

Введение в С

Лекция 1

Процедурное программирование — это парадигма программирования, основанная на концепции вызова *процедур (функций)*.

Язык Си поддерживает процедурное программирование.

Парадигмы программирования

Императивное

при написании программы мы “говорим”
компьютеру: “Сделай это, потом это, потом...”

фокус на последовательности действий: **как
надо сделать**

программа — последовательность инструкций
в процессе изменяется состояние программы

Декларативное

при написании программы мы “говорим”
компьютеру: “Я хочу получить это...”

фокус на логике: **что надо сделать**

программа описывает логику
нет указания состояния программы

Парадигмы программирования

Императивное

```
nums = [1, 2, 3, 4, 5, 6]
```

```
result = []
```

```
i = 0
```

```
while i < len(nums):
```

```
    num = nums[i]
```

```
    if num % 2 == 0:
```

```
        doubled = num * 2
```

```
        result.append(doubled)
```

```
    i += 1
```

```
print(result)  # [4, 8, 12]
```

Парадигмы программирования

Декларативное

```
nums = [1, 2, 3, 4, 5, 6]
```

```
result = list(map(lambda x: x * 2,  
                  filter(lambda x: x % 2 == 0, nums)))
```

```
print(result)  # [4, 8, 12]
```

Императивное программирование

Структурное программирование

- программа – набор **блоков** `if... else`, `for`, `while`
- упорядоченное выполнение команд

Процедурное программирование

- программа разбивается на набор **процедур** (функций)
- **повторное использование**

Объектно-ориентированное программирование

- программа состоит из **объектов**
- объекты **объединяют** данные и поведение
- **инкапсуляция**, **наследование** и **полиморфизм**

Событийно-ориентированное программирование

- программа **отвечает** на внешние события

Структурное программирование

```
#include <stdio.h>

int main() {

    int numbers[] = {3, -1, 4, -5, 6};
    int sum = 0;

    for (int i = 0; i < 5; i++) {
        if (numbers[i] > 0) {
            sum += numbers[i];
        }
    }

    printf("Sum: %d\n", sum);
    return 0;
}
```

```
#include <stdio.h>

int main() {

    int numbers[] = {3, -1, 4, -5, 6};
    int sum = 0;
    int i = 0;

start:
    if (i >= 5) goto end;
    if (numbers[i] > 0) sum +=
numbers[i];
    i++;
    goto start;

end:

    printf("Sum: %d\n", sum);
    return 0;
}
```

Процедурное программирование

```
def input_data():  
    return [3, -1, 4, -5, 6]  
  
def calculate_positive_sum(numbers):  
    total = 0  
    for num in numbers:  
        if num > 0:  
            total += num  
    return total  
  
def output_result(sum_result):  
    print("Sum:", sum_result)  
  
def main():  
    nums = input_data()  
    result = calculate_positive_sum(nums)  
    output_result(result)  
  
main()
```

```
numbers = [3, -1, 4, -5, 6]  
sum_result = 0  
  
for num in numbers:  
    if num > 0:  
        sum_result += num  
  
print("Sum:", sum_result)
```


Объектно-ориентированное программирование

```
#include <iostream>
#include <vector>
using namespace std;

class NumberProcessor {
    vector<int> numbers;
public:    NumberProcessor(vector<int> nums):
numbers(nums) {}

    int sumPositives() {
        int sum = 0;
        for (int n : numbers) {
            if (n > 0) sum += n;
        }
        return sum;
    };
};

int main() {
    vector<int> nums = {3, -1, 4, -5, 6};
    NumberProcessor processor(nums);
    cout<<"Sum: "<<processor.sumPositives()<<endl;
    return 0; }
```

```
#include <iostream>
using namespace std;

int main() {
    int nums[] = {3, -1, 4, -5, 6};
    int sum = 0;

    for (int i = 0; i < 5; i++) {
        if (nums[i] > 0) sum += nums[i];
    }

    cout << "Sum: " << sum << endl;
    return 0;
}
```

Событийно-ориентированное программирование

```
import tkinter as tk

def on_click():
    nums = [3, -1, 4, -5, 6]
    result = sum(n for n in nums if n > 0)
    label.config(text=f"Sum: {result}")

root = tk.Tk()
button = tk.Button(root, text="Calculate",
command=on_click)
label = tk.Label(root, text="Click the button")
button.pack()
label.pack()
root.mainloop()
```

```
nums = [3, -1, 4, -5, 6]
result = sum(n for n in nums if n > 0)
print(f"Sum: {result}")
```

Немного истории

Рождение программирования

1945 — ЭНИАК (ENIAC, Electronic Numerical Integrator and Computer):

первая электронная вычислительная машина. Программы вводились **вручную** через переключатели и кабели.

1949 — появился ассемблер (**Assembly**) — язык, ближе к машинному коду, но с мнемониками. Существенно упростил программирование железа.

Программирование было тяжёлым, ошибки стоили дорого. Всё делалось на уровне «железа».

```
section .data
    msg     db      'Hello, world!', 0xA    ; Сообщение + перевод строки
    len     equ     $ - msg                ; Длина сообщения

section .text
    global _start

_start:
    ; write(1, msg, len)
    mov     eax, 4        ; Системный вызов write
    mov     ebx, 1        ; File descriptor: 1 (stdout)
    mov     ecx, msg      ; Указатель на сообщение
    mov     edx, len      ; Длина сообщения
    int     0x80          ; Вызов ядра

    ; exit(0)
    mov     eax, 1        ; Системный вызов exit
    xor     ebx, ebx      ; Код возврата 0
    int     0x80          ; Вызов ядра
```

Немного истории

Языки высокого уровня

1957 — появление **Fortran (IBM)**: первый язык высокого уровня для научных расчётов. Он сильно упростил запись формул.

1958 — **LISP** и **ALGOL** (основа будущих языков): впервые ввели понятия **блоков кода**, **области видимости**, **рекурсии**.

1959 — **COBOL** для бизнеса: приближен к английскому языку, удобен для отчётности и управления данными.

Языки стали более «человекоориентированными», но всё ещё громоздкими, строго типизированными и далекими от системного программирования.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. HELLO.
```

```
PROCEDURE DIVISION.  
    DISPLAY "Hello, world!".  
    STOP RUN.
```

Немного истории

Процедурное программирование и кризис сложности (1960-е)

ALGOL 60 стал академическим эталоном. На его основе появились: **Pascal**, **Modula**, **Ada**.

BASIC (Beginner's All-purpose Symbolic Instruction Code) (1964) — простой язык для обучения. Работал на терминалах, получил широкое распространение в университетах.

Возникла идея **структурного программирования** (Дейкстра): без **goto**, с чёткими блоками **if**, **while**, **for**.

Языки становились чище, код — структурированным. Но большинство были высокоуровневыми и не подходили для низкоуровневого системного программирования.

```
program HelloWorld;  
  
begin  
    writeln('Hello,  
world!');  
end.
```

```
10 PRINT "Hello, world!"  
20 END
```

Много истории

Появление UNIX

— **Bell Labs** (часть AT&T) пытались участвовать в разработке Multics — масштабной ОС. Проект оказался перегруженным и сложным

— Разочарованные разработчики **Кен Томпсон** и **Деннис Ритчи** решили сделать свою ОС: простую, переносимую, компактную

— В 1969 году они разработали UNIX. Первая версия была написана Томпсоном на ассемблере для миникомпьютера PDP-7. Такая реализация была рабочей, но **непереносимой**

— Для облегчения программирования Томпсон создал язык **B** (упрощённый потомок **BCPL (Basic Combined Programming Language)**), но он имел ограничения по работе с **типами данных** и **памятью**, поэтому Деннис Ритчи в **1972** году разработал улучшенный язык — **C**. Были добавлены **статическая типизация, указатели, структуры** и **операции с памятью**.

Много истории

Революция

— Целью разработки **C** было создание **переносимого** языка **низкого уровня**, который позволял писать операционную систему, **не привязывая код к конкретному железу**, но при этом сохраняя контроль и эффективность, близкую к ассемблеру

— UNIX был постепенно переписан на **C** в 1973–1975 годах. Сначала частично (ядро), а позже почти полностью. Это сделало систему переносимой и стимулировало её распространение

— При этом **C** стал базовым языком для **системного программирования**, **драйверов**, **компиляторов** и **операционных систем** благодаря своей производительности, минималистичности и низкоуровневому контролю через **указатели** и **управление памятью**

— Влияние UNIX и **C** огромно: они заложили основы многих современных ОС (**Linux**, **MacOS**, **Windows**) и языков программирования (**C++**, **Java**, **Python**), а философия UNIX продолжает влиять на проектирование ПО

Где используется C

- Операционные системы: ядра **Unix** и **Linux**, части **Windows** и **macOS**.
- Системы управления базами данных: **Oracle Database**, **MySQL**, **SQLite**, **Microsoft SQL Server**, **PostgreSQL**.
- Компиляторы и инструменты разработки: **GCC** (GNU Compiler Collection).
- Графические библиотеки: **OpenGL**, **DirectX**.
- **Doom**, **Quake**, которые заложили основы 3D-графики.
- Научные программы: **MATLAB** и **Mathematica** используют **C** для вычислительных частей.
- Популярные языки программирования, которые изначально написаны на **C** или используют его компиляторы: **C++**, **Objective-C**, **Python**, **Ruby**, **PHP**, **Perl** и другие.
- **Встраиваемые системы и драйверы устройств**, где важна производительность и прямое управление оборудованием.

Достоинства и недостатки

Достоинства

Гибкость

Компактность

Переносимость

Эффективность

Мощность

Универсальность

Недостатки

Трудность в освоении

Отсутствие автоматического управления
памятью

Слабая типизация

Первая программа

```
#include <stdio.h>  // подключение библиотеки стандартного ввода/вывода

int main()          // основная функция программы
{
    puts("Hello, world!");
    return 0;
}
```

```
$ ./hello
Hello, world!
```

```
// hello.c
```

```
#include <stdio.h>
```

```
int main() {
```

```
    printf("Hello, world!\n");
```

```
    return 0;
```

```
}
```

Препроцессор

```
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4

/* stdio.h – стандартный заголовок */
#ifndef _STDIO_H
#define _STDIO_H 1

typedef struct _IO_FILE FILE;

extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;

extern int printf(const char *format, ...);

#endif /* _STDIO_H */.....

# 2 "hello.c" 2

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Компилятор

```
.file    "hello.c"
.text
.globl   main
.type    main, @function
main:
    pushq    %rbp
    movq     %rsp, %rbp
    leaq     .L.str(%rip), %rdi
    call     printf
    movl     $0, %eax
    popq     %rbp
    ret
.L.str:
    .string  "Hello, world!\n"
```

Ассемблер

0000000000000000 <main>:

0:	55	push	%rbp
1:	48 89 e5	mov	%rsp,%rbp
4:	48 8d 3d 00 00 00 00	lea	0x0(%rip),%rdi
b:	e8 00 00 00 00	callq	10 <main+0x10>
10:	b8 00 00 00 00	mov	\$0x0,%eax
15:	5d	pop	%rbp
16:	c3	retq	

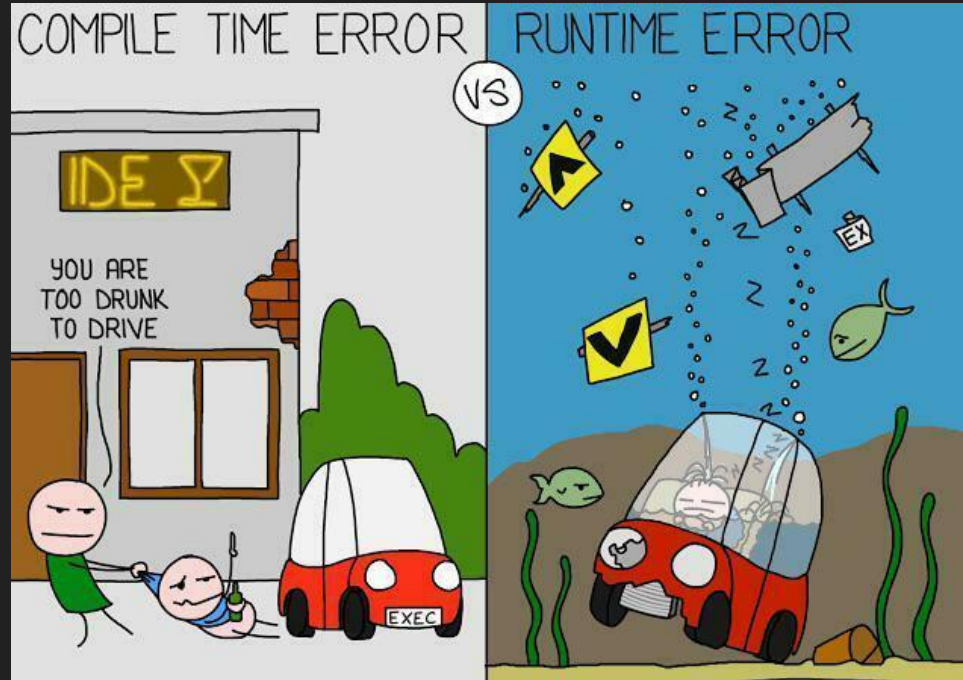
ЛИНКОВЩИК

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00 |.ELF.....|
00000010  02 00 3e 00 01 00 00 00  00 00 40 00 00 00 00 00 |..>.....@....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 40 00 38 00 |.....@.8.|
...
```

Этапы жизненного цикла программы

build-time — до выполнения программы. Выполняются директивы препроцессора, компиляция программы, сборка исполняемого файла.

run-time — во время выполнения программы. Выделяется и инициализируется память, выполняется код программы.



main

- Любая программа на C содержит функцию `main`
- Функция `main` может быть в любом месте программы
- Выполнение любой программы начинается с функции `main`
- Функция `main` должна быть только одна

варианты объявления `main`

```
int main()
```

```
int main(int argc, char * argv[])
```

Тело функции

- Заключается в фигурные скобки
- Содержит операторы — инструкции для выполнения

```
{  
    puts("Hello, world!");    // выводит "Hello, World!" в консоль  
    return 0;  
}
```

Комментарии

Однострочные (предпочтительнее)

//комментарий

Многострочные

*/*комментарий на 1 строке
комментарий на 2 строке*/*

Комментарии нужны

Заглавный комментарий. Размещайте заглавный комментарий, который описывает назначение файла, вверху каждого файла.

Заголовок функции. Разместите заголовочный комментарий на каждой функции вашего файла. Заголовок должен описывать поведение и/или цель функции.

Параметры / возврат. Если ваша функция принимает параметры, то кратко опишите их цель и смысл. Если ваша функция возвращает значение — кратко опишите, что она возвращает.

Комментарии на одной строке. Если внутри функции имеется секция кода, которая длинна, сложна или непонятна, то кратко опишите её назначение.

НО...

Сложный код, написанный с использованием хитрых ходов, следует не комментировать, а переписывать!

Следует делать как можно меньше комментариев, делая код **самодокументируемым** путём выбора правильных имён и создания ясной логической структуры.

Переменные

Переменная — это **именованная** область памяти, предназначенная для хранения данных, которые могут изменяться в процессе работы программы.

- имя
- тип
- значение
- адрес
- время жизни
- область видимости

Идентификатор — это последовательность символов, используемая для обозначения одного из следующих элементов:

- Имени объекта или **переменной**
- Имени структуры или объединения
- Имени перечисленного типа
- Члена структуры, объединения или перечисления
- Функции
- Имени определения типа (typedef)
- Имени метки
- Имени макроса
- Параметра макроса

Соглашение по именованию

смешанный регистр, начиная с нижнего

по-английски

большая область видимости — длинное имя,
небольшая область видимости — короткое имя

Префикс **n** для количества объектов

Суффикс **No** для обозначения номера сущности

Префикс **is** только для логических переменных и методов (можно ещё **can**, **has**, **should**)

```
camelStyle;
```

```
averageTemperature;
```

```
nStudents;
```

```
invoiceNo;
```

```
isConnected;
```

Типы данных

определяет:

- **как** данные представлены в памяти
- **объем** памяти под данные
- **множество значений** величины этого типа
- **операции и функции**, применяемые к данным

Целочисленные типы

Тип	Минимальный размер (байт)	Диапазон значений (знаковый)	Диапазон значений (беззнаковый)
char	1	обычно от -128 до 127	от 0 до 255
short	2	от -32768 до 32767	от 0 до 65535
int	обычно 4	от -2,147,483,648 до 2,147,483,647	от 0 до 4,294,967,295
long	минимально 4	от -2^{31} до 2^{31} или от -2^{63} до $2^{63}-1$	от 0 до $2^{32}-1$ или выше
long long	обычно 8	примерно от $-9 \cdot 10^{18}$ до $9 \cdot 10^{18}$	от 0 до $2^{64}-1$

Примеры

```
int a = -20;  
  
unsigned int ub = 4000000000;  
  
char letter = 'A';  
  
char c = 65;  
  
short temp = -32000;  
  
int count = 100000;  
  
unsigned int uCount = 5000000;  
  
long long big = 10000000000000;
```

int

```
int a = -20;
```

- Переменная типа `int` хранит целое число.
- Компилятор выделяет под неё столько памяти, сколько нужно для хранения одного целого значения.
- Точные границы (`int` от ... до ...) зависят от компьютера, компилятора и ОС. Обычно `int` занимает 4 байта: от -2,147,483,648 до 2,147,483,647
- Знак числа хранится в старшем бите:
 - `0` — положительное число (или ноль)
 - `1` — отрицательное число
- Из-за этого отрицательных чисел на 1 меньше, чем положительных.
- Если увеличить максимальное значение, получится минимальное.
- Если уменьшить минимальное, получится максимальное.
- Тип `int` условно можно представить как сомкнутое кольцо чисел.

Числа с плавающей точкой

Тип	Размер (байт) может зависеть от платформы	Точность (знаков)
float	4	~6–7
double	8	~15–16
long double	8–16 (зависит от компилятора)	~18–21

float

Пример:

число: 5.75

в двоичном формате: 101.11

нормализованная двоичная форма:

1.0111 × 10²

Знак = 0 (положительное)

Экспонента = 2 + 127 = 129 → 10000001

(со смещением на 127)

Мантисса = 011100000000000000000000 (без первой единицы!)

01000000 10111000 00000000 00000000

Величина с модификатором типа float занимает 4 байта

1 бит отводится для знака

8 бит для экспоненты

23 бита для мантиссы

Старший бит мантиссы всегда равен 1, поэтому он не заполняется.

char

Тип `char` хранит один символ (букву, цифру, пробел и др.)

В памяти символы представлены **в виде целых чисел**

Соответствие символов и чисел задаётся таблицей кодировки, которая зависит от системы.

Самая распространённая таблица — `ASCII`.

Таблица ASCII содержит латинские буквы (прописные и строчные), цифры 0–9 и специальные символы.

Символ новой строки	<code>\n</code>
---------------------	-----------------

Горизонтальная табуляция	<code>\t</code>
--------------------------	-----------------

Вертикальная табуляция	<code>\v</code>
------------------------	-----------------

Возврат на шаг	<code>\b</code>
----------------	-----------------

Возврат каретки	<code>\r</code>
-----------------	-----------------

Обратная косая	<code>\\</code>
----------------	-----------------

Одиночная кавычка (апостроф)	<code>\'</code>
------------------------------	-----------------

Двойные кавычки	<code>\"</code>
-----------------	-----------------

Звонок	<code>\a</code>
--------	-----------------

Правила объявления переменных

```
double total = 0.0;  
double speed = 3.0e8;  
int nLines = 15, columnNo = 25;  
int isEmpty = 0;
```

Хорошо!!!
Переменные инициализируются
при объявлении

```
double total;  
double speed;  
int nLines, columnNo;  
int isEmpty;
```

Плохо!!!
Объявление без инициализации

Модель памяти компьютера

- Память **дискретна**, состоит из отдельных **байтов**
- Каждый байт **пронумерован**, номер байта называется **адресом**
- Минимально доступный программисту участок памяти – один байт
- Переменная может занимать больше одного байта
- Обычная переменная не может занимать меньше одного байта
- Все байты, занимаемые переменной, идут **подряд**
- **Адрес переменной** – адрес старшего байта (или младшего)

Время жизни и область видимости

Время жизни – это время, в течение которого переменная **связана** с определенной областью памяти. Определяется классом памяти.

Может быть:

- локальным
- глобальным

Область видимости – это блок программы, из которого можно **обратиться** к переменной.

Может быть:

- блок
- функция
- файл
- вся программа

Область видимости и время жизни

```
#include <stdio.h>

int main() {
    int x = 10;                // переменная видна во всей функции main
    if (x > 5) {                // всё хорошо: x видна во всей main
        int y = 20;            // у видна только внутри этого блока if
        printf("%d\n", y);      // работает
    }                           // переменная у уничтожается (время жизни закончилось)
    printf("%d\n", y);          // ❌ ошибка: у здесь не видна
    printf("%d\n", x);          // всё хорошо: x видна во всей main
    return 0;
}
```

Ввод и вывод

- В **C** нет встроенных средств ввода/вывода
- Для ввода/вывода используются функции **стандартных библиотек**
- Стандартное устройство ввода – клавиатура (**stdin**)
- Стандартное устройство вывода – терминал (**stdout**)

Функции ввода

- `scanf(<шаблон>, <адреса>)`
- `getchar()`
- `gets(<адрес>)`

- `fscanf(<поток>, <шаблон>, <адреса>)`
- `fgetc(<поток>)`
- `fgets(<адрес>, <размер>, <поток>)`

Функции вывода

- `printf(<шаблон>, <данные>)`
- `putchar(<символ>)`
- `puts(<строка>)`

- `fprintf(<поток>, <шаблон>, <данные>)`
- `fputc(<символ>, <поток>)`
- `fputs(<строка>, <поток>)`

Получение данных от пользователя

```
int a = 0, b = 0;  
float f = 0.0;  
double d = 0.0;  
scanf("%d %d %f %lf", &a,  
&b, &f, &d);
```

Вводит int, float, double

Хорошо

```
#define SIZE 256  
char str[SIZE] = {0};  
fgets(str, SIZE, stdin);
```

Указан размер, защита от переполнения

Хорошо

```
#define SIZE 256  
char str[SIZE] = {0};  
gets(str);
```

Может выйти за границы массива

Плохо

```
#define SIZE 256  
char str[SIZE] = {0};  
scanf("%s", str);
```

Чтение до первого пробела, возможен ввод не всей строки

Плохо

Основные шаблоны функции `scanf`

<code>%c</code>	Ввод одного символа
<code>%d</code>	Ввод десятичного целого числа (со знаком)
<code>%u</code>	Ввод беззнакового целого числа
<code>%i</code>	Ввод десятичного числа , также понимает <code>0</code> и <code>0x</code>
<code>%o</code>	Ввод восьмеричного числа
<code>%x</code>	Ввод шестнадцатеричного числа
<code>%lld</code>	Ввод числа типа long long
<code>%f</code>	Ввод числа с плавающей точкой типа float
<code>%lf</code>	Ввод числа с плавающей точкой типа double
<code>%s</code>	Ввод строки до первого пробела или табуляции

```
int a = 0;
unsigned int b = 0;
char ch = ' ';
scanf("%d %u %c", &a, &b, &ch);
```

```
float f = 0.0;
double d = 0.0;
scanf("%f %lf", &f, &d);
```

Программа-приветствие пользователю

```
#include <stdio.h>
#define SIZE 100

int main() {
    char str[SIZE] = {0};
    puts("What's your name?");
    fgets(str, SIZE, stdin);
    printf("Hello, %s!\n", str);
    return 0;
}
```

Работа программы:

What's your name?

Alice

Hello, Alice!