# 1. БАЗОВЫЕ ПОНЯТИЯ

| Концепция | Объяснение | Пример |
|---|---|---|
| **Указатель** | Переменная, хранящая адрес другой переменной | `int *ptr;` |
| **Адрес** | Номер ячейки памяти | `0x7ffd42a` |
| **Значение** | Данные в ячейке памяти | `42` |

# 2. ОПЕРАТОРЫ

| Оператор | Смысл | Пример |
|---|---|---|
| `&` | **Взять адрес** переменной | `&x` → адрес x |
| `*` | **Разыменовать** (получить значение по адресу) | `*ptr` → значение по адресу ptr |
| `->` | Доступ к полю структуры через указатель | `ptr->field` |

# 3. ОБЪЯВЛЕНИЕ УКАЗАТЕЛЕЙ

```
c
int *p;          // указатель на int
char *c;         // указатель на char
float *f;        // указатель на float
void *v;         // универсальный указатель (нельзя
разыменовать)
```

# 4. БАЗОВЫЕ ОПЕРАЦИИ

```c
int x = 42;
int *p = &x;    // p хранит адрес x

printf("Адрес x: %p\n", (void*)&x);     // 0x7ffd42a
printf("Значение p: %p\n", (void*)p);   // тот же адрес
printf("Значение x: %d\n", x);          // 42
printf("*p: %d\n", *p);                 // 42
(разыменование)

*p = 100;       // меняем x через указатель
printf("x = %d\n", x);  // 100
```

# 5. УКАЗАТЕЛИ И МАССИВЫ

```c
int arr[3] = {10, 20, 30};
int *p = arr;   // p указывает на arr[0]

// Эквивалентные записи:
arr[1] = 50;
*(arr + 1) = 50;
p[1] = 50;
*(p + 1) = 50;
```

# 6. АРИФМЕТИКА УКАЗАТЕЛЕЙ

```c
int arr[5] = {10, 20, 30, 40, 50};
int *p = arr;

p++;        // теперь p указывает на arr[1] (20)
p += 2;     // теперь p указывает на arr[3] (40)
p--;        // теперь p указывает на arr[2] (30)

// Разность указателей = количество элементов между ними
int diff = &arr[4] - &arr[1];  // diff = 3
```

# 7. УКАЗАТЕЛИ В ФУНКЦИЯХ

### Проблема (не работает):

```c
void increment_wrong(int a) {
    a++;  // меняется копия!
```

```
}
```

**Решение (работает):**

```
c
void increment(int *a) {
    (*a)++;  // меняем исходную переменную
}

int main() {
    int x = 5;
    increment(&x);  // передаем адрес
    printf("%d", x); // 6
}
```

# 8. ДИНАМИЧЕСКАЯ ПАМЯТЬ

| Функция | Назначение | Пример |
|---|---|---|
| `malloc()` | Выделить память | `malloc(10 * sizeof(int))` |
| `calloc()` | Выделить и обнулить | `calloc(10, sizeof(int))` |
| `realloc()` | Изменить размер | `realloc(ptr, 20 * sizeof(int))` |
| `free()` | Освободить память | `free(ptr)` |

**Пример:**

```
c
int n = 5;
int *arr = (int*)malloc(n * sizeof(int));

// ВСЕГДА проверяйте выделение памяти!
if (arr == NULL) {
    printf("Ошибка памяти!\n");
    return 1;
}
```

```
// Работа с массивом
for (int i = 0; i < n; i++) {
    arr[i] = i * 10;
}

// ОСВОБОДИТЬ когда не нужен!
free(arr);
arr = NULL;  // хороший тон
```

# 9. УКАЗАТЕЛИ НА УКАЗАТЕЛИ

```
c
int x = 42;
int *p = &x;    // p → x
int **pp = &p;  // pp → p → x

printf("x = %d\n", x);       // 42
printf("*p = %d\n", *p);     // 42
printf("**pp = %d\n", **pp); // 42

**pp = 100;  // меняем x через двойной указатель
printf("x = %d\n", x);  // 100
```

# 10. УКАЗАТЕЛИ И СТРОКИ

```
c
char str[] = "Hello";
char *p = str;

while (*p != '\0') {
    printf("%c", *p);
    p++;  // перемещаемся по строке
}
// Вывод: Hello
```

# 11. ЧАСТЫЕ ОШИБКИ

| Ошибка | Пример | Почему плохо |
| --- | --- | --- |
| **Неинициализированный указатель** | `int *p; *p = 5;` | Неопределенное поведение |

| | | |
|---|---|---|
| **Утечка памяти** | `malloc(100);` (без free) | Программа жрет память |
| **Висячий указатель** | `free(p); *p = 10;` | Обращение к освобожденной памяти |
| **Двойное освобождение** | `free(p); free(p);` | Крах программы |

# 12. ПРАВИЛА БЕЗОПАСНОСТИ

1. **Инициализируйте указатели** либо адресом, либо `NULL`
2. **Проверяйте** `NULL` перед использованием
3. **Каждому** `malloc()` **— свой** `free()`
4. **После** `free()` **обнуляйте указатель**: `ptr = NULL`
5. **Не используйте указатель после** `free()`