Семинар №8

ФАКИ 2017

Бирюков В. А.

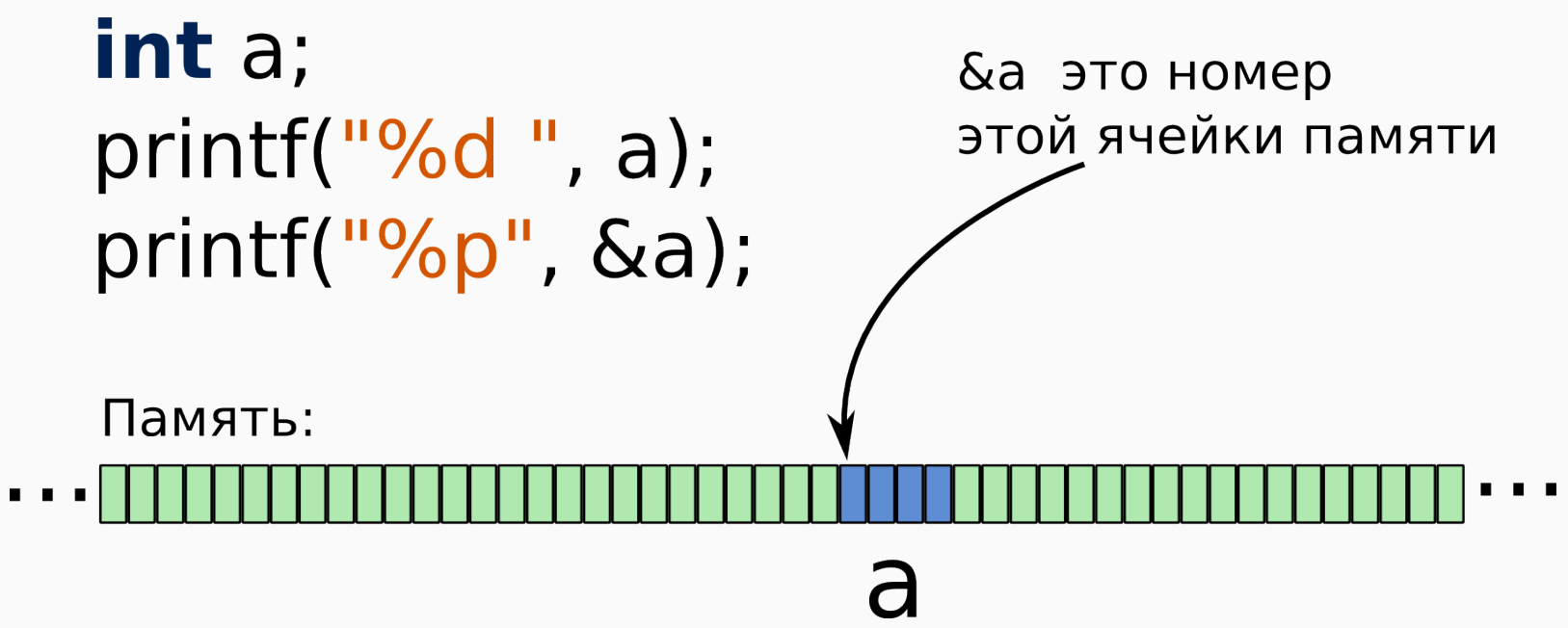
October 19, 2017

Указатели

Адрес переменной

У каждой переменной есть адрес – номер ячейки памяти.

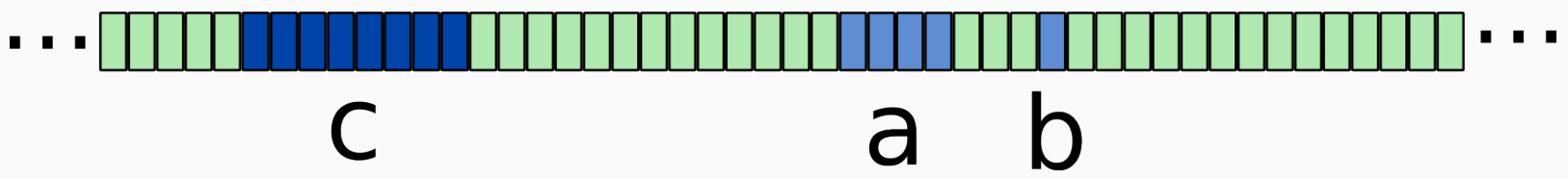
```
int a;
printf("%d ", a);
printf("%p", &a);
```



Указатель – это переменная, которая хранит адреса.

Размер указателя = 8 байт в 64-х битных системах.

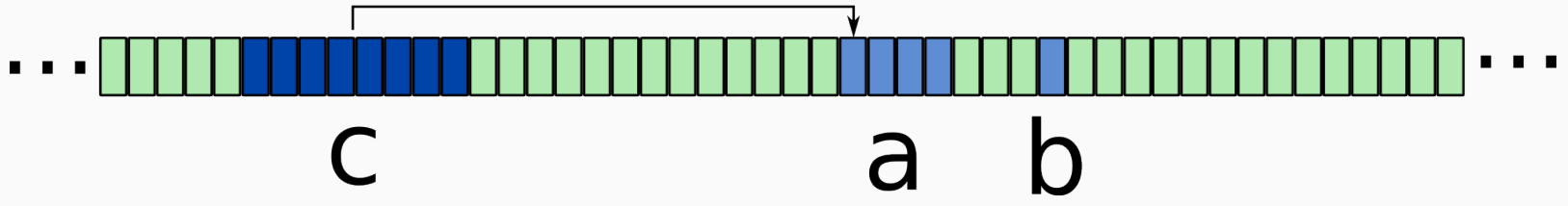
```
int a;
char b;
int* c;
```



Указатель – это переменная, которая хранит адреса.

Размер указателя = 8 байт в 64-х битных системах.

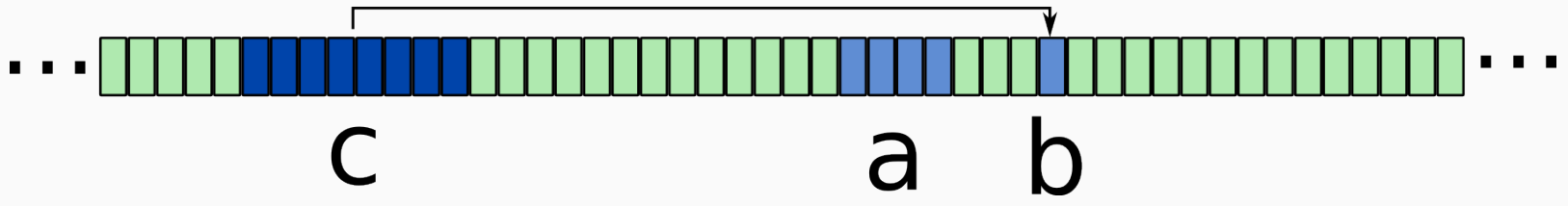
```
int a;
char b;
int* c = &a;
```



Указатель – это переменная, которая хранит адреса.

Размер указателя = 8 байт в 64-х битных системах.

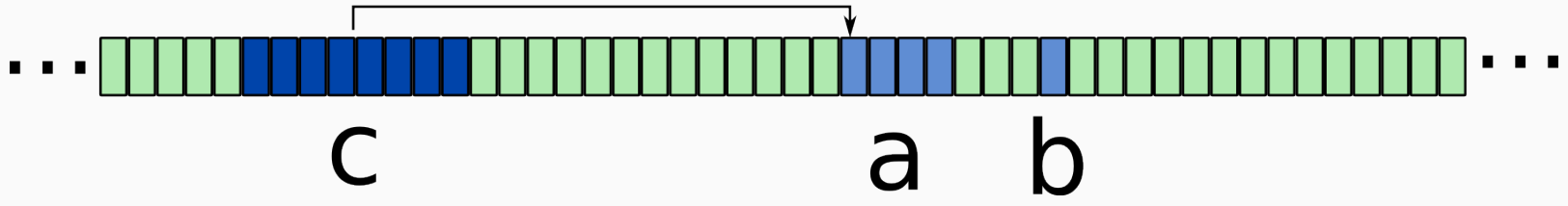
```
int a;
char b;
char* c = &b;
```



Указатель – это переменная, которая хранит адреса.

Размер указателя = 8 байт в 64-х битных системах.

```
int a = 1;
int* c = &a;
*c = 5;
```



Передача аргументов в функцию

Передача аргументов в функцию

Передача по значению

```
int min(int a, int b)
{
    if (a < b)
        b = a;
    return b;
}
int main()
{
    int a = 10, b = 40;
    int c = min(a, b);
    printf("%d\n", b);
}
```

Передача аргументов в функцию

Передача по значению

```
int min(int a, int b)
{
    if (a < b)
        b = a;
    return b;
}
int main()
{
    int x = 10, y = 40;
    int c = min(x, y);
    printf("%d\n", y);
}
```

Передача аргументов в функцию

Передача по значению

```
int min(int a, int b)
{
    if (a < b)
        b = a;
    return b;
}
int main()
{

    int c = min(10, 40);

}
```

Передача аргументов в функцию

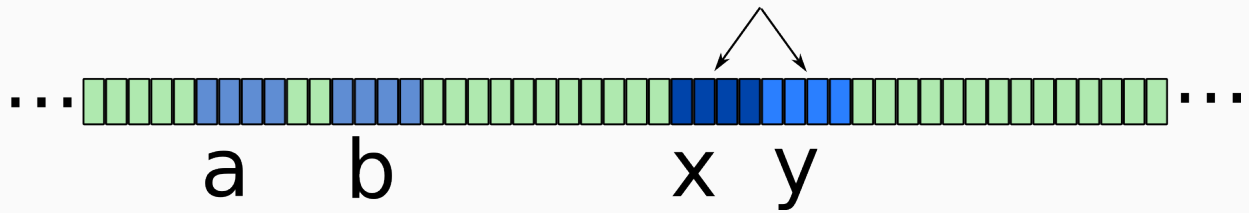
Передача с помощью указателей

```
void normalize(float* a, float* b)
{
    float sum = *a + *b;
    *a = *a / sum;
    *b = *b / sum;
}
int main()
{
    float x = 10.0, y = 40.0;
    normalize(&x, &y);
    printf("%f\n", y);
}
```

Функция swap()

Передача по значению

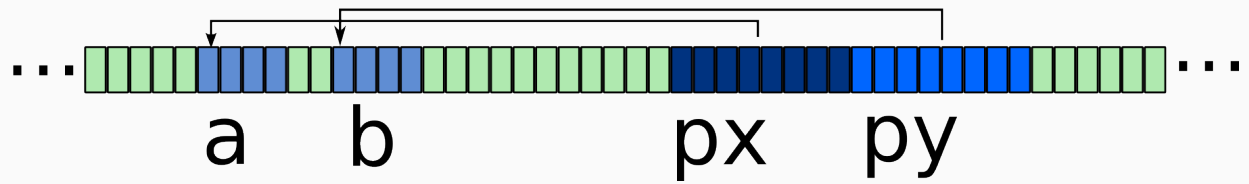
```
void swap(int x, int y)
{
    int  temp = x;
    x = y;
    y = temp;
}
...
swap(a, b);
```



Функция swap()

Передача по адресу

```
void swap(int* px, int* py)
{
    int  temp = *px;
   *px = *py;
   *py = temp;
}
...
swap(&a, &b);
```



Передача массивов в функцию

Автоматически передаются с помощью указателей

```
void add_num(int n, int arr[], int x)
{
    for (int i = 0; i < n; ++i)
        arr[i] += x;
}
int main()
{
    int arr[5] = {1, 5, 7, 3, 16};
    add_num(5, arr, 2);
}
```

Передача массивов в функцию

Автоматически передаются с помощью указателей

```
void add_num(int n, int* arr, int x)
{
    for (int i = 0; i < n; ++i)
        arr[i] += x;
}
int main()
{
    int arr[5] = {1, 5, 7, 3, 16};
    add_num(5, arr, 2);
}
```

Указатели

Итого

- Каждая переменная имеет адрес – номер ячейки памяти
- Указатель – это переменная, которая хранит адреса
- Обычные параметры передаются в функцию по значению
- Чтобы можно было изменять переменные в функциях нужно передавать указатели
- Массивы всегда передаются через указатели

Директивы и typedef

Директивы #include и #define

- **#include** — вставляет текст из указанного файла, например:

  `#incude <stdio.h>`

  Находит файл stdio.h и вставляет за место этой строчки.
- **#define** — задаёт макрос или символическую константу

  `#define SIZE 100`

  Перед компиляцией заменяет в тексте SIZE на 100

Typedef

- Typedef – ключевое слово в языке C
- Используется для того, чтобы дать типу новое имя
- ```
  typedef int my_new_int;
  ```

  ```
  typedef unsigned long long ull;
  ```
  ```

Структуры

Структуры

- Структура – это композитный тип данных, группирующий, без сокрытия набор значений

- ```
  struct book
  {
      char title[50];
      int pages;
      float price;
  };
  ```

```
int a;          // Объявление int
struct book b; // Объявление структуры book
```
```

Инициализация структуры:

```
struct book b1 = {"Don Quixote", 710, 900.0};
struct book b2 = {"War and Peace", 1500, 1200.0};
struct book b3 = {"The Dark Tower", 300, 500.0};
```

Доступ к элементу структуры(оператор .):

```
printf("%s costs %.2f R", b1.title, b1.price);
b3.price += 1000.0;
```

Typedef для структур

- Typedef – используется для того, чтобы дать типу новое имя
- ```
  typedef struct book Book;
  ```

```
Book b1 = {"Don Quixote", 710, 900.0};
Book b2 = {"War and Peace", 1500, 1200.0};
Book b3 = {"The Dark Tower", 300, 500.0};
```
```

Структуры передаются по значению, как и обычные переменные:

```
void print_book_info(Book b)
{
    printf("Book info:\n");
    printf("Title: %s\n", b.title);
    printf("Pages: %d\n", b.pages);
    printf("Price: %f\n", b.price);
}
```

Два варианта работы с указателем на структуру:

```
void change_price(Book* b, float new_price)
{
    (*b).price = new_price;
}
```

```
void change_price(Book* b, float new_price)
{
    b->price = new_price;
}
```

Массивы структур

```
Book wnp = {"War and Peace", 1500, 1200.0};
Book scifi_books[100] = {
        {"The Dark Tower", 300, 500.0},
        {"Fahrenheit 451", 400, 700.0},
        {"The Day of the Triffids", 304, 450.0}
};
scifi_books[3] = wnp;

change_price(&scifi_books[0], 1200.0);
print_book_info(scifi_books[0]);
```

Абстрактные типы данных: стек и очередь

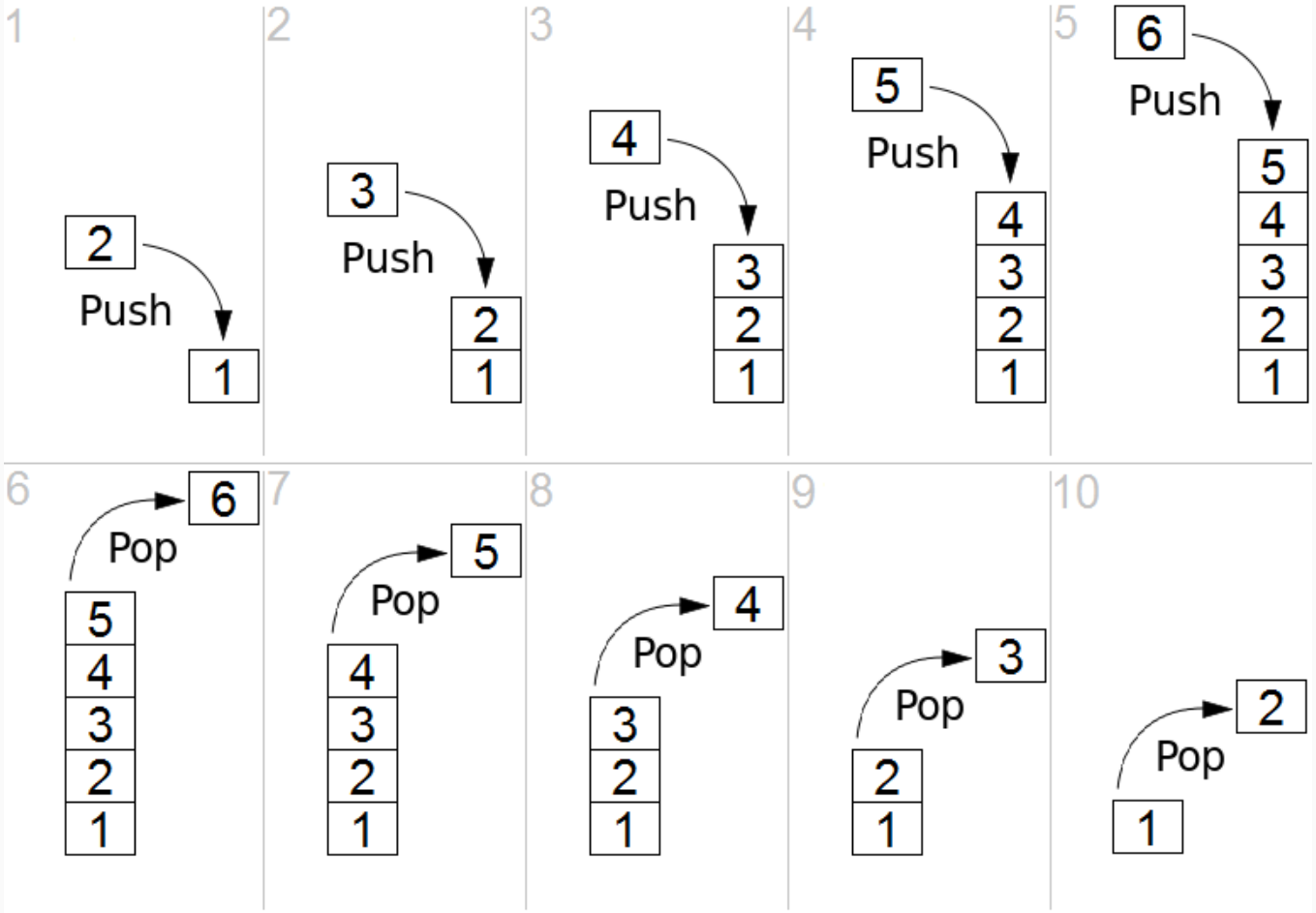
Стек

Стек(stack) – абстрактный тип данных, представляющий собой список элементов, организованных по принципу «последним пришёл — первым вышел».

Операции со стеком:

- **push** – добавляет элемент в вершину стека
- **pop** – удаляет элемент с вершины стека

Стек


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
Pop
6
5
4
3
2
1
7
Pop
5
4
3
2
1
8
Pop
4
3
2
1
9
Pop
3
2
1
10
Pop
2
1

Стек

Реализация стека в языке C с помощью массива

```
struct stack
{
    int n;
    int values[100];
};
typedef struct stack Stack;
```

Стек

Добавление элемента в стек (без проверки на размер)

```
struct stack
{
    int n;
    int values[100];
};
typedef struct stack Stack;

void stack_push(Stack* s, int x)
{
    s->values[s->n] = x;
    s->n += 1;
}
```

Стек

Удаление элемента из стека (без проверки на размер)

```
struct stack
{
    int n;
    int values[100];
};
typedef struct stack Stack;

int stack_pop(Stack* s)
{
    s->n -= 1;
    return s->values[s->n];
}
```

```
int main()
{
    Stack A;
    stack_create(&A);
    stack_push(&A, 5);
    stack_push(&A, 7);
    stack_push(&A, 3);
    stack_pop(&A);
    printf("%d", stack_pop(&A));
}
```

Функция stack_create() просто устанавливает A.n = 0

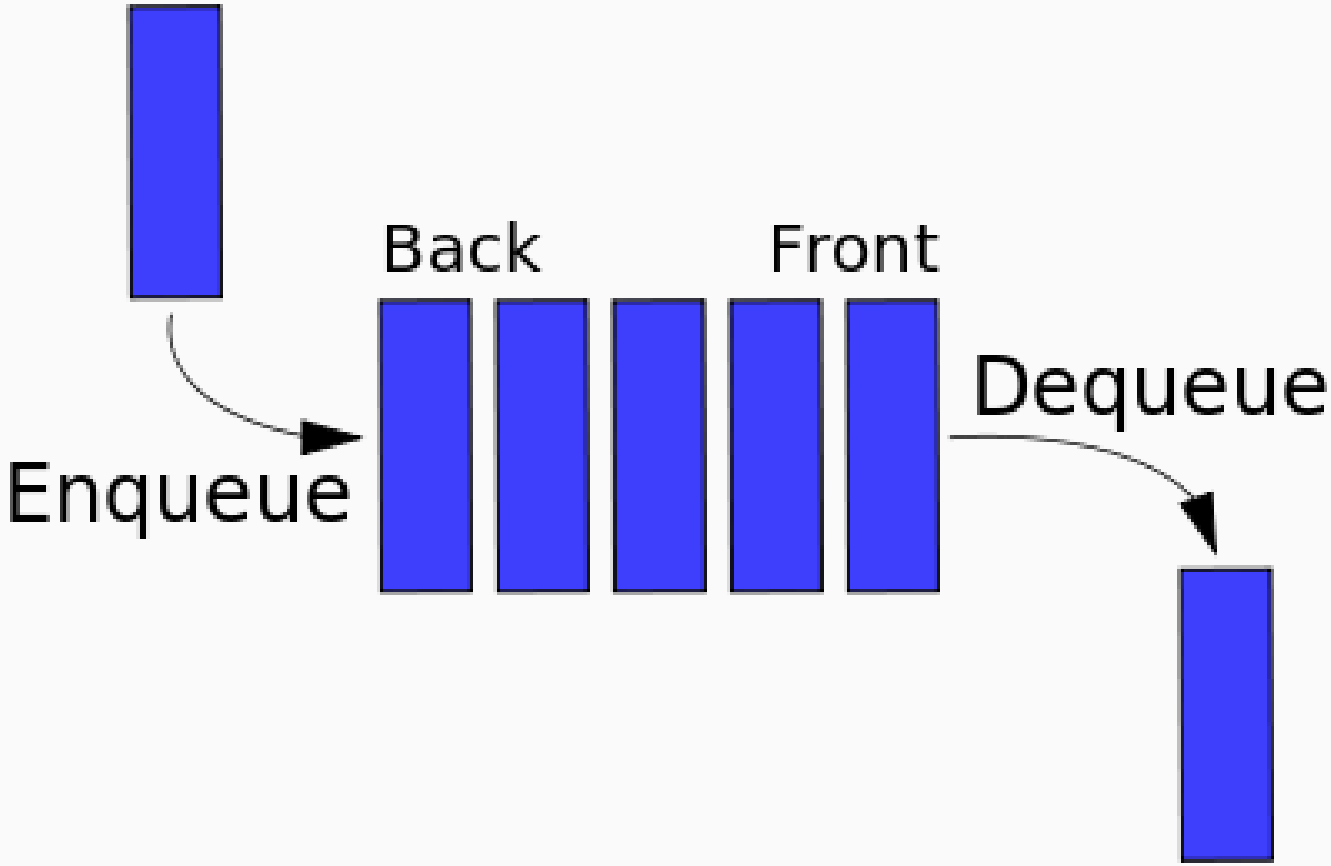
Очередь

Очередь (queue) – абстрактный тип данных, представляющий собой список элементов, организованных по принципу «первый пришёл — первый вышел».

Операции с очередью:

- **enqueue** – добавляет элемент в конец очереди
- **dequeue** – удаляет элемент с начала очереди

Очередь



Очередь

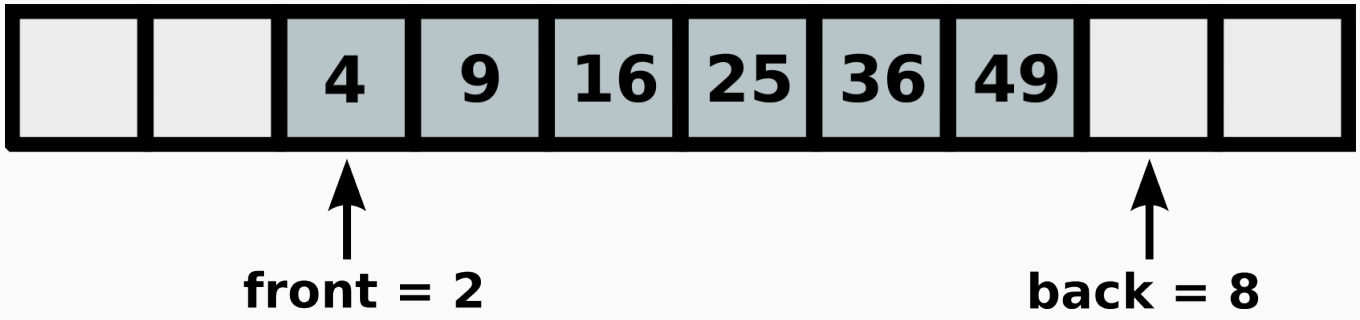
Реализация очереди в языке C с помощью массива

```
struct queue
{
    int front, back;
    int values[100];
};
typedef struct queue Queue;
```

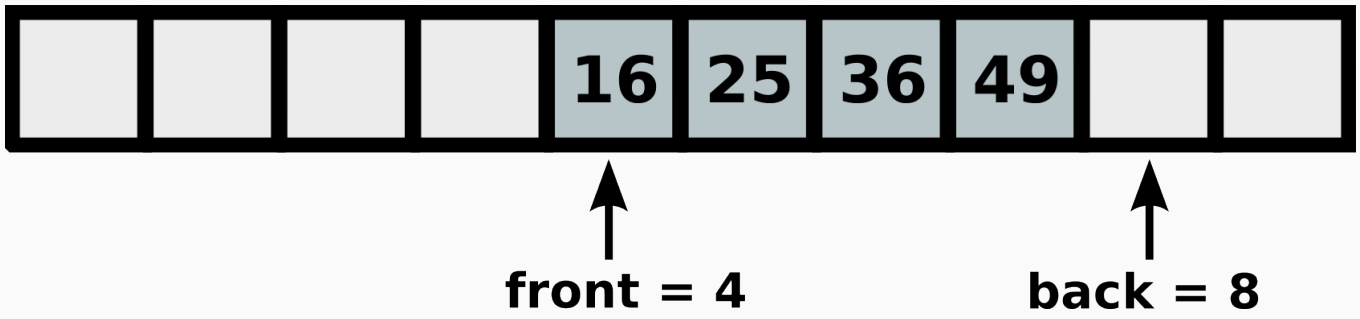
Очередь

Схема реализации очереди с помощью массива

```
Queue A;
queue_create(&A);
for (int i = 0; i < 8; ++i)
    enqueue(&A, i*i);
dequeue(&A);
dequeue(&A);
```



```
dequeue(&A);
dequeue(&A);
```



Очередь

Схема реализации очереди с помощью массива

```
enqueue(&A, 9);
enqueue(&A, 2);
enqueue(&A, 5);
```