

# Семинар №8

ФАКИ 2017

---

Бирюков В. А.

November 3, 2017

# Указатели и память

---

**int\***

**char\***

**unsigned long\***

**float\***

**void\***

Указатели разных типов хранят адреса (одно и то же)

Разные типы нужны, чтобы компилятор знал:

- 1) что вернуть при разыменовании
- 2) как проводить арифметику указателей

- Приведение типов `int` и `float`:

```
float x = 5.2;  
int y = (int)x;           // explicit  
int z = x;                // implicit
```

- Верно и для указателей:

```
float a;  
float* pf = &a;  
int* p1 = (int*)pf;       // explicit  
int* p2 = pf;             // implicit
```

```
int arr[4] = {1414547789, 10};
```



```
int* pi;
```



```
int arr[4] = {1414547789, 10};
```



```
int* pi = &arr[0];
```



# Указатели и память. Арифметика указателей.

```
int arr[4] = {1414547789, 10};
```



```
int* pi = &arr[0];  
pi += 1;
```



# Указатели и память. Арифметика указателей.

```
int arr[4] = {1414547789, 10};
```



```
int* pi = &arr[0];  
pi += 3;
```





```
int arr[4] = {1414547789, 10};
```



```
int* pi = &arr[0];  
*(pi + 3) == arr[3];
```



$\text{arr}[i] \quad \leftrightarrow \quad *(\text{arr} + i)$

Имя массива во многих случаях  
эквивалентно указателю на первый элемент

Указатель `char*` на `int`

---

```
int arr[4] = {1414547789, 10};
```



```
char* pc = &arr[0];
```



Примечание: при решении реальных задач присваивать указатели разных типов нежелательно.

```
int arr[4] = {1414547789, 10};
```



```
char* pc = arr;
```



```
int arr[4] = {1414547789, 10};
```



```
char* pc = (char*)arr;
```

```
int arr[4] = {1414547789, 10};
```



```
char* pc = (char*)arr;
pc += 3;
```

```
int arr[4] = {1414547789, 10};
```




```
char* pc = (char*)arr;   
pc += 8;
```



```
int arr[4] = {1414547789, 10};
```



```
char* pc = (char*)arr;   
printf("%s", pc);
```

Malloc = Memory allocation

---

# Функции для динамического выделения памяти

Нужно подключить библиотеку `stdlib.h`

- `void* malloc(size_t n)` – выделяет `n` байт и возвращает указатель `void*` на начало этой памяти
- `void free(void* p)` – освобождает выделенную память
- `void* realloc (void* p, size_t new_n)` – перевыделяет выделенную память

Если забудите освободить выделенную память произойдёт утечка памяти.

```
int* p = malloc(n);
```

```
// Используем p как массив  
// размера n/4
```

```
free(p);
```

```
int* p = malloc(n * sizeof(int));
```

```
// Используем p как массив  
// размера n
```

```
free(p);
```

```
int* p = (int*)malloc(n * sizeof(int));
```

```
// Используем p как массив размера n  
// Из void* -> int* (не обязательно)
```

```
free(p);
```

# Абстрактные типы данных: стек и очередь

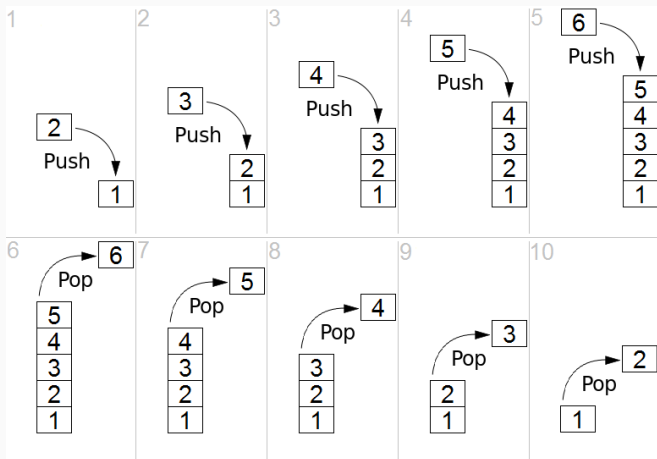
---

**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





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

## 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;
}
```

## 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];
}
```

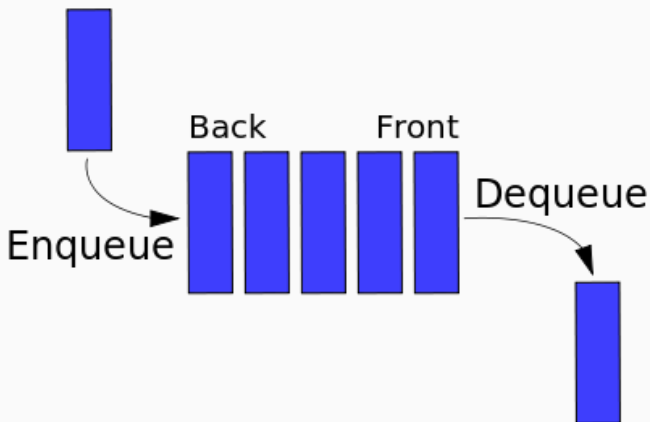
```
int main()  
{  
    Stack A;  
    stack_create(&A);  
    stack_push(&A, 5);  
    stack_push(&A, 7);  
    stack_push(&A, 3);  
    stack_pop(&A);  
    printf("%d", stack_pop(&A));  
}
```

Функция `stack_create()` просто устанавливает  $A.n = 0$

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

Операции с очередью:

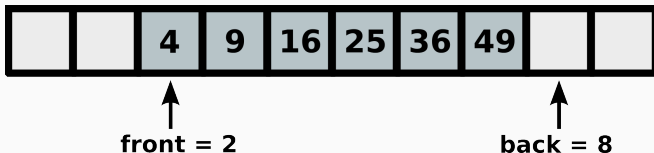
- **enqueue** – добавляет элемент в конец очереди
- **dequeue** – удаляет элемент с начала очереди



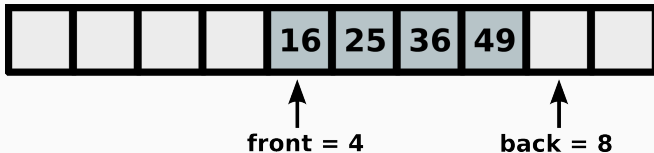
```
struct queue
{
    int front , back;
    int values[100];
};
typedef struct queue Queue;
```



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



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



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

