1. БАЗОВЫЙ СИНТАКСИС

```
c
// Прототип (объявление)
тип имя_функции(тип1 параметр1, тип2 параметр2);

// Определение
тип имя_функции(тип1 параметр1, тип2 параметр2) {
    // тело функции
    return значение; // если тип не void
}
```

Пример:

```
c
int sum(int a, int b);          // Прототип

int sum(int a, int b) {         // Определение
    return a + b;
}
```

2. ТИПЫ ФУНКЦИЙ

```
c
// 1. Без параметров и возврата
void say_hello() {
    printf("Hello!\n");
}

// 2. С параметрами, без возврата
void print_sum(int a, int b) {
    printf("Sum: %d\n", a + b);
}

// 3. С возвратом значения
int multiply(int a, int b) {
    return a * b;
}

// 4. С указателем как параметром
void change_value(int *ptr) {
    *ptr = 100;
}
```

3. ВЫЗОВ ФУНКЦИИ

```c
// Простой вызов
int result = sum(5, 3);

// Вызов без возврата
print_sum(10, 20);

// Вызов с указателем
int x = 5;
change_value(&x); // теперь x = 100
```

4. ПЕРЕДАЧА АРГУМЕНТОВ

По значению (копия) - НЕ меняет оригинал:

```c
void increment_wrong(int a) {
    a++; // копия изменилась, оригинал — нет
}

int main() {
    int num = 5;
    increment_wrong(num);
    printf("%d\n", num); // 5
}
```

По ссылке (указатель) - меняет оригинал:

```c
void increment_right(int *a) {
    (*a)++; // меняем оригинал
}

int main() {
    int num = 5;
    increment_right(&num);
    printf("%d\n", num); // 6
}
```

ВАЖНО: `(*a)++` - скобки обязательны! Иначе увеличится указатель.

5. ВОЗВРАТ ЗНАЧЕНИЙ

```c
// Возврат числа
```

```
int get_five() {
    return 5;
}

// Возврат указателя
int* get_address(int *ptr) {
    return ptr;
}

// Без возврата (void)
void print_error() {
    printf("Error!\n");
    // return не нужен
}

// Ранний возврат
int safe_divide(int a, int b) {
    if (b == 0) {
        printf("Division by zero!\n");
        return 0; // ранний выход
    }
    return a / b;
}
```

6. РЕКУРСИЯ

```
c
// Факториал
int factorial(int n) {
    if (n <= 1) return 1;      // базовый случай
    return n * factorial(n-1); // рекурсивный вызов
}

// Числа Фибоначчи
int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

7. ОБЛАСТЬ ВИДИМОСТИ

```
c
int global = 100; // глобальная - видна везде

void func() {
    int local = 50;       // локальная - только в func
```

```
    static int counter = 0; // статическая - сохраняется
между вызовами
    counter++;

    printf("local: %d\n", local);
    printf("global: %d\n", global);
    printf("Вызовов функции: %d\n", counter);
}
```

8. ПРОТОТИПЫ ФУНКЦИЙ

```
c
// ХОРОШО: прототипы в начале
int sum(int a, int b); // прототип
int multiply(int a, int b); // прототип

int main() {
    int x = sum(5, 3);    // компилятор знает о функции
    return 0;
}

int sum(int a, int b) {   // определение
    return a + b;
}

// ПЛОХО: без прототипа
int main() {
    int x = sum(5, 3);    // ОШИБКА: sum не объявлена
    return 0;
}

int sum(int a, int b) {   // слишком поздно
    return a + b;
}
```

9. ПОЛЕЗНЫЕ ШАБЛОНЫ

Шаблон 1: Функция-валидатор

```
c
int is_valid(int value) {
    if (value < 0 || value > 100) return 0;
    return 1;
}
```

Шаблон 2: Функция с несколькими возвратами

```c
int find_index(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) return i; // ранний возврат
    }
    return -1; // не найдено
}
```

Шаблон 3: Функция-утилита

```c
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

10. ТИПИЧНЫЕ ОШИБКИ

```c
// 1. Нет прототипа
// 2. Путают = и ==
if (x = 5) { ... } // ОШИБКА: присваивание, а не
сравнение

// 3. Забывают скобки для указателей
(*ptr)++; // ПРАВИЛЬНО
*ptr++;   // ОШИБКА: увеличивается указатель

// 4. Возврат указателя на локальную переменную
int* bad_func() {
    int local = 5;
    return &local; // ОПАСНО! local уничтожится
}

// 5. Бесконечная рекурсия
void infinite() {
    infinite(); // СТОП! Укажет базовый случай
}
```

11. КОРОТКИЕ EXAMPLES

```c
// 1. Минимум двух чисел
int min(int a, int b) {
    return (a < b) ? a : b;
```

```
}

// 2. Проверка четности
int is_even(int n) {
    return n % 2 == 0;
}

// 3. Длина строки (аналог strlen)
int my_strlen(char *s) {
    int len = 0;
    while (*s++) len++;
    return len;
}

// 4. Копирование строки
void my_strcpy(char *dest, char *src) {
    while ((*dest++ = *src++));
}
```