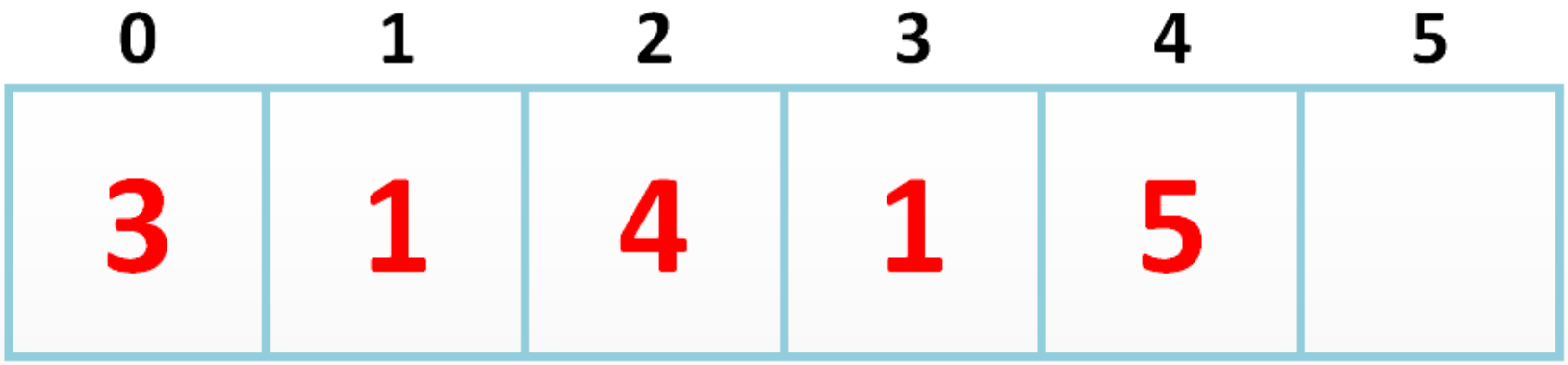
Лекция 1.
Динамические массивы и некоторые структуры данных.

План лекции

- Массивы
- Динамические массивы в C
- Динамические массивы в C++
- Вектор <vector>
- Стек и <stack>
- Очередь и <queue>
- Дек и <deque>

Массивы

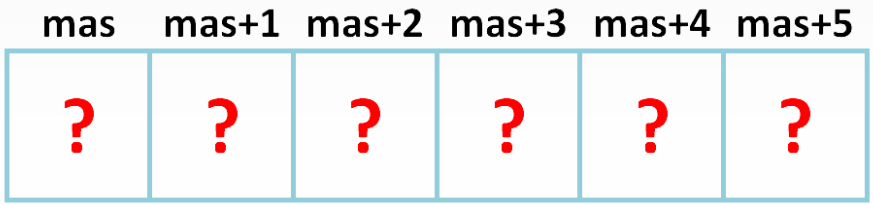
Массив – это набор компонентов, расположенных подряд в памяти.



Массивы

Объявление массива:

```
int mas[6];
```

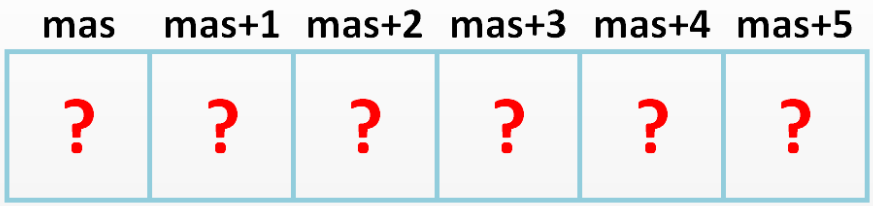


выделяет программе ровно столько памяти, сколько необходимо для хранения 6 элементов типа int.

Динамические массивы в С

Динамический массив - такой массив, размер которого может меняться по ходу выполнения программы.

```
int* mas = malloc(sizeof(int) * 6);
```

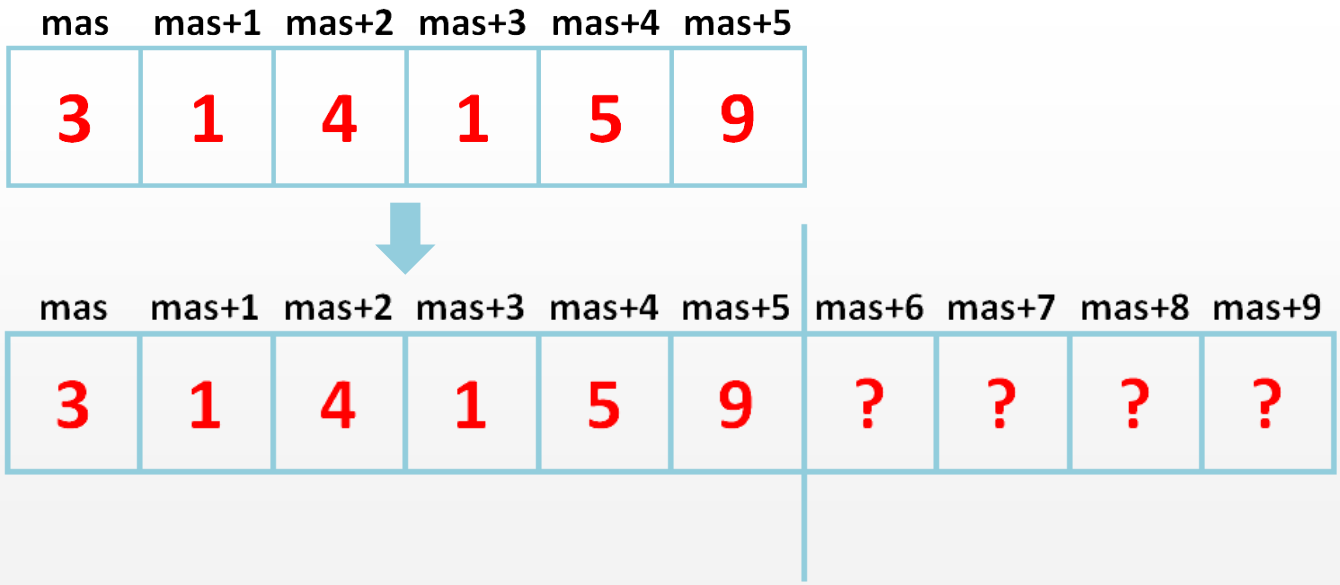


Память для массива выделяется в явном виде – на шесть элементов, для каждого элемента на размер типа данных int

Динамические массивы в C

Размер такого массива можно изменить, применив функцию realloc:

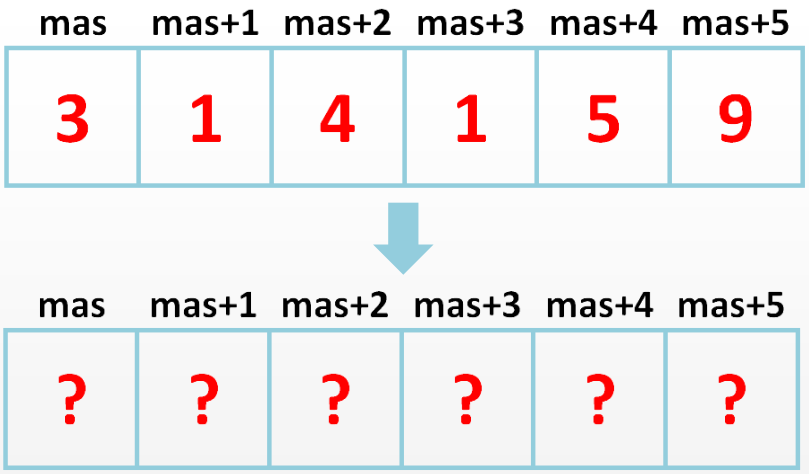
```
mas = realloc(mas, 10 * sizeof(int));
```



Динамические массивы в С

Чтобы освободить память, занятую массивом, используют функцию free:

```
free(mas);
```

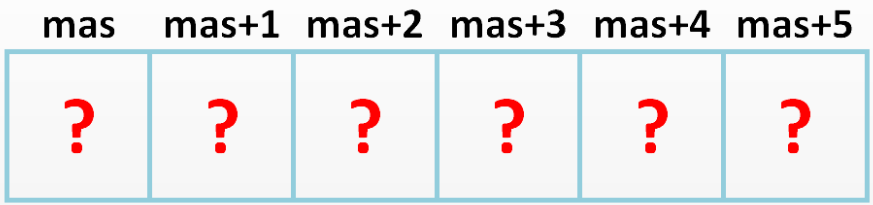


Здесь mas, mas+1 и т.д. обозначают адреса ячеек массива в памяти, после применения функции free массив mas как таковой перестаёт существовать.

Динамические массивы в C++

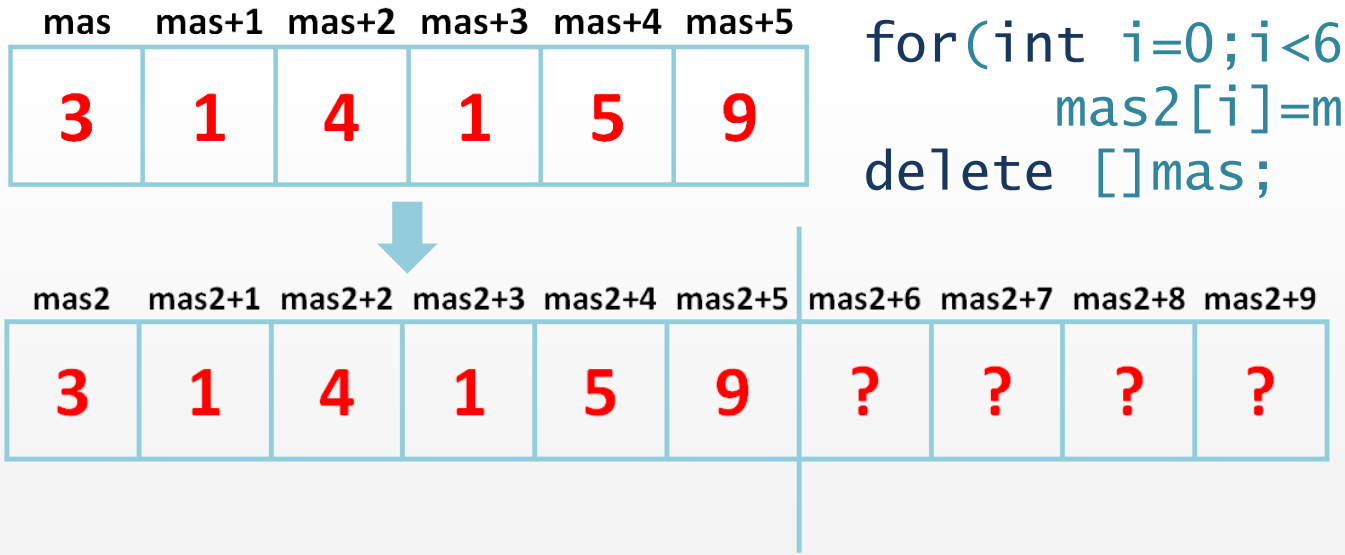
Динамический массив в C++ - это массив переменной длины. Функция new выделяет память на определённое количество элементов заданного типа данных.

```
int* mas = new int[6];
```



Динамические массивы в C++

Размер такого массива нельзя менять. Можно только создать новый массив нужного размера и скопировать в него данные из старого массива, а затем освободить память:




```
int* mas2 = new int[10];
for(int i=0;i<6;i++)
      mas2[i]=mas[i];
delete []mas;
```

Динамические массивы в C++

В C++ есть библиотека <vector>, добавляющая структуру данных, которая, по сути, является динамическим массивом.

```
#include <vector>
void main()
{
	//Инициализация вектора
	vector <int> mas;
}
```

<vector>

```
#include <vector>
```

Вектор работает, как массив, но гораздо более функционален. Когда как можно просто добавлять элементы в конец вектора, в массиве для этого нужно сначала освободить достаточно памяти.

```
//Вектор
vector <int> mas;
int a;

for(int i=0;i<n;i++)
{
	cin >> a;
	mas.push_back(a);
}
```

```
//Массив
int* mas = new int[n];

for(int i=0;i<n;i++)
	cin >> mas[i];
```

<vector>

```
#include <vector>
```

Но можно работать с вектором, как с массивом. Здесь `vector <int> mas(n);` автоматически расширяет вектор до n элементов, равных по умолчанию нулю.

```
//Вектор
vector <int> mas(n);

for(int i=0;i<n;i++)
        cin >> mas[i];
```

```
//Массив
int* mas = new int[n];

for(int i=0;i<n;i++)
        cin >> mas[i];
```

Работа с <vector>
#include <vector>

```
//Заполнение вектора n элементами, равными -1
vector <int> mas(n, -1);

//Получение размера (длины) вектора.
//size_t – это unsigned int. Можно писать просто int
size_t n = mas.size();

//Добавление элемента в конец
mas.push_back(100500);

//Первый элемент вектора
//То же самое, что и int a = mas[0]
int a = mas.front();

//Последний элемент вектора
//То же самое, что и int a = mas[mas.size()-1]
int a = mas.back();

//Удаление последнего элемента
mas.pop_back();
```

Работа с <vector>

#include <vector>

```
//Изменение размера вектора до n и, если добавляются
//новые элементы, заполнение их нулями
mas.resize(n);

//Изменение размера и заполнение новых элементов
//заданными значениями
mas.resize(n, -1);

//Очистка вектора (изменение размера до нуля)
mas.clear();
```

Работа с <vector>

```
#include <vector>
```

Итератор – указатель на элемент сложной стандартной структуры в C++. В векторе индекс элемента переопределён, как итератор.

```
//Вектор
vector <int> mas(n);
vector <int>::iterator it;

for(it=mas.begin();it!=mas.end();it++)
        cout << *it;
```

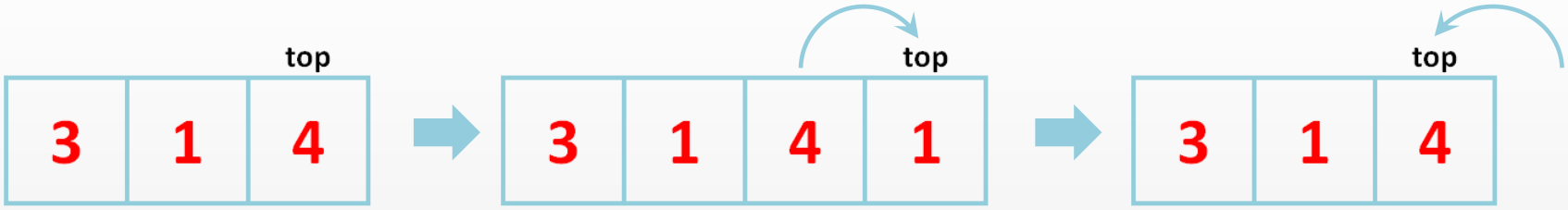
```
//Массив
int* mas = new int[n];

for(int i=0;i<n;i++)
        cout << mas[i];
```

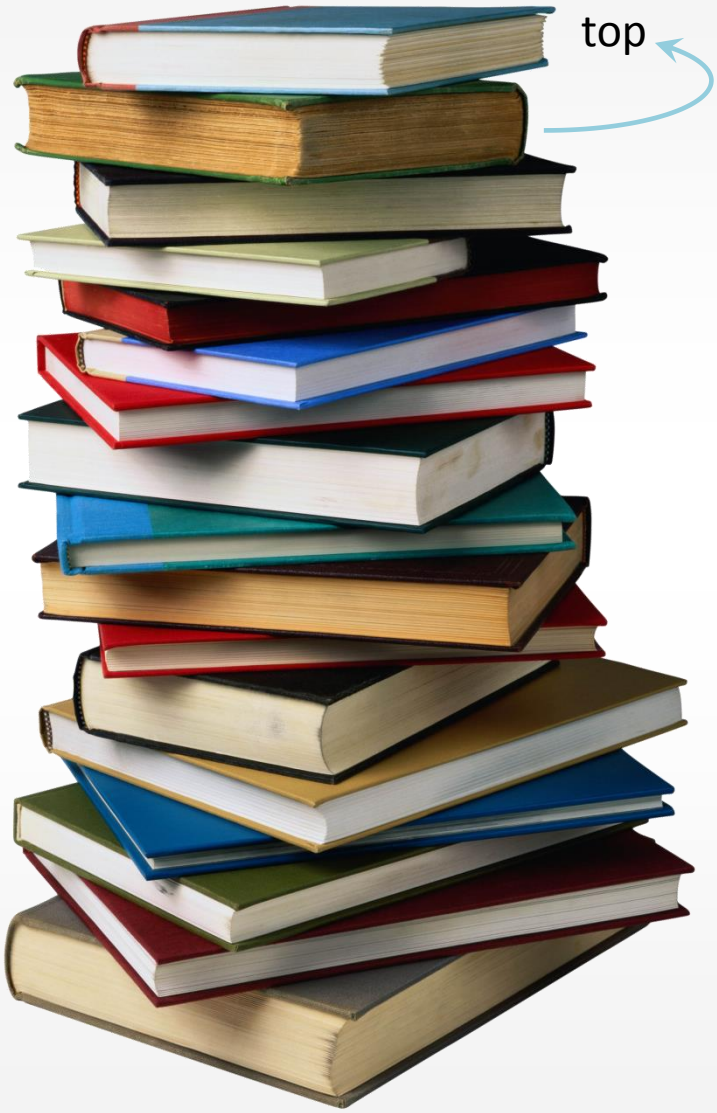
В итератор не считывают и не присваивают! Он – указатель!

Стек

Стек – это структура данных, представляющая из себя последовательность элементов, которые можно добавлять и удалять только с одного конца (LIFO, «Last In – First Out», последним вошёл – первым вышел)



Стек



Стек наглядно – стопка книг. Вытащить книгу из середины или низа стопки, особенно большой, не уронив ни одной книги, практически нереально.

Стек

```
//Инициализация стека, top – верхний элемент
int top = 0;
int s[100];

//Добавление элемента в стек
s[top] = a;
top++;

//Верхний элемент
a = s[top-1];

//Удаление элемента из стека
top--;
```

Работа с <stack>
#include <stack>

```
//Решение задачи о выводе n чисел в обратном порядке:
stack <int> s; //Инициализация стека

for(int i=0;i<n;i++) {
	cin >> a;
	s.push(a); //Добавление элемента в стек
}

//s.empty() возвращает true, если стек пуст
while(!(s.empty())) {
	cout << s.top(); //Верхний элемент
	s.pop(); //Удаление верхнего элемента
}
```

Очередь

Очередь – это структура данных, представляющая из себя последовательность элементов, которые можно добавлять с одного конца, а удалять с противоположного (FIFO, «First In – First Out», первым вошёл – первым вышел)

Очередь

Очередь наглядно – очень грустная очередь,
из которой нельзя выйти,
если ты стоишь не в начале.

Очередь

```
//Инициализация очереди
int front = 0; //Первый элемент
int back = 0; //Последний элемент + 1
int q[100];

//Добавление элемента в очередь
q[back] = a;
back++;
if(back == 100) back = 0;

//Первый элемент
a = q[front];

//Удаление первого элемента из очереди
front++;
if(front == 100) front = 0;
```

Работа с <queue>

#include <queue>

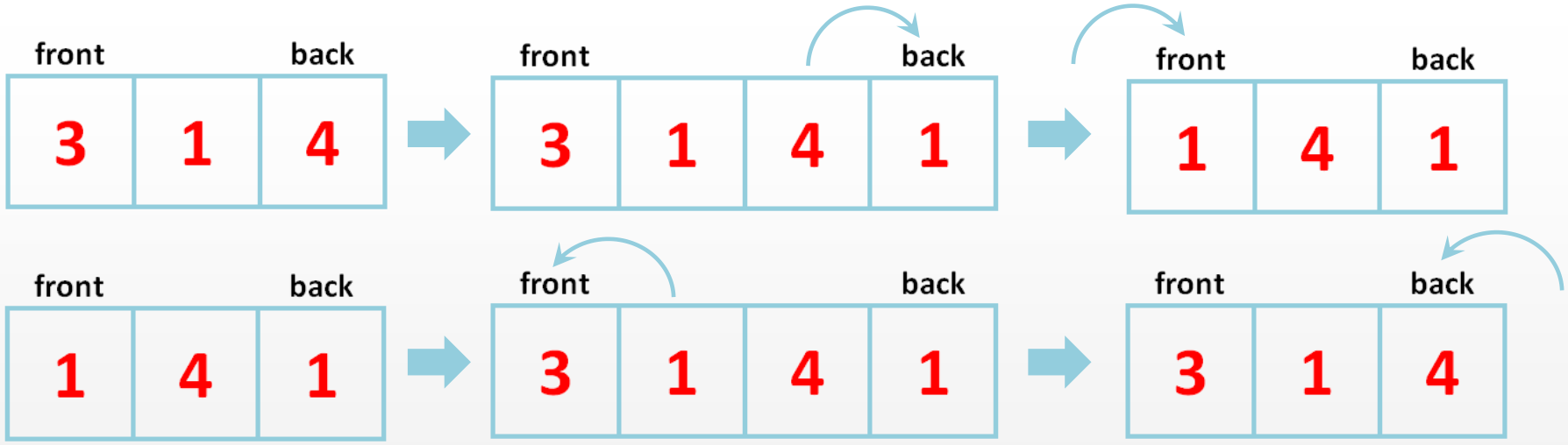
```
//Извращённое решение задачи о выводе n чисел:
queue <int> q; //Инициализация очереди

for(int i=0;i<n;i++) {
	cin >> a;
	q.push(a); //Добавление элемента в очередь
}

//q.empty() возвращает true, если очередь пуста
while(!(q.empty())) {
	cout << q.front(); //Первый элемент
	q.pop(); //Удаление первого элемента
}
```

Дек

Дек – это структура данных, представляющая из себя последовательность элементов, которые можно добавлять и удалять с обоих концов (Double Ended QUEue, очередь с двумя концами)



Дек

Дек наглядно – это книги на полке.
Гораздо удобнее добавлять книги с краю,
чем запихивать их в середину.

Дек

```
//Инициализация дека:

//Первый элемент                    //Последний элемент + 1
int front = 0;                      int back = 0;
int d[100];

//Добавление элемента в дек

front--;                            d[back] = a;
if(front < 0) front = 99;           back++;
d[front] = a;                       if(back == 100) back = 0;

//Первый элемент                    //Последний элемент
a = s[front];                       a = s[(back-1)%100];

//Удаление элемента из дека

front++;                            back--;
if(front == 100) front = 0;         if(back < 0) back = 99;
```

Работа с <deque>

```
#include <deque>
```

```
deque <int> d; //Инициализация дека

d.push_front(a); //Добавление элемента в дек с начала
d.push_back(a); //Добавление элемента в дек с конца

d.pop_front(a); //Удаление элемента из дека с начала
d.pop_back(a); //Удаление элемента из дека с конца

a = d.front(); //Первый элемент
a = d.back(); //Последний элемент

d.empty(); //Проверка на пустоту
```

Вообще, для всех перечисленных структур данных из стандартной библиотеки C++ действительны функции `empty()` и `size()`, а также работают итераторы.

Заключение

Сейчас гораздо удобнее использовать уже имеющиеся в языке структуры данных, чем писать новые – обычно в библиотеках уже реализовано множество удобных и оптимальных функций.

По этой же причине в последнее время программисты C++ предпочитают использовать <vector>, а не обычные «классические» массивы.

Вопросы?