленте) причем их запись возможна только в одном направлении.



# Стек

- **Стек** — последовательность, для которой определены следующие операторы:
- образование пустой последовательности **T()**
- **first(x)** — **top**
- **tail(x)** — **pop**
- **appendl(x, l)** — **push**
- **empty(x)**

- Стек является памятью типа последним вошел — первым вышел (last in first out, LIFO).
- Стек является наиболее широко используемым типом данных и применяется, например, при анализе языковых конструкций.

- Дек?

# Дек

- Деком (англ. *deque* – аббревиатура от *double-ended queue*, двухсторонняя очередь) называется структура данных, в которую можно удалять и добавлять элементы как в начало, так и в конец.

- Структуры данных — это объекты определенного уровня абстракции, для представления которых в памяти ЭВМ можно использовать различные *структуры хранения данных*,
- например, стек можно представить в виде массива, а используя указатели его можно представить в виде списочной структуры. Реализация операции добавления и удаления для стека зависит от выбора структуры хранения данных.

- Таким образом системы хранения и манипулирования данными взаимосвязаны и должны рассматриваться совместно. Другими словами, физическое представление данных определяется в единстве с основными операциями манипулирования данными, и использование данных определенных типов осуществляется только через посредство соответствующих процедур.

# АТД

- Подобный аспект рассмотрения данных связывают с понятием **абстрактных типов данных (АТД)**. (Б.Лисков, 1974)
- Если точно определить внешние спецификации АТД, указав аргументы, процедуры (операции) и результат, то можно использовать этот тип данных, не определяя внутреннюю организацию данных.
- Существуют различные методы абстрагирования данных. Существуют методы формальной спецификации АТД (алгебраическая, аксиоматическая)
- Класс - это абстрактный тип данных, снабженный некоторой (возможно частичной) реализацией

- Для описания АТД используется формат, который включает заголовок с именем АТД, описание типа данных и список операций. Формат АТД служит не только основой для проектирования программной системы. Не менее важен формат и для точного описания интерфейса, предоставляемого пользователю - программисту для управления структурой данных. Поэтому формулировки формата АТД должны быть исчерпывающими, точными и краткими.



- В описании формата АТД должны использоваться, понятия, объекты и переменные, использующиеся только на уровне интерфейса к структуре данных. На уровень интерфейса не должны выноситься детали внутренней реализации структуры данных и операций управления ею.

- Операция обозначает обслуживание, которое объект предлагает своим клиентам. Возможны пять видов операций клиента над объектом:
  - 1) модификатор (изменяет состояние объекта);
  - 2) селектор (дает доступ к состоянию, но не изменяет его);
  - 3) итератор (доступ к содержанию объекта по частям, в строго определенном порядке);
  - 4) конструктор (создает объект и инициализирует его состояние);
  - 5) деструктор (разрушает объект и освобождает занимаемую им память)

- Типы данных впервые были описаны Д. Кнудом в его книге «Искусство программирования» [3]. В главе 2, «Информационные структуры», Кнут описывает т.н. *структуры данных*, определяемые как способы организации данных внутри программы. Кнут описывает такие типы данных, как списки, деревья, стеки, очереди, деки и т.д.

Рассмотрим, например, как Кнут описывает тип данных «стек».

- Кроме собственно описания самой структуры данных, Кнут описывает «алгоритмы обработки» этой структуры с помощью словаря специальных *терминов*. Для стека этот словарь содержит термины: *push* (втолкнуть), *pop* (вытолкнуть) и *top* (верхний элемент стека). Таким образом, типы данных описываются Кнудом с помощью специального языка, задающего определенную терминологию и толкование этой терминологии.

# АТД СТЕК операции

- create:  $\rightarrow \text{Stack}[\text{Elem}]$
- push:  $\text{Stack}[\text{Elem}] \times \text{Elem} \rightarrow \text{Stack}[\text{Elem}]$
- pop:  $\text{Stack}[\text{Elem}] \rightarrow \text{Elem}$
- top:  $\text{Stack}[\text{Elem}] \rightarrow \text{Elem}$
- *erasetop:  $\text{Stack}[\text{Elem}] \rightarrow \text{Stack}[\text{Elem}]$*
- empty:  $\text{Stack}[\text{Elem}] \rightarrow \text{Bool}$

# АТД СТЕК аксиомы и предусловия

- $\text{top}(\text{push}(s,x))=x$
- $\text{pop}(\text{push}(s,x))=x$
- $\text{empty}(\text{create}())=\text{true}$
- $\text{not empty}(\text{push}(s,x))=\text{true}$
  
- $\text{pop}(s)$  require  $\text{not empty}(s)$
- $\text{top}(s)$  require  $\text{not empty}(s)$

- **Спецификация стеков как АД (Б.Мейер)**
- ТИПЫ (TYPES)
- STACK [G]
- ФУНКЦИИ (FUNCTIONS)
- $\text{put: STACK [G] } \times \text{ G } \rightarrow \text{ STACK [G]}$
- 
- $\text{empty: STACK [G] } \rightarrow \text{ BOOLEAN}$
- $\text{new: STACK [G]}$
- АКСИОМЫ (AXIOMS)
- Для всех  $x: G, s: \text{STACK [G]}$
- (A1)  $\text{item (put (s, x))} = x$
- (A2)  $\text{remove (put (s, x))} = s$
- (A3)  $\text{empty (new)}$
- (A4)  $\text{not empty (put (s, x))}$
- ПРЕДУСЛОВИЯ (PRECONDITIONS)
- $\text{remove (s: STACK [G])}$  require not empty (s)
- $\text{item (s: STACK [G])}$  require not empty (s)

- Типы данных впервые были описаны Д. Кнудом в его книге «Искусство программирования» [3]. В главе 2, «Информационные структуры», Кнут описывает т.н. *структуры данных*, определяемые как способы организации данных внутри программы. Кнут описывает такие типы данных, как списки, деревья, стеки, очереди, деки и т.д.

Рассмотрим, например, как Кнут описывает тип данных «стек».

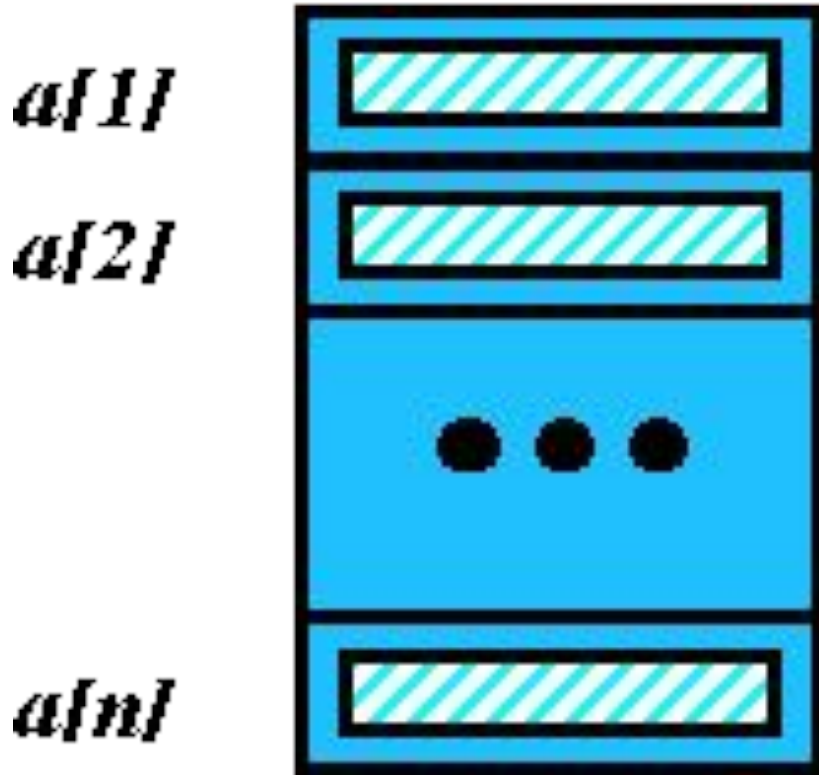
- Кроме собственно описания самой структуры данных, Кнут описывает «алгоритмы обработки» этой структуры с помощью словаря специальных *терминов*. Для стека этот словарь содержит термины: *push* (втолкнуть), *pop* (вытолкнуть) и *top* (верхний элемент стека). Таким образом, типы данных описываются Кнудом с помощью специального языка, задающего определенную терминологию и толкование этой терминологии.

- АТД имя.  
Общая характеристика типа данных
- **ДАнные:**
  - Описание общих параметров.
  - Описание структуры хранения данных.
- **ОПЕРАЦИИ:**
  - Конструктор:*
    - Вход: данные от пользователя.
    - Начальные значения: данные для инициализации.
    - Процесс: инициализация данных.
    - Постусловие: состояние структуры и параметров после инициализации.
  - Операция:*
    - Вход: данные от пользователя.
    - Предусловия: необходимое состояние структуры данных перед выполнением операции.
    - Процесс: действия, выполняемые над данными.
    - Выход: данные, возвращаемые пользователю.
    - Постусловие: состояние структуры после выполнения операции.
  - Операция:*



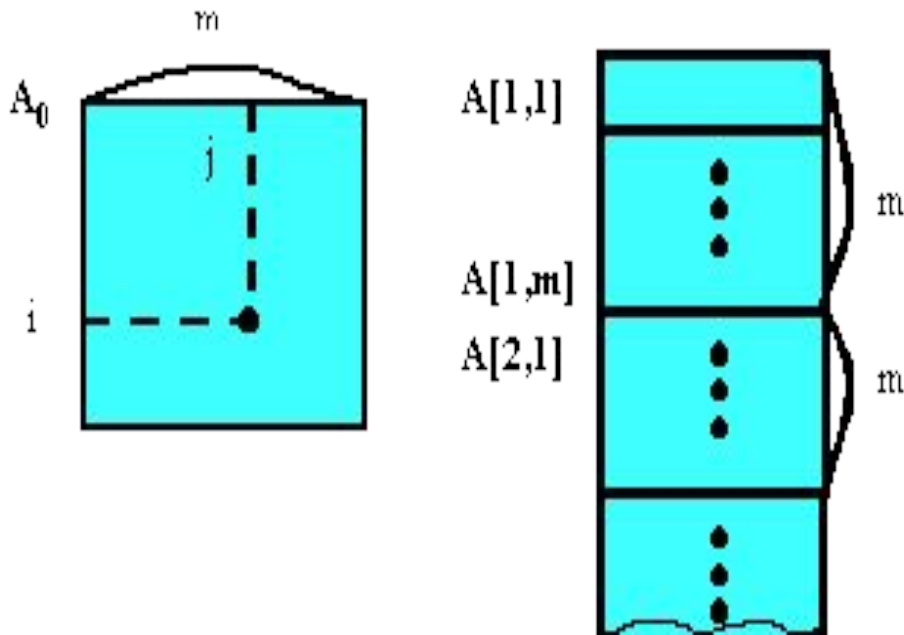
# Структуры хранения данных

# Одномерный массив



- Этот способ является обычным при вводе данных в последовательность адресов запоминающего устройства.
- В случае когда данные представляют собой упорядоченное множество, хранение данных лучше осуществлять в соответствии с их порядком.

# Двумерный массив



- Этот способ организации данных широко применяется на практике.
- Если длина ( $m$ ) строки или столбца известна, то для выборки элемента необходимо определить место его расположения в памяти по формуле  $A[i, j] = A_0 + (i - 1) * m + j$

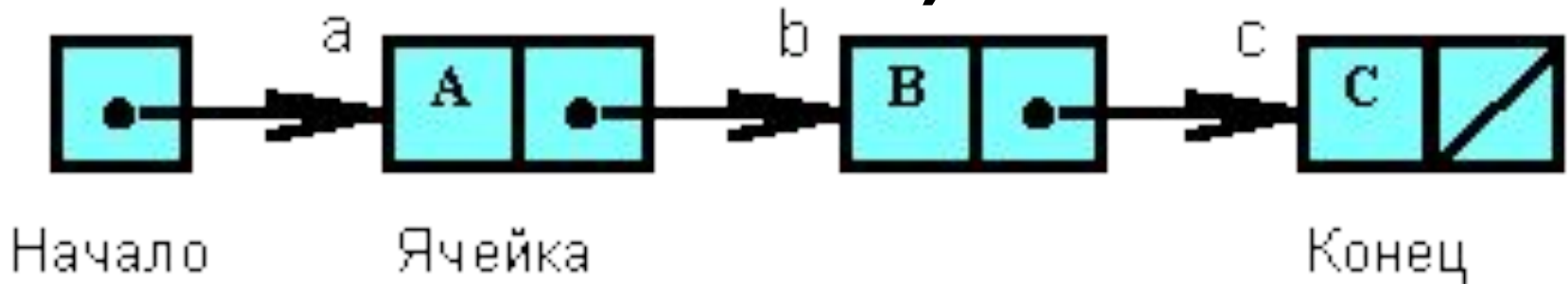
# ***Динамические структуры данных Список (Линейный связанный)***

- Предполагается, что данные записываются не в последовательные адреса памяти, как в случае массивов, а в произвольные места. Для того чтобы задать связь между данными, применяют указатели. Наименьшим структурным элементом в этом случае является ячейка.

- Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.
- Списочная структура (или линейная структура, связанный список) представляет собой структуру данных в виде нескольких линейно связанных ячеек.

- Линейные связные списки являются простейшими динамическими структурами данных. Из всего многообразия связанных списков можно выделить следующие основные:
  - *однонаправленные (односвязные) списки;*
  - *двунаправленные (двусвязные) списки;*
  - *циклические (кольцевые) списки.*

# Односвязный список (Однонаправленный связный список)



- Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа.

- **Однонаправленный (односвязный) список** – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент пустой (равен NULL).
- Добавление и исключение данных можно выполнить с помощью простой операции изменения значения указателя.



# Описание простейшего элемента списка выглядит следующим образом:

```
struct имя_типа { информационное поле;  
    адресное поле; };
```

- где информационное поле – это поле любого, ранее объявленного или стандартного, типа;
- адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

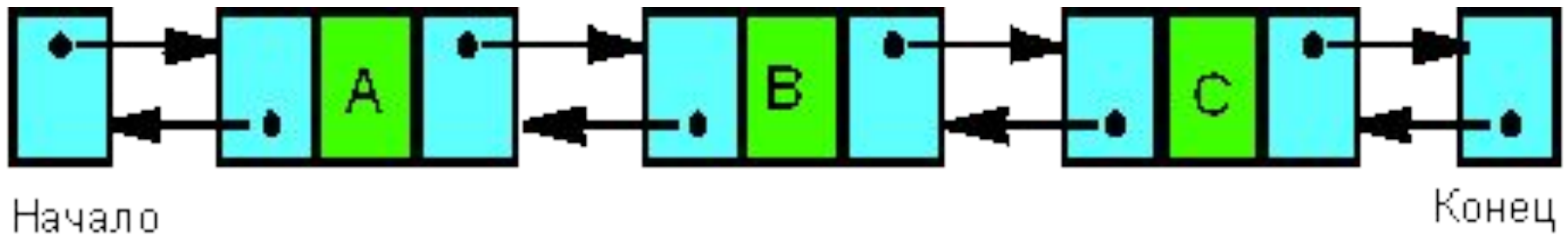
Например:

```
struct Node { int key;//информационное поле  
    Node*next;//адресное поле };
```

- Особое внимание следует обратить на то, что при выполнении любых операций с линейным однонаправленным списком необходимо обеспечивать позиционирование какого-либо указателя на первый элемент(голова списка). В противном случае часть или весь список будет недоступен.
- В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.
- Для сортировки динамических структур данных удобно использовать алгоритм обменной (пузырьковой) сортировки. Причем, в этом случае, элементы не переставляются местами, а просто меняются их адресные части.

# Двунаправленные (двусвязные) списки

- Для ускорения многих операций целесообразно применять переходы между элементами списка в обоих направлениях. Это реализуется с помощью двунаправленных списков, которые являются сложной динамической структурой.
- **Двунаправленный (двусвязный) список** – это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы. При этом два соседних элемента должны содержать взаимные ссылки друг на друга.



- Достоинство: каждый элемент списка доступен из каждого текущего элемента.
- Недостаток связан с необходимостью выделения места для обратного указателя.
- Так как элемент двунаправленного списка имеет два указателя, то при выполнении операций включения/исключения элемента надо изменять больше связей, чем в однонаправленном списке.

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа { информационное поле;  
                    адресное поле 1;  
                    адресное поле 2; };
```

где информационное поле – это поле любого, ранее объявленного или стандартного, типа;

адресное поле 1 – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка ;

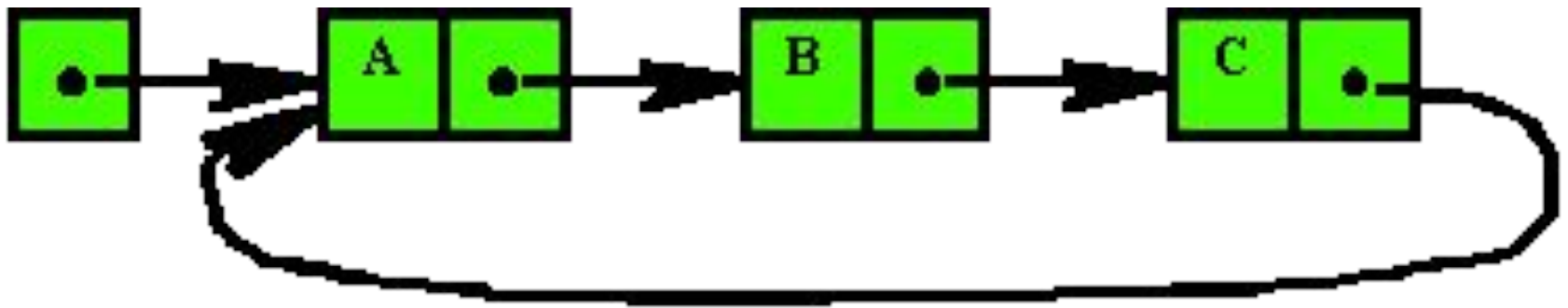
адресное поле 2 – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес предыдущего элемента списка.

# Например:

- ***struct list {***
  - type elem ;***
  - list \*next, \*pred ;***
  - }***
- ***list \*headlist ;***
- где *type* – тип информационного поля элемента списка;
- *\*next, \*pred* – указатели на следующий и предыдущий элементы этой структуры соответственно.
- Переменная-указатель *headlist* задает список как единый программный объект, ее значение – указатель на первый (или заглавный) элемент списка.

- Особое внимание следует обратить на то, что в отличие от однонаправленного списка здесь нет необходимости обеспечивать позиционирование какого-либо указателя именно на первый элемент списка, так как благодаря двум указателям в элементах можно получить доступ к любому элементу списка из любого другого элемента, осуществляя переходы в прямом или обратном направлении.
- Однако по правилам хорошего тона программирования указатель желательно ставить на заголовок списка.

# Циклические (кольцевые, замкнутые) списки



- Односвязный список, у которого конечный элемент ссылается на начальный элемент, называется циклическим списком.
- В этом списке константы NULL не существует.
- Он тоже может быть односвязным или двусвязным. Последний элемент кольцевого списка содержит указатель на первый, а первый (в случае двусвязного списка) — на последний.



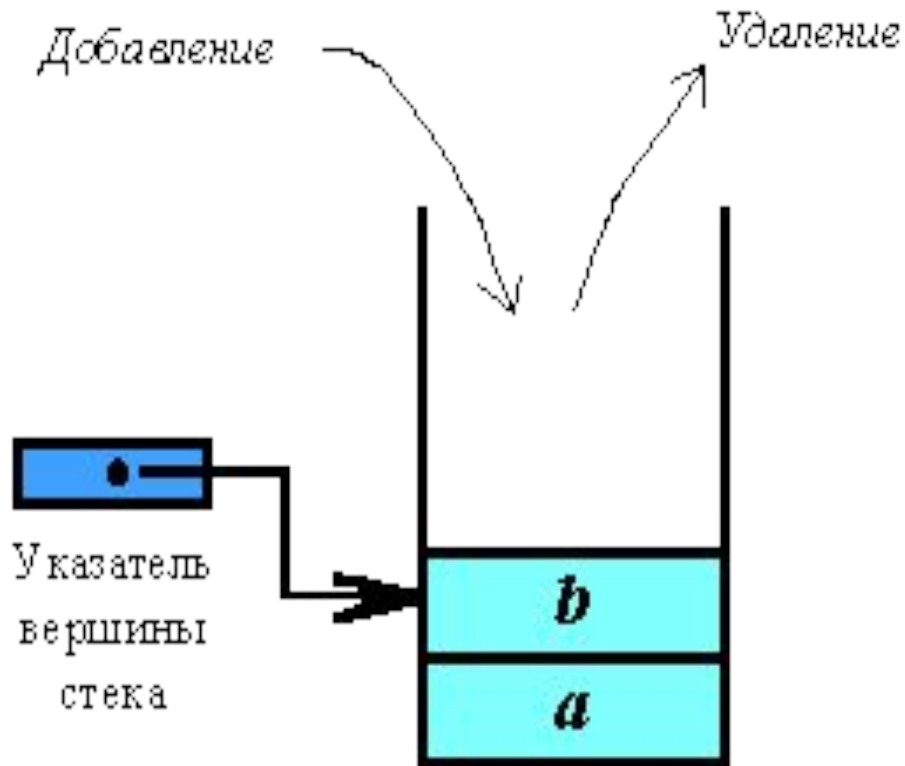
# Списки, достоинства

- лёгкость добавления и удаления элементов
- размер ограничен только объёмом памяти компьютера и разрядностью указателей
- динамическое добавление и удаление элементов

# Списки, недостатки

- сложность определения адреса элемента по его индексу (номеру) в списке
- на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны)
- работа со списком медленнее, чем с массивами, так как к любому элементу списка можно обратиться, только пройдя все предшествующие ему элементы
- элементы списка могут быть расположены в памяти разреженно, что окажет негативный эффект на кэширование процессора
- над связными списками гораздо труднее (хотя и в принципе возможно) производить параллельные векторные операции, такие как вычисление суммы
- кэш-промахи при обходе списка

# Стек



- Представляет собой один из возможных способов организации памяти. Ввод данных в стек осуществляется таким образом, что каждое последующее вводимое данное "накладывается" на предшествующие данные, а их считывание осуществляется по правилу "последним вошел — первым вышел" (LIFO).

- Такая организация памяти применяется при использовании структур данных с указателями.
- Пример: Чтобы в односвязном списке пройти в обратном направлении и вернуться по цепи в исходное состояние, необходимо значения всех пройденных указателей запомнить в стеке.

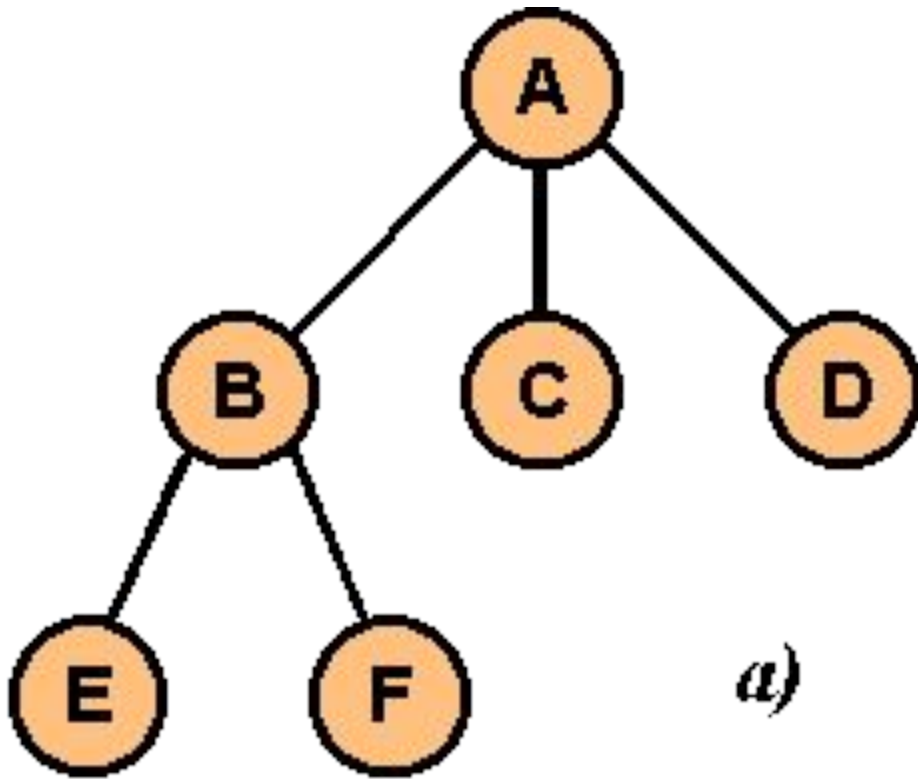
# Реализация стека

- Стек как динамическую структуру данных легко организовать на основе линейного списка.
- Поскольку работа всегда идет с заголовком стека, то есть не требуется осуществлять просмотр элементов, удаление и вставку элементов в середину или конец списка, то достаточно использовать экономичный по памяти линейный однонаправленный список.
- Для такого списка достаточно хранить указатель вершины стека, который указывает на первый элемент списка.
- Если стек пуст, то списка не существует, и указатель принимает значение NULL.

# Реализация очереди

- Очередь как динамическую структуру данных легко организовать на основе линейного списка. Поскольку работа идет с обоими концами очереди, то предпочтительно будет использовать линейный двунаправленный список. Хотя для работы с таким списком достаточно иметь один указатель на любой элемент списка, здесь целесообразно хранить два указателя – один на начало списка (откуда извлекаем элементы) и один на конец списка (куда добавляем элементы). Если очередь пуста, то списка не существует, и указатели принимают значение NULL.

# Древовидные структуры



a)

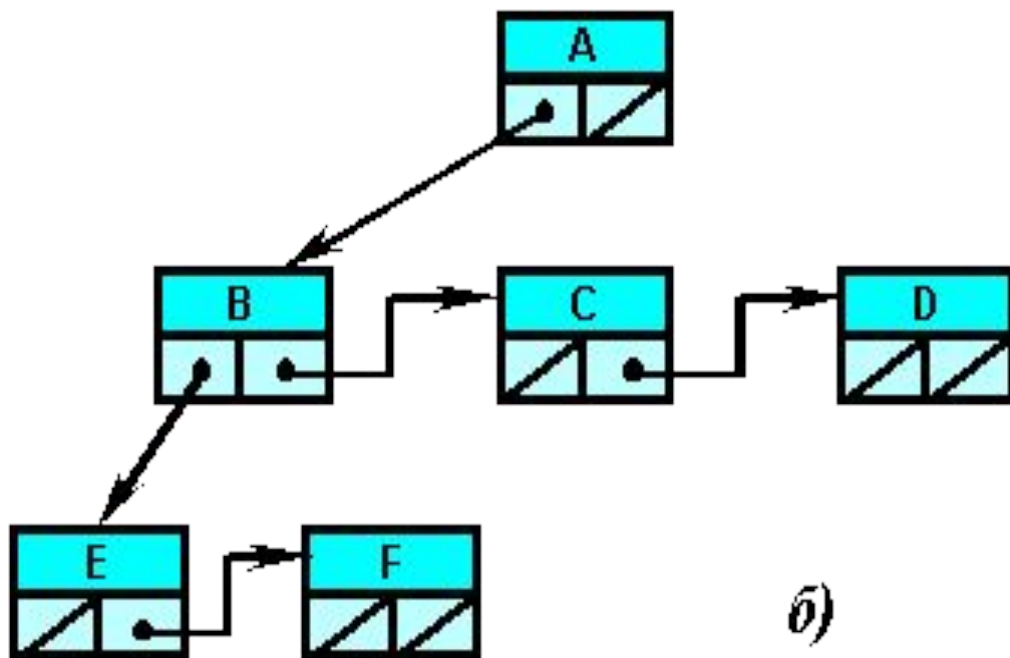
- Кроме линейных динамических структур данных часто используются нелинейные структуры. Разновидностью нелинейных структур являются деревья.
- Используются во многих структурированных данных.
- Существуют деревья с двумя ветвями (бинарное дерево) и дерево с большим количеством ветвей.

# Дерево. Основные определения

- Структура типа «дерево» характеризуется следующим свойством: Каждый объект в этой структуре, за исключением одного (корня) – самой верхней вершины подчинен только одному объекту из этой структуры.
- Кроме корня в дереве различают вершины – узлы, листья, родители, потомки.
- Дерево без вершин – **пустое дерево**.
- Дерево может содержать единственную вершину.
- **Корень** – вершина, не имеющая предков.
- **Листья (терминальные вершины)** – вершины, не имеющие потомков.
- Нетерминальные вершины называют **внутренними**.
- **Поддерево** – любая вершина вместе со своими потомками.



- Считается, что корень дерева находится на уровне 0.
- Максимальный уровень какой-либо из вершин дерева называется его **глубиной** или **высотой**.
- Число непосредственных потомков внутренней вершины называют **степенью вершины**.
- Максимальная степень всех вершин называется **степенью дерева**.
- Число ветвей (или ребер), которые нужно пройти от корня к вершине  $x$ , называется **длиной пути** к  $x$ .  
Длина пути к корневой вершине равна 0, а ко всем потомкам корня равна 1. Длина пути всего дерева определяется как сумма длин путей всех его компонент. Ее также называют **длиной внутреннего пути**.



б)

- В основном применяются деревья с большим количеством ветвей, но для обеспечения однородности структуры ячеек памяти деревья с большим количеством ветвей представляются в форме бинарных деревьев.



**б)**

- Структура узла содержит информационную часть и указатели на порожденный узел и на подобный узел.

Рассмотренный ранее двумерный массив можно представить в виде древовидной структуры.

