

# ЯЗЫКИ ПРОГРАММИРОВАНИЯ И СТРУКТУРЫ ДАННЫХ

## Лекция 3

**Массивы и указатели.**

**Динамические структуры данных.**

**Линейные структуры.**

**Списки.**



<https://do.ssau.ru/moodle/course/view.php?id=1375>

<https://do.ssau.ru/moodle/mod/forum/view.php?id=34435>

# Тип: указатель

**Указатель** – переменная, которая хранит адрес другой переменной (адрес памяти).

- Аналогия с указателями у дороги – они содержат информацию о том, как добраться до нужного места.

**Указатель** (англ. *pointer*) — переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения — *нулевого адреса*.

**NULL** используется для указания того, что в данный момент указатель не хранит адрес переменной.

Объявляется с использованием символа **\*** перед именем переменной.

Формат

*тип \*имя\_переменной;*

Пример

```
int *a; // указатель на переменную
        // целочисленного типа
```

# Операции над указателями

Две основные операции над указателями:  
**присваивание** и **разыменование**.

- **Присваивание**

- Для присваивания указателю некоторого адреса используется унарный оператор адресации **&**, который возвращает адрес памяти, по которому расположен операнд.

- **Разыменование**

- для обращения к значению в памяти, на которое указывает указатель, используется унарный оператор разыменование **\***, который возвращает значение переменной, расположенной по адресу, на который указывает операнд.

- **Нулевой указатель**

- указатель, хранящий специальное значение, используемое для того, чтобы показать, что данная переменная-указатель не ссылается (не указывает) ни на какой объект
- В языке C++ это **0** или макрос **NULL**.

# Арифметика указателей

Операции: `++`, `--`, `+` и `-`

Пример.

Пусть `p` – указатель на `int` переменную с адресом 2000.

После выполнения `++p`  
`p` указывает на адрес 2004  
(следующее `int` значение).

Указатели можно сравнивать, используя операторы отношения `==`, `<` и `>`.

Смысл имеет только сравнение указателей одного и того же типа.

# Структуры данных

- **Структура данных** – это форма хранения и представления информации.
- Структуры данных бывают **простыми и сложными**: представляют атомарную единицу информации или набор однотипных данных.
- **Простые структуры данных** характеризуются **типом хранимой единицы информации**, например, целочисленный, вещественный, логический, текстовый тип и т.д.
- **Сложные структуры данных** делятся на **динамические и статические**.
- **Динамические** в процессе своего жизненного цикла позволяют изменять свой размер (добавлять и удалять элементы), а **статические** – нет.

# Классификация сложных структур данных

## ● Линейные

- Массив
- Список
- Стек
- Очередь
- Хэш-таблица

## ● Иерархические

- Двоичные деревья
- N-арные деревья
- Иерархический список

## ● Сетевые

- Простой граф
- Ориентированный граф

## ● Табличные

- Таблица реляционной базы данных
- Двумерный массив
- Разреженный массив (ортогональные списки)

# Динамические структуры данных

- Основным свойством динамических структур является отсутствие физической смежности элементов структуры в памяти и непостоянство числа элементов структуры в процессе её обработки.
- Размещение динамической структуры может быть реализовано как на смежной, так и на связной (как правило) памяти.
- Каждый элемент структуры на смежной памяти состоит из двух полей:
  - информационного поля или поля данных;
  - служебного поля – поля связок, в котором содержатся один или несколько указателей, связывающих данный элемент с другими элементами структуры.
- Размер структуры ограничивается только доступным размером памяти;
- При изменении логической последовательности элементов структуры не требуется их перемещения в памяти, достаточно лишь скорректировать указатели.

# Классификация динамических структур данных

## Последовательность и

- Вектор
- Матрица
- Строка
- Запись
- Очередь
- Стек
- Дек

## Деревья

- Бинарные
- Сортированные бинарные

## Сети

Динамические линейные структуры:

1. Очередь – структура данных, реализующая: добавление – в конец, а удаление – из начала.
2. Стек – структура данных, реализующая: добавление и удаление с одной стороны.
3. Дек – структура данных, реализующая: добавление и удаление с двух сторон.





# Линейные структуры данных

- **Массив** – это линейная структура однотипных данных, занимающих непрерывное пространство в памяти машины.



- Упорядоченность элементов массива определяется набором целых чисел, называемых индексами, которые связываются с каждым элементом массива и однозначно определяют его положение среди остальных элементов этого массива.

# Составной типа данных: Массив

**Массив** – набор элементов

- одинакового типа
- расположенных в памяти подряд (друг за другом)
- обращение происходит с применением общего имени
- обращение к конкретному элементу осуществляется по индексу

Массив является структурой с произвольным доступом.

Массивы с одним индексом называют **одномерными**, с двумя — **двумерными** и т. д.

Одномерный массив соответствует вектору, двумерный — матрице.

# Определение массива

Статическими называют массивы, размер которых в программе определён и не может меняться.

*тип имя\_массива [размер-константа] ;*

```
int A[4];
```

```
// одномерный статический массив целых чисел длины 4  
// нумерация элементов от 0 до 3
```

Номер элемента	0-й	1-й	2-й	3-й	...
Имя элемента	A[0]	A[1]	A[2]	A[3]	...
Адрес элемента	0108	010a	010c	010e	...

*тип имя\_массива [размер1] [размер2] ;*

```
int A[4][10];
```

```
// двумерный статический массив целых чисел длины ????  
// нумерация элементов ????
```

# Динамический массив

**Динамическим** называется массив, размер которого может меняться во время исполнения программы.

- Динамические массивы делают работу с данными более гибкой, так как не требуют предварительного определения хранимых объёмов данных, а позволяют регулировать размер массива в соответствии с реальными потребностями.
- Указатель можно рассматривать как динамический массив.
- С помощью операторов `new/delete` можно выделять/освобождать память для динамического массива.
- Указатель ссылается на первый элемент массива (имя массива без индекса образует указатель на начало этого массива).
- `ptr[4]` эквивалентно `*(ptr+4)`
- Позволяет обращаться к элементам массива по индексу.

# Операторы выделения и освобождения памяти

**new** *тип*[*размер*]  
возвращает адрес  
непрерывного участка  
памяти для объекта  
типа *ТИП* размерностью  
*размер*

**delete**[ ] *имя\_указателя*  
освобождает память,  
выделенную оператором `new`,  
начиная с адреса, на который  
ссылается указатель.

## Пример.

```
int ar_sz=10;  
float *ptr = new float[ar_sz];  
// выделили область памяти  
ptr[1] = 1; ptr[2] = 2;  
delete[] ptr; // освободили память
```

# Массивы указателей

Указатели могут храниться в массивах.

```
// Объявление динамического 5-элементного
// массива указателей на int.
const int arr_size = 5;
void main() {
    int *arr[arr_size];
    for (int i = 0; i < arr_size; i++) {
        arr[i] = new int[i+1];
        for (int j = 0; j < i + 1; j++) {
            arr[i][j] = j + 1;
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}
```

**Вывод:**

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

# Двумерный массив

В предыдущем примере – фактически получили двумерный массив с размерами: 1 2 3 4 5

Двумерный массив  $\Leftrightarrow$  Одномерный массив одномерных массивов.

Объявление:

```
Тип имя_массива[размер1][размер2];
```

```
int B[3][10];
```

```
// матрица с 3 строками и 10 столбцами
```

**`B[2, 5]` – неправильное обращение**

`B[2][5]` – правильное обращение

```
char имя_массива[размер1][размер2]
```

– массив строк

# Массивы указателей

## Выделение и освобождение памяти.

### Динамический двумерный массив.

```
// выделять память нужно так
int **arr;
arr = new int*[N];
for(int i = 0; i < N; i++)
    arr[i] = new int[N];
```

```
// освобождать память нужно так
for(int i = 0; i < N; i++)
    delete[] arr[i];
delete[] arr;
```



# Утечки памяти

Если не освобождают память, то может остаться «мусор» - фактически занятые на время выполнения программы участки памяти, к которым из программы нельзя обратиться.

```
int *arr;  
arr = new int[5];  
arr = new int[6]; // утечка памяти
```

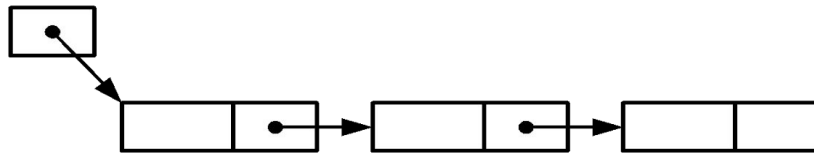
Выделенная память должна освобождаться.  
Каждому **new** должен соответствовать **delete**.

# Линейные структуры данных

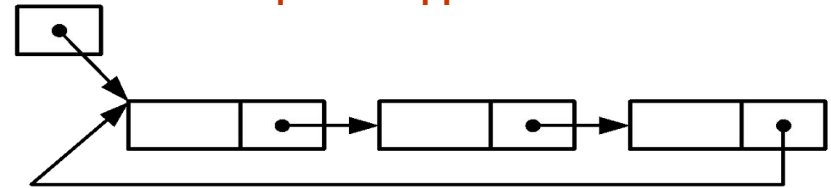
- **Список** – совокупность элементов типа структура, расположенных в произвольных местах памяти, связанных друг с другом через поля связи – переменные в которых хранятся ссылки (адреса) на следующий (предшествующий) элемент.
- Динамическая линейная структура данных, в которой каждый элемент ссылается только на предыдущий – *однонаправленный линейный список*, ссылается на предыдущий и следующий за ним – *двунаправленный линейный список*.
- Достоинство этой структуры данных, помимо возможности изменять размер, - это простота реализации.

# Виды списков

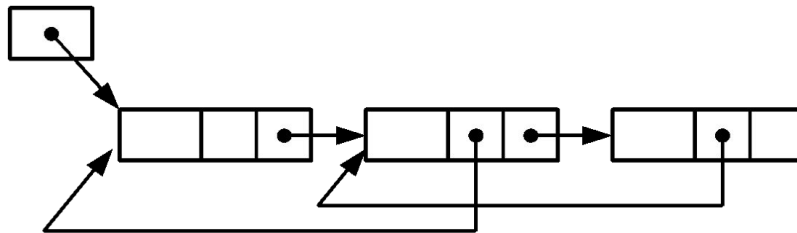
Линейный односвязный



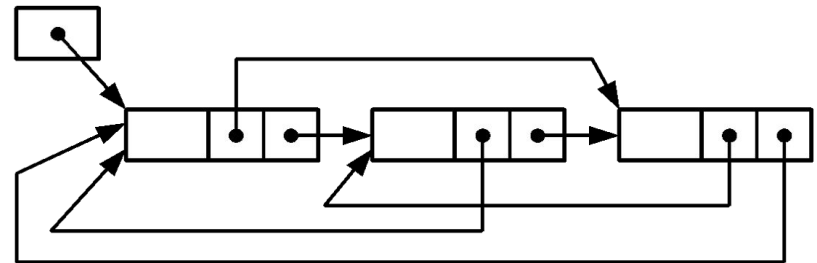
Кольцевой односвязный



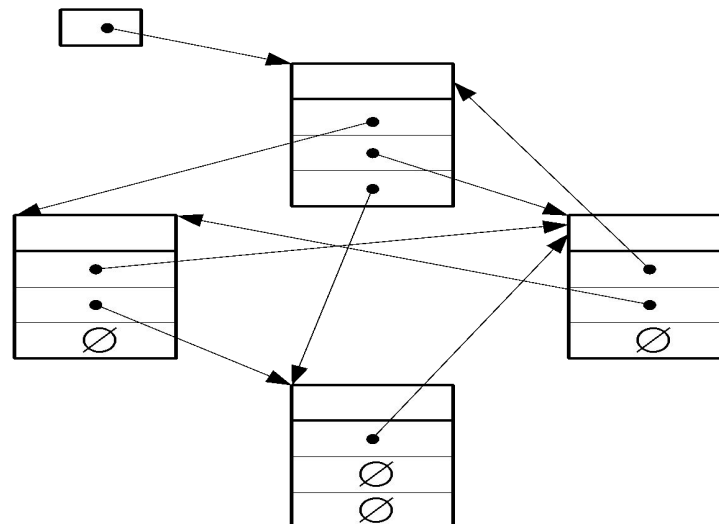
Линейный двусвязный



Кольцевой двусвязный



Сетевой n-связный



# Линейный односвязный список

```
struct list  
{ int val;  
  list *next;}
```

## Операции:

```
int data;  
list * plist;
```

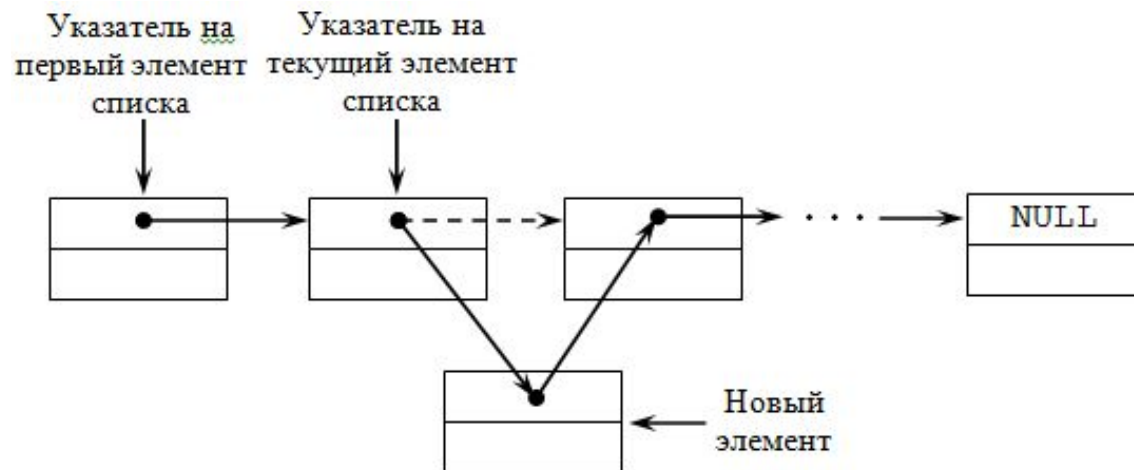
```
list* push(list* ,int);  
int pop(list*);
```

# Однонаправленный список.

## Добавление узла

- Для добавления узлов достаточно изменить значения адресных полей.
- Вставка первого и последующих элементов списка отличаются друг от друга.

Поэтому в функции, реализующей данную операцию, сначала осуществляется проверка, на какое место вставляется элемент.

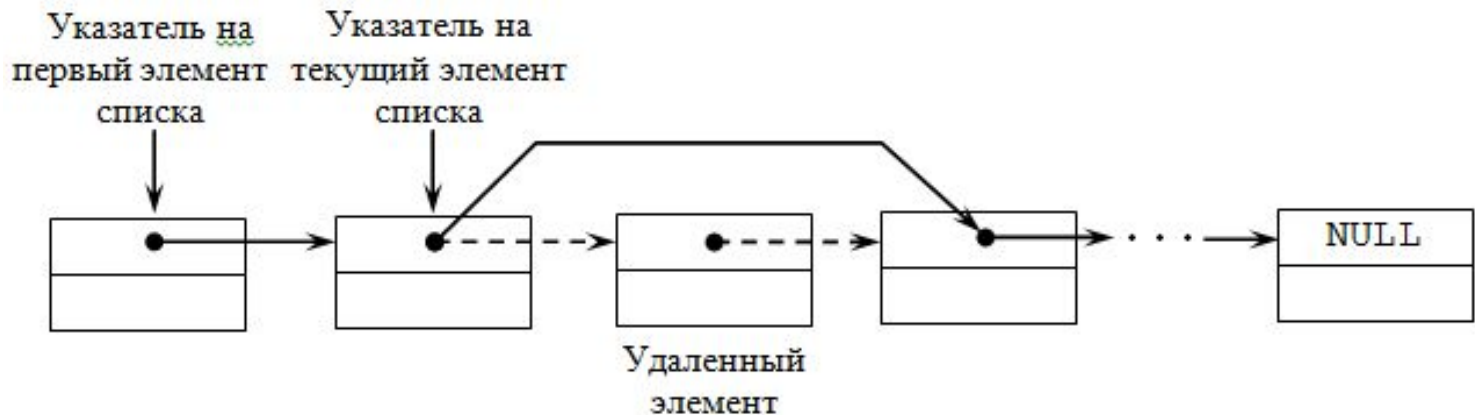


# Однонаправленный список.

## Удаление узла

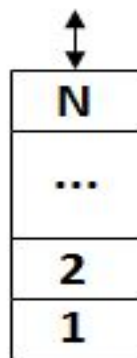
После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга.



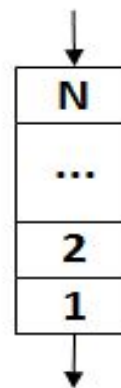
# Стек и очередь

- **Стек** – это динамическая линейная структура данных, для которой определены всего две операции изменения набора элементов: добавление элемента в конец и удаление последнего элемента.
- Ещё говорят, что стек реализует принцип LIFO (Last in, First Out) – последним пришёл и первым ушёл.



Стек.

- **Очередь** – очень похожая на стек, динамическая структура данных, с той лишь разницей, что она реализует принцип FIFO (First in, First out) – первым пришёл и первым ушёл. В программировании с помощью очередей, например, обрабатывают события пользовательского интерфейса, обращения сервисам и прочие информационные запросы.



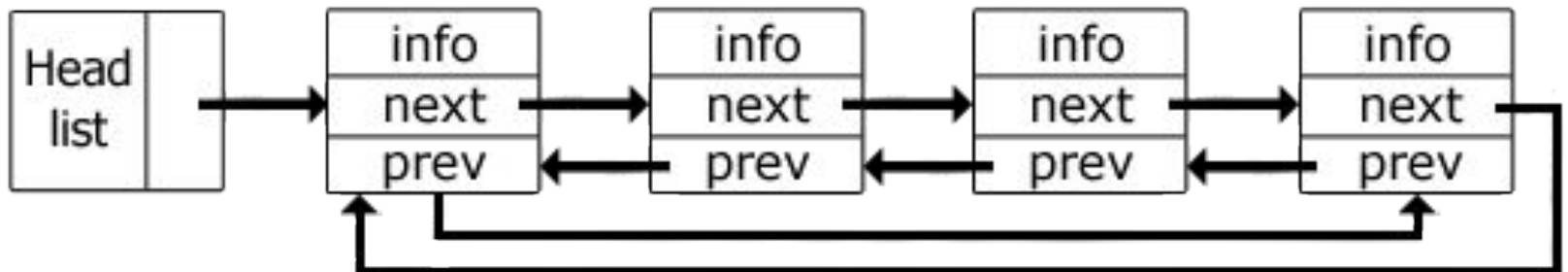
Очередь.

# Кольцевой связный список

Может быть односвязным или двусвязным.

Последний элемент кольцевого списка содержит указатель на первый, а первый (в случае двусвязного списка) — на последний.

Узла с указателем на NULL не существует.





# XOR-связный список

В каждом элементе хранится только один адрес — результат выполнения операции XOR над адресами предыдущего и следующего элементов списка.

Для того, чтобы перемещаться по списку, необходимо взять два последовательных адреса и выполнить над ними операцию XOR, которая и даст реальный адрес следующего элемента.

## Пример

Пусть даны указатели `First` и `Second`.

`First` указывает на узел № 2, `Second` — на узел 3.

`First->P` = XOR от адресов узлов № 1 и 3,

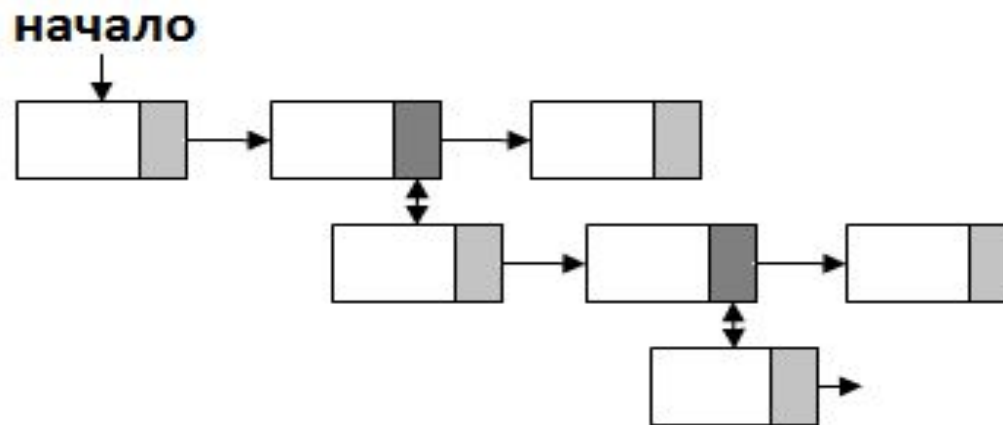
`Second->P` = XOR от адресов узлов № 2 и 4.

XOR (`First->P`, `Second`) = адрес узла 1

XOR (`Second->P`, `First`) = адрес узла 4

# Иерархический список

- Каждый элемент списка может быть также началом списка следующего подуровня иерархии.
- Пример иерархического списка – структура интернет форумов: последовательность сообщений образует линейный список, в то время как сообщения, являющиеся ответами на другие сообщения, порождают новые потоки обсуждения.



Иерархический список.