

### Теоретический материал

Структура данных — это способ организации информации для более эффективного использования. В программировании структурой обычно называют набор данных, связанных определённым образом. Например, массив — это структура. Со структурой можно работать: добавлять данные, извлекать их и обрабатывать, например изменять, анализировать, сортировать. Для каждой структуры данных — свои алгоритмы. Работа программиста — правильно выбирать уже написанные готовые либо писать свои.

Главное свойство структур данных в том, что у любой единицы данных должно быть чёткое место, по которому её можно найти. Как определяется это место и как происходит поиск, зависит от конкретной структуры.

Характеристики структур данных следующие:

- Данные в памяти представлены определённым образом, который однозначно позволяет определить структуру.
- Чаще всего внутри структуры можно добавить элемент или извлечь оттуда. Это свойство не постоянное — бывают структуры, которые нельзя изменять после создания.
- Существуют алгоритмы, которые позволяют взаимодействовать с этой структурой.

При этом данных необязательно должно быть много. Массив из одного элемента — уже структура данных.

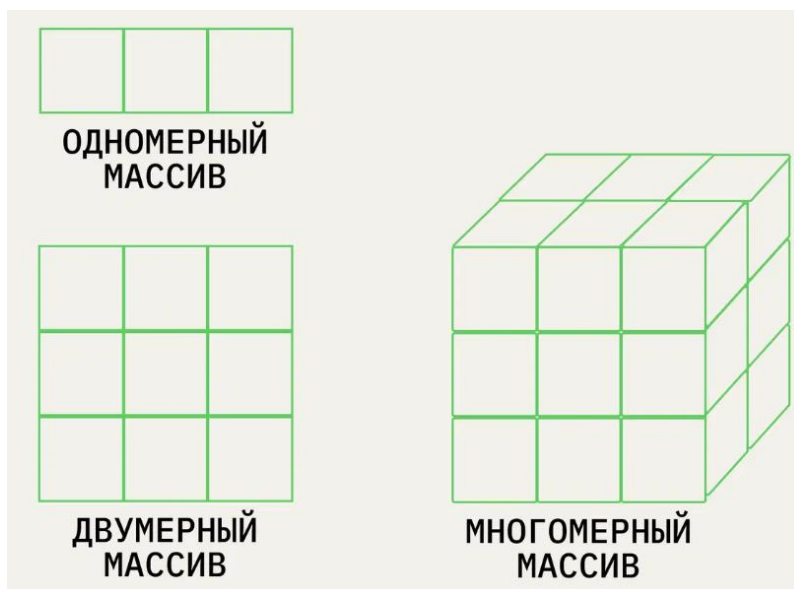
#### 1. Массив (Array)

Одна из самых простых структур данных, которая встречается чаще всего. Именно на массивах основаны многие другие структуры данных: списки, стеки, очереди.

Для простоты восприятия можно считать, что массив — это таблица. Каждый его элемент имеет индекс — «адрес», по которому этот элемент можно извлечь. В большинстве языков программирования индексы начинаются с нуля. То есть первый элемент массива имеет индекс не [1], а [0]. Данные в массиве можно просматривать, сортировать и изменять с помощью специальных операций.

*Массивы бывают двух видов:*

- **Одномерные.** У каждого элемента только один индекс. Можно представить это как строку с данными, где одного номера достаточно, чтобы чётко определить положение каждой переменной.
- **Многомерные.** У каждого элемента два или больше индексов. По сути, это комбинация из нескольких одномерных массивов, то есть вложенная структура.



*Как применяют массивы:*

- В качестве блоков для более сложных структур данных. Массивы предусмотрены в синтаксисе большинства языков программирования, и на их основе удобно строить другие структуры.
- Для хранения несложных данных небольших объёмов.
- Для сортировки данных.

## **2. Динамический массив (Dynamic array)**

В классическом массиве размер задан заранее — мы точно знаем, сколько в нём индексов. А динамический массив — это тот, у которого размер может изменяться. При его создании задаётся максимальная величина и количество заполненных элементов. При добавлении новых элементов они сначала заполняются до максимальной величины, а при превышении сразу создаётся новый массив, с большей максимальной величиной.

Элементы в динамический массив можно добавлять без ограничений и куда угодно. Однако, если добавлять их в середину, остальные придётся сдвигать, что

занимает много времени. Поэтому лучше всего динамический массив работает при добавлении элементов в конце.

*Как применяют динамические массивы:*

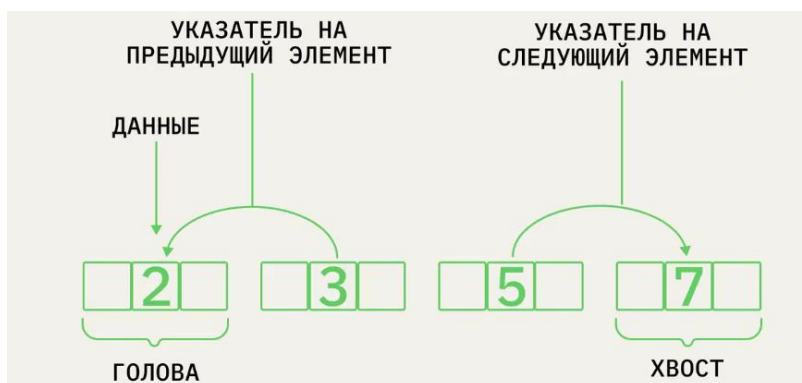
- В качестве блоков для структур данных.
- Для хранения неопределённого количества элементов.

### 3. Связный список (Linked list)

Ещё одна базовая структура данных, которую, как и массивы, используют для реализации других структур. Связный список — это группа из узлов. В каждом узле содержатся:

- Данные.
- Указатель или ссылка на следующий узел.
- В некоторых списках — ещё и ссылка на предыдущий узел.

В итоге получается список, у которого есть чёткая последовательность элементов. При этом сами элементы более разрозненны, чем в массиве, поскольку хранятся отдельно. Быстро перемещаться между элементами списка помогают указатели.

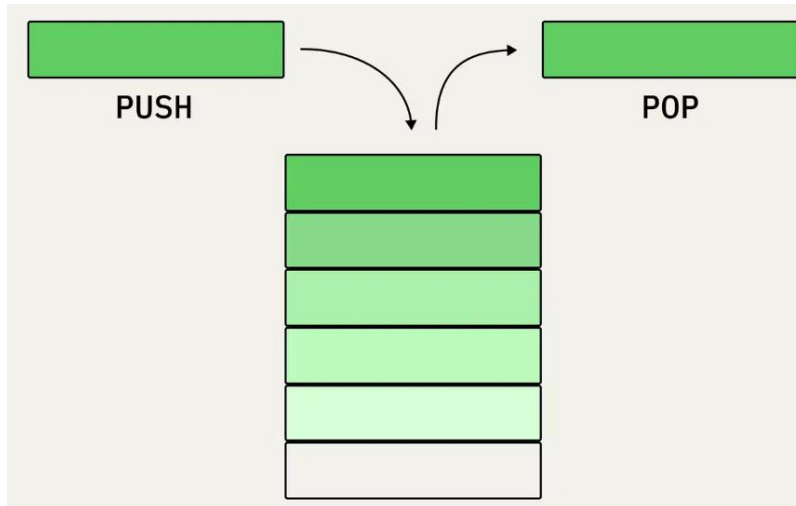


*Как применяют связные списки:*

- Для построения более сложных структур данных.
- Для реализации файловых систем.
- Для формирования хэш-таблиц.
- Для выделения памяти в динамических структурах данных.

## 4. Стек (Stack)

Эта структура данных позволяет добавлять и удалять элементы только из начала. Она работает по принципу LIFO — Last In, First Out (англ. «последним пришёл — первым ушёл»). Последний добавленный в стек элемент должен будет покинуть его раньше остальных.

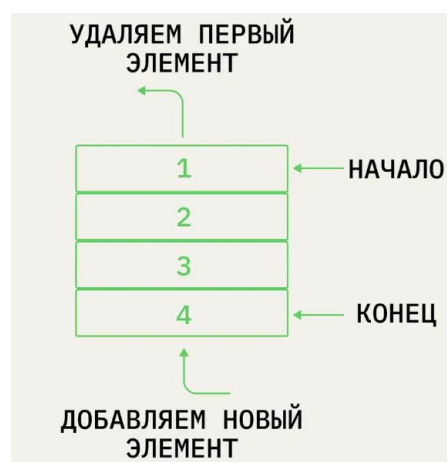


*Как применяют стеки:*

- Для реализации рекурсии.
- Для вычислений постфиксных значений.
- Для временного хранения данных, например истории запросов или изменений.

## 5. Очередь (Queue)

Этот вид структуры представляет собой ряд данных, как и стек. Но в отличие от него она работает по принципу FIFO — First In, First Out (англ. «первым пришёл — первым ушёл»). Данные добавляют в конец, а извлекают из начала.



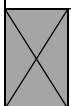
Бывают неклассические, двусторонние очереди (деки). В них можно добавлять элементы и извлекать их из начала и конца структуры. Элементы посередине недоступны.

**Как применяют очереди:**

- Для реализации очередей, например на доступ к определённому ресурсу.
- Для управления потоками в многопоточных средах.
- Для генерации значений.
- Для создания буферов.

**Пример 1**

**Задача:**



Написать на языке C++ программу, в которой создается и распечатывается связный список из произвольного количества элементов (большего 2).

**Решение:**

```

1  #include <iostream>
2  #include <ctime>
3  using namespace std;
4
5  // Узел связанного списка
6  struct Node
7  {
8  public:
9      int key;          // поле данных
10     Node* next;       // указатель на следующий узел
11 };
12
13 // Вспомогательная функция для генерации нового узла связанного списка из кучи
14 Node* newNode(int key)
15 {
16     // выделяем память под новый узел и устанавливаем его данные
17     Node* node = new Node;
18     node->key = key;
19
20     // указатель нового узла ни на что не указывает
21     // nullptr - ключевое слово C++. Обозначает нулевой указатель,
22     // который никуда не указывает
23     node->next = nullptr;
24
25     return node;
26 }
27
28 // Функция для реализации связанного списка, содержащего n узлов
29 Node* constructList(int n)
30 {
31     Node* head = newNode(1); //указатель на первый узел
32
33     Node* last = newNode(2); //указатель на последний узел (изначально - второй узел)
34     head->next = last;
35
36     srand(time(NULL));
37     for (int i = 0; i < n-2; i++)
38     {
39         Node* new_node = newNode(rand() % 100 + 1); //случайное число от 1 до 100
40         last->next = new_node;
41         last = new_node;
42     }
43
44     // возвращаем указатель на первый узел в списке
45     return head;
46 }
47
48 // Вспомогательная функция для печати связанного списка
49 void printList(Node* head)
50 {
51     Node* ptr = head;
52     while (ptr)
53     {
54         cout << ptr->key << " -> ";
55         ptr = ptr->next;
56     }
57
58     cout << "nullptr";
59 }
60
61 int main()
62 {
63     // 'head' указывает на первый узел (также известный как головной узел) связанного списка
64     Node* head = constructList(10);
65
66     // распечатать связанный список
67     printList(head);
68
69     return 0;
70 }

```

**Ответ:**

```
Консоль отладки Microsoft Visual Studio
1 -> 2 -> 50 -> 24 -> 89 -> 16 -> 92 -> 2 -> 77 -> 7 -> nullptr
C:\Users\Alexander\source\repos\rt6_1\x64\Debug\rt6_1.exe (процесс 15196) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

## Задание 1

**Задача:**

Создать массив натуральных чисел произвольного размера  $n$ ,  $[a_1, \dots, a_n]$ .  
Определить и вывести на экран количество элементов массива:

- а) являющихся нечетными числами;
- б) являющихся квадратами четных чисел;
- в) кратных 3 и не кратных 5;
- г) удовлетворяющих условию  $2^k < a_k < k!$ ;
- д) удовлетворяющих условию  $a_k < \frac{a_{k-1} + a_{k+1}}{2}$ ;
- е) имеющих четные порядковые номера и являющихся нечетными числами.

**Решение:**

**Ответ:**

## Задание 2

**Задача:**

Создать массив натуральных чисел произвольного размера  $n$ ,  $[a_1, \dots, a_n]$ . Найти количество и сумму тех элементов массива, которые делятся на 5 и не делятся на 7.

**Решение:**

**Ответ:**








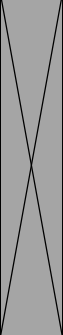

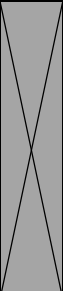
## Задание 3

**Задача:**

Создать массив целых чисел произвольного размера  $n$ ,  $[a_1, \dots, a_n]$ . Имеются ли в массиве:

- а) два идущих подряд нулевых члена;
- б) три идущих подряд нулевых члена?

**Решение:**

<b>Ответ:</b>	
	
<b>Задание 4</b>	
<b>Задача:</b>	
	Создать связанный список произвольного размера, элементы которого содержат целые числа. Посчитать сумму всех целых чисел в связанном списке.
<b>Решение:</b>	
	
<b>Ответ:</b>	
	
<b>Задание 5*</b>	
<b>Задача:</b>	
	Создать массив действительных чисел. Выяснить, является ли он упорядоченным по убыванию.
<b>Решение:</b>	
	
<b>Ответ:</b>	
	
<b>Задание 6*</b>	
<b>Задача:</b>	
	<p>Для связного списка реализовать следующие операции в виде функций:</p> <ul style="list-style-type: none"> <li>• Операция, проверяющая список на пустоту.</li> <li>• Три операции добавления объекта в список (в начало, конец или внутрь после любого (n-го) элемента списка);</li> <li>• Операция, возвращающая количество элементов в списке;</li> <li>• Операция доступа к списку, состоящему из всех элементов исходного списка, кроме первого.</li> </ul>
<b>Решение:</b>	
	
<b>Ответ:</b>	
<b>Задание 7*</b>	
<b>Задача:</b>	
	<p>Для связанных списков L1 и L2 (произвольного размера) описать функции, которые:</p> <ul style="list-style-type: none"> <li>а) проверяет на равенство списки L1 и L2;</li> <li>б) определяет, входят ли все элементы списка L1 в список L2;</li> <li>в) проверяет, есть ли в списке L1 хотя бы два одинаковых элемента;</li> <li>г) переносит в конец непустого списка L1 его первый элемент;</li> </ul>



	<p>д) переносит в начало непустого списка L1 его последний элемент;</p> <p>е) добавляет в конец списка L1 все элементы списка L2;</p> <p>ж) переворачивает список L1, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке;</p> <p>з) оставляет в списке L только первые вхождения одинаковых элементов.</p>
	<p><b>Решение:</b></p>
	<p><b>Ответ:</b></p>
<p><b>Задание 8*</b></p>	
<p><b>Задача:</b></p>	
	<p>Создать список из заданного количества элементов. Выполнить циклический сдвиг этого списка на N элементов вправо или влево.</p>
<p><b>Решение:</b></p>	
<p><b>Ответ:</b></p>	
<p><b>Задание 9*</b></p>	
<p><b>Задача:</b></p>	
	<p>Используя стек (например, <code>std::stack</code>, доступный в <code>#include &lt;stack&gt;</code>) напечатать содержимое текстового файла, выписывая буквы каждой его строки в обратном порядке.</p>
<p><b>Решение:</b></p>	
<p><b>Ответ:</b></p>	
<p><b>Задание 10*</b></p>	
<p><b>Задача:</b></p>	
	<p>Сформировать файл из натуральных чисел и с помощью очереди (например, <code>std::queue</code>, доступный в <code>#include &lt;queue&gt;</code>) за один просмотр файла напечатать элементы файла в следующем порядке: сначала все однозначные числа, затем двузначные, сохраняя исходный порядок чисел в каждой из этих групп.</p>
<p><b>Решение:</b></p>	
<p><b>Ответ:</b></p>	