??????????? C++

- C: общее
- C: стандартные функции
- C: переменные

C: ?????

[Исходный код из книги по C]

P. J. Plauger’s The Standard C Library - желательная для прочтения.

Поиск нужного пакета:

```
apt-cache search manpages | grep ^manpag
```

Установка пакетов

```
sudo apt install manpages-ru manpages-ru-dev
```

Список доступных локалей

```
locale -a
```

Установка локали для man страницы

```
LANG=ru_RU.utf8 man 2 open
```

Компиляция

```
cc test.c
cc -0 test test.c
```

Вариант через gcc

```
gcc -Wall -Wextra -Werror -g program.c -o my_program
```

Добавить поиск в текущей директории (чтобы не писать ./)

```
export PATH=$PATH:.
```

Базовое описание языка

```
1 // A first program in C.
2 #include <stdio.h>
3
```

```
// function main begins program execution
int main(void) {
   printf("Welcome to C!\n");
}
```

- директивы препроцессора.

*.h заголовочные файлы, в которых содержится описание функций и ссылки на файлы с их расположением.

Константы

добавляется слово const

```
const int var=7;
```

Комментарии

```
// comments
/* comments
long */
```

Циклы

Цикл for:

```
for (j=0; j<9; j++) {}
```

Цикл с предусловием:

```
while (i < 5) {}
```

Цикл с послеусловием:

```
do {
    // тело цикла
} while (условие);
```

break — немедленный выход из цикла
continue — переход к следующей итерации (пропуск остатка тела)

Функции

Шаблон описания

```
тип_возвращаемого_значения название(входящие аргументы) {

}

int add(int a, int b);     // Функция, возвращающая int
void print_message();      // Функция, не возвращающая значение
int (*func_ptr)(int, int); // Указатель на функцию
```

void - отсутствие аргумента

Функция int main(void) - точка входа.

Получение нескольких значений из функции

Передача указателя

```
#include <stdio.h>

// Функция возвращает int, а через указатели изменяет float
int get_values(int *integer_part, float *float_part) {
    *integer_part = 42;      // Записываем по адресу
    *float_part = 3.14;      // Записываем по адресу
    return 1;                // Статус: успешно
}

int main() {
    int num;
    float pi;

    int status = get_values(&num, &pi);  // Передаем адреса!

    printf("Целое: %d\n", num);   // 42
    printf("Дробное: %f\n", pi);  // 3.140000
    printf("Статус: %d\n", status); // 1

    return 0;
}
```

Структура

```
#include <stdio.h>

// Определяем структуру для хранения двух значений
typedef struct {
    int integer;
    float decimal;
} TwoValues;

// Функция возвращает структуру
TwoValues get_values() {
    TwoValues result;
    result.integer = 42;
    result.decimal = 3.14;
    return result;  // Возвращаем структуру целиком
}

int main() {
    TwoValues values = get_values();

    printf("Целое: %d\n", values.integer);   // 42
    printf("Дробное: %f\n", values.decimal); // 3.140000

    return 0;
}
```

Глобальные переменные

Массив

```
#include <stdio.h>

void get_values(int arr[]) {
    arr[0] = 42;
    arr[1] = 100;
    arr[2] = 255;
}

int main() {
    int values[3];
    get_values(values);  // Массив передается по указателю
```

```
    for (int i = 0; i < 3; i++) {
        printf("%d ", values[i]);  // 42 100 255
    }

    return 0;
}
```

C: ??????????? ???????

stdio.h

Модуль ввода/вывода.

printf() Форматированный вывод на консоль.

```
#include <stdio.h>


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

Первый аргумент — шаблон вывода, затем аргументы шаблона (если есть). Функция может принимать переменное число аргументов. При успешном выполнении возвращает кол-во выведенных символов, ошибки — отрицательное значение.
Формат-строка может содержать обычные символы и спецификаторы формата, начинающиеся с символа %. Они указывают, как выводить следующие за строкой аргументы. Примеры спецификаторов:

%c	вывод отдельного символа (char)
%s	вывод строки (char*)
%c	один символ
%d или %i	вывод целого числа со знаком (int)
%u	Целое число без знака (unsigned int) в десятичной системе
%f или %F	вывод числа с плавающей точкой (float, double)
%o	вывод в восьмеричной системе
%x или %X	вывод в шестнадцатеричной системе (строчными и заглавными буквами соответственно)
%%	Символ процента %

Модификаторы типов:
h — для short int (%hd).
l — для long int (%ld) или double (%lf).
L — для long double (%Lf).

Управляющие последовательности внутри строки формата тоже обрабатываются:
\n — перевод на новую строку;
\t — табуляция;
\\ — обратный слеш;

\" — двойная кавычка.

```
printf("Привет, %s!", "мир");  // Выведет «Привет, мир!»
```

Дополнительные возможности форматирования

Флаги:

-	выравнивание по левому краю
+	принудительный вывод знака
#	альтернативное представление для некоторых типов

Ширина поля:

%5d — число будет выведено в поле шириной в 5 символов (по умолчанию — выравнивание по правому краю).
Точность: %.2f — число с плавающей точкой будет выведено с 2 знаками после запятой.
Модификатор размера: %ld (длинное целое), %lld (длинное длинное).

```
printf("%5d", 123);   // '   123'
printf("%-5d", 123);  // '123 ' (выравнивание по левому краю)
printf("%05d", 12);   // '00123' (ведущие нули)
printf("%+8.2f", 3.14159);  // ' +3.14' (знак и ширина)
```

scanf() Считывание данных из стандартного потока ввода (обычно с клавиатуры) и сохранение в переменных. Работает по принципу форматированного ввода: вы указываете шаблон (форматную строку), а функция интерпретирует введённые данные в соответствии с ним.

```
int scanf(const char *format, ...);
```

Возвращает количество успешно прочитанных и присвоенных значений. Если происходит ошибка или конец файла достигнут, возвращается EOF. Однако если например вместо ожидаемой цифры будет введен символ, то он останется в буфере.

Передача адресов переменных
Ключевой момент: функция scanf записывает данные в память, и ей нужно передать адреса переменных, куда эти данные сохранятся. Для этого перед именем переменной ставится амперсанд & (операция взятия адреса).

```
int num;
scanf("%d", &num); // Передаём адрес переменной num
```

Пример программы

```
#include <stdio.h>

int main() {
   int age;
   printf("Введите возраст: ");
   if (scanf("%d", &age) != 1) {
      printf(stderr, "Ошибка ввода");
      return 1;
      }
   printf("Вы ввели: %d", age);
   return 0;
}
```

Интересные вещи возникают при неправильном вводе.

```
#include <stdio.h>

int main() {
   int age;
   char my;
   printf("Введите возраст: ");
   if (scanf("%d", &age) != 1) {
      printf("Введено: %d\n", age);
      scanf("%c", &my);
      printf("Введено: %c\n", my);
      return 1;
      }
   printf("Вы ввели: %d\n", age);
   return 0;
}
```

Результат:

```
Введите возраст: tre
Введено: 32765
Введено: t
```

- Если переменная не инициализирована, в ней будет какой-то мусор (в данном случае 32765)

- После несоответствия формата ожидаемого ввода, переменная не меняется. То есть в age остается мусор.
- Ввод попадает в буфер, и только потом считывается scanf.
- Если в буфере будет больше элементов, чем ожидается, то в буфере останется остаток. Он будет считан при последующем вызове функции scanf.
- \n считается символом, поэтому если последовательно получать число и символ, а ввести число и нажать Enter то во вторую переменную попадет \n. То есть буфер после считывания не очищается.

?: ??????????

Переменные бывают локальные (в пределах функции) и глобальные. Глобальные определяются до функции main вне остальных функций.

Переменные можно просто объявлять и дополнительно инициализировать начальное значение. Размеры типов зависят от платформы и компилятора! Вывод размера:

```
sizeof(имя_типа);
```

Квалификаторы типов (модификаторы)

Они не создают новые типы, но изменяют их свойства.
const Значение нельзя изменить const int x = 5;
volatile Значение может измениться внешне (прерывания, hardware) volatile int status;
restrict Указатель — единственный способ доступа к данным (для оптимизации) int *restrict ptr;
signed Тип со знаком (по умолчанию для char не определено) signed char c;
unsigned Тип без знака (только положительные числа) unsigned int u;

Псевдонимы типов (typedef)

Позволяет дать новое имя существующему типу.

```
typedef int my_int;           // my_int — это псевдоним для int
typedef unsigned long ulong;  // ulong — псевдоним для unsigned long

my_int x = 10;                // То же, что int x = 10;
ulong y = 100;                // То же, что unsigned long y = 100;
```

Часто используется для упрощения сложных типов:

```
typedef struct {
    char name[50];
    int age;
} Person;                    // Теперь можно писать Person p; вместо struct Person p;
```

Базовые типы:

void	Нет значений Означает "отсутствие типа"	Нет размера	

integer (int)	Целые числа, -2,147,483,648 … 2,147,483,647 Зависит от разрядности ОС!	4	`int var;` `int var=7;`
unsigned int	Беззнаковое целое 0 … 4,294,967,295	4	
short	Числа со знаком -32,768 … 32,767	2	
unsigned short	Беззнаковое короткое целое 0 … 65,535	2	
long		4 или 8 Зависит от платформы	
unsigned long			
long long	Очень длинное целое	8 байт	
unsigned long long		8	
float	Дробные с точностью до 7 знака	4	`float var_float = 12.45;`
double	Дробные с точностью 15 знаков	8	
long double		10, 12 или 16 Зависит от платформы	
bool	Логическое		`bool vr = true;`
char	Символ Желательно указывать явно signed / unsigned	1	`char n = 'g';`
unsigned char	Беззнаковый символ 0 … 255	1	

Массивы

Хранят несколько элементов одного типа подряд в памяти. Из базовых.

```
int numbers[10];            // Массив из 10 целых чисел
char name[50];              // Массив из 50 символов
```

```
double matrix[3][3];       // Двумерный массив (матрица 3x3)
```

Указатели (pointer)

Хранят адрес в памяти, где находится значение. Для обращения по адресам необходимы:

1. Возможность находить адрес, по которому в памяти располагается та или иная переменная;
2. Место, где этот адрес можно было бы сохранить для дальнейшей работы;
3. Возможность обращаться к ячейке памяти по ее адресу.

Операция получения адреса переменной & (амперсанд). Хранятся адреса переменных тоже в переменных, в специальном типе,
переменные указатели. Количество указательных типов, равно количеству типов данных.
Объявление переменной-указателя

```
int *ptr, i;               // Указатель на int и обычный int
char *str;                 // Указатель на char (строка)
void *generic;             // Универсальный указатель (на любой тип)
```

Для изменения / получения данных из адреса оператор разыменовывания * но в операциях с переменной:

```
char i = 0, *p;
p = &i;
*p = 2;
i = i + 2;
printf("%d %d", *p, i);
```

Но если попытаться напрямую изменить адрес, например p=p+2; то будет ошибка (такого адреса нет).

В ячейке памяти хранится адрес первой ячейки памяти. Тип переменной указывается для понимания, сколько байт нужно считывать.

Перечисления (enum)

Определяют набор именованных целочисленных констант.

```
enum Color {
    RED,    // 0
    GREEN,  // 1
```

```
    BLUE    // 2
};

enum Color favorite = GREEN;
```

Структуры

Объединяют разные типы в один.

```
struct Person {
    char name[50];
    int age;
    float height;
};

struct Person person1;
```

Объединения (union)

Как структура, но все поля используют одну и ту же область памяти (экономят память).

```
union Data {
    int i;
    float f;
    char str[20];
};

union Data data;           // data.i, data.f, data.str занимают одну память
```

Преобразование типов.

Бывают явные (делает программист) и неявные (компилятор). Неявные делятся на:

1. При присваивании результата вычисления переменной, тип результата к преобразуется к типу переменной. Округление происходит не по математическим законам, а путем отбрасывания значения.
2. При вычислении выражения операнды каждой операции преобразуются к какому-то одному типу (если они различаются), при этом результат операции будет того же типа. Правило «меньший» тип к «большему», для минимизации потерю информации. Операции преобразования сначала выполняются от самого глубокого до внешнего, одного приоритета слева направо.

```
float c = 5/6;
```

5 и 6 - целочисленные, поэтому результат целочисленный. И равен 0. Только после этого он преобразуется в float, но уже поздно. В с будет 0. А если сделать один операнд с плавающей точкой - второй преобразуется во float и дробная часть не потеряется.

```
float c = 5.0/6;
```

Пример приоретизации операций:

```
2 / 4 * 3,14
// результат 0, так как одноуровневые операции слева направо
```

Явное преобразование:

```
double result = (double)a / b;
```