Лабораторная работа 2 (0010 = 2)

Представление данных в ЭВМ

Цель работы: сопоставить размеры стандартных типов C/C++ на различных платформах; изучить форматы представления чисел на примере выбранной платформы.

Все задания Л2 выполняются на чистом C/C++, без использования ассемблера.

Задание Л2.№1

Выполните измерения согласно заданию Л1.№2 на платформах, доступных на ВЦ (таблица Л2.1).

Размеры каких типов одинаковы для наиболее популярных платформ? Какие характеристики типов приведены в стандарте C, а какие — могут различаться?

Штраф −1 балл за платформу таблицы Л2.1, если в аудитории она доступна1 , а данных по ней нет.

Бонус +2 балла за платформу. При подготовке к работе выполните измерения на платформе, отсутствующей в таблице Л2.1. Укажите ОС, компилятор, режим (разрядность) сборки, архитектуру процессора, назначение платформы — без этих сведений баллы не начисляются.

1. Платформы и компиляторы, рассмотренные в анализе:

ОС	Архитектура	Компилятор	Разрядность сборки
Windows	x86_64 (64)	MinGW (GCC) 16.0.2	64
Linux	x86_64 (64)	GCC 14.2.0	64
Linux	x86_64 (64)	Clang/LLVM 20.1.0	64
Windows	x86_64 (64)	MS Visual C++ 1940	64
Windows	x86 (32)	MS Visual C++ 1940	32

2. Размеры типов данных (в байтах) на разных платформах

Тип	Win x64 MinGW	Linux x64 GCC	Linux x64 Clang	Win x64 MSVC	Win x86 MSVC 32
char	1	1	1	1	1
char*	8	8	8	8	4
bool	1	1	1	1	1
wchar_t	2	4	4	2	2
short	2	2	2	2	2
unsigned short	2	2	2	2	2
int	4	4	4	4	4
long	4	8	8	4	4
long long	8	8	8	8	8
float	4	4	4	4	4

Тип	Win x64 MinGW	Linux x64 GCC	Linux x64 Clang	Win x64 MSVC	Win x86 MSVC 32
double	8	8	8	8	8
double*	8	8	8	8	4
long double	16	16	16	8	8
size_t	8	8	8	8	4
ptrdiff_t	8	8	8	8	4
void*	8	8	8	8	4

3. Анализ и выводы

Размеры одинаковые для большинства популярных платформ:

- char, bool, short, unsigned short, int, long long, float, double — размеры одинаковы на всех рассмотренных платформах.
- Размеры указателей (char*, double*, void*), size_t, ptrdiff_t зависят от разрядности платформы (4 байта на 32-битных системах, 8 байт на 64-битных).
- wchar_t имеет размер 2 байта на Windows и 4 байта на Linux (GCC/Clang).
- long на Windows 64-бит — 4 байта, на Linux 64-бит — 8 байт.
- long double у MSVC — 8 байт, у GCC и Clang — 16 байт.

4. Характеристики типов в стандарте C и вариации

- **Стандарт C гарантирует:**
  - char занимает 1 байт.
  - short не меньше 16 бит.
  - int не меньше 16 бит.
  - long не меньше int.
  - long long не меньше long.
  - bool представлен в <stdbool.h>, занимает 1 байт.
  - Размеры указателей зависят от архитектуры.
  - Минимальные размеры и отношения между типами, но не фиксированные размеры.
- **Могут различаться:**
  - Размер wchar_t (зависит от реализации ОС и компилятора).
  - Размер long (32 или 64 бит в зависимости от платформы).
  - Размер long double (различается между компиляторами и архитектурами).
  - Размеры указателей зависят от разрядности архитектуры (32/64 бит).

5. Итог

- Наиболее популярные типы с фиксированным размером на всех платформах: char, bool, short, unsigned short, int, long long, float, double.
- Размеры указателей и связанных с ними типов (size_t, ptrdiff_t) зависят от архитектуры (32/64 бит).
- Размер long и wchar_t различается между Windows и Linux.
- long double особенно вариативен между компиляторами.

Задание Л2.№2

Разработайте функцию void $viewPointer$(void *p), которая принимает нетипизированный указатель p, преобразует его в типизированные:

a) char *p1 = reinterpret_cast(p);
b) unsigned short *p2 = reinterpret_cast(p);
c) double *p3 = reinterpret_cast(p); и печатает адреса p, $p1$, $p2$, $p3$.

Убедитесь, что p, $p1$, $p2$, $p3$ — один и тот же адрес, то есть что reinterpret_cast не меняет преобразуемого указателя и, следовательно, может быть использован для интерпретации одной и той же области памяти как значений различных типов.

Дополните $viewPointer$() печатью смежных с p адресов: $p1$ + 1, $p2$ + 1, $p3$ + 1.

Сопоставьте разницу между pi и pi + 1 в байтах для типизированного указателя T * pi с размером типа T.

Проверьте, позволяют ли текущие настройки компилятора рассчитать p+1. Если да — какова разница между p и p+1 в байтах?

Разработайте функцию void $printPointer$(void *p), которая преобразует p в типизированные $p1$, $p2$, $p3$ аналогично $viewPointer$() и печатает значения соответствующих типов по адресу p: *$p1$, *$p2$, *$p3$.

Дополните $printPointer$() печатью значений по смежным с p адресам: *($p1$ + 1), *($p2$ + 1), *($p3$ + 1).

Все целые числа выводите в шестнадцатеричном виде.

Проверьте работу функций $viewPointer$() и $printPointer$() на значениях 0x1122334455667788 ($long\ long$), "0123456789abcdef" ($char$[]).

```
Задание №2
========================================================================
=== Тест 1: long long 0x1122334455667788 ===
Адреса указателей:
p  = 0x7ffcb92b4b58
p1 = 0x7ffcb92b4b58
p2 = 0x7ffcb92b4b58
p3 = 0x7ffcb92b4b58

Адреса p_i + 1:
p1 + 1 = 0x7ffcb92b4b59 (разница 1 байт)
p2 + 1 = 0x7ffcb92b4b5a (разница 2 байт)
p3 + 1 = 0x7ffcb92b4b60 (разница 8 байт)

Размеры типов:
sizeof(char) = 1
sizeof(unsigned short) = 2
sizeof(double) = 8
Значения по адресам:
*p1 = '' (0x88)
*(p1+1) = 'w' (0x77)
*p2 = 0x7788
*(p2+1) = 0x5566
*p3 = 3.84141e-226
*(p3+1) = 0

=== Тест 2: строка "0123456789abcdef" ===
Адреса указателей:
p  = 0x7ffcb92b4b60
p1 = 0x7ffcb92b4b60
p2 = 0x7ffcb92b4b60
p3 = 0x7ffcb92b4b60

Адреса p_i + 1:
p1 + 1 = 0x7ffcb92b4b61 (разница 1 байт)
p2 + 1 = 0x7ffcb92b4b62 (разница 2 байт)
p3 + 1 = 0x7ffcb92b4b68 (разница 8 байт)

Размеры типов:
sizeof(char) = 1
sizeof(unsigned short) = 2
sizeof(double) = 8
Значения по адресам:
*p1 = '0' (0x30)
*(p1+1) = '1' (0x31)
*p2 = 0x3130
*(p2+1) = 0x3332
*p3 = 9.95833e-43
*(p3+1) = 1.81795e+185

========================================================================
```

Рис. 1: Информация о системе и размерах типов

Листинг:

Файл task2_2.h:

```c
#ifndef task2_2_H
#define task2_2_H

void viewPointer(void *p);
void printPointer(void *p);

void run_task2_2()
{
    printf("\nЗадание №2\n");
    printf("================================================================================\n");
    // Тест 1: long long 0x1122334455667788
    long long ll = 0x1122334455667788LL;
    printf("=== Тест 1: long long 0x1122334455667788 ===\n");
    viewPointer(&ll);
    printPointer(&ll);
    printf("\n");
    // Тест 2: строка "0123456789abcdef"
    char str[] = "0123456789abcdef";
    printf("=== Тест 2: строка \"0123456789abcdef\" ===\n");
    viewPointer(str);
    printPointer(str);
    printf("\n================================================================================\n");
}

#endif
```

Файл task2_2.c:

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

#include "task2_2.h"

void viewPointer(void *p) {
    char *p1 = (char *)p;
    unsigned short *p2 = (unsigned short *)p;
    double *p3 = (double *)p;

    printf("Адреса указателей:\n");
    printf("p  = %p\n", p);
    printf("p1 = %p\n", (void *)p1);
    printf("p2 = %p\n", (void *)p2);
    printf("p3 = %p\n", (void *)p3);

    printf("\nАдреса p_i + 1:\n");
    printf("p1 + 1 = %p (разница %ld байт)\n", (void *)(p1 + 1), (char *)(p1 + 1) - (char *)p1);
    printf("p2 + 1 = %p (разница %ld байт)\n", (void *)(p2 + 1), (char *)(p2 + 1) - (char *)p2);
    printf("p3 + 1 = %p (разница %ld байт)\n", (void *)(p3 + 1), (char *)(p3 + 1) - (char *)p3);

    printf("\nРазмеры типов:\n");
    printf("sizeof(char) = %zu\n", sizeof(char));
    printf("sizeof(unsigned short) = %zu\n", sizeof(unsigned short));
    printf("sizeof(double) = %zu\n", sizeof(double));
}

void printPointer(void *p) {
    char *p1 = (char *)p;
    unsigned short *p2 = (unsigned short *)p;
    double *p3 = (double *)p;

    printf("Значения по адресам:\n");

    // Для char — выводим как символы и шестнадцатеричные значения
    printf("*p1 = '%c' (0x%02X)\n", *p1, (unsigned char)*p1);
    printf("*(p1+1) = '%c' (0x%02X)\n", *(p1 + 1), (unsigned char)*(p1 + 1));

    // Для unsigned short — выводим в hex
    printf("*p2 = 0x%04X\n", *p2);
    printf("*(p2+1) = 0x%04X\n", *(p2 + 1));

    // Для double — выводим в формате %g (число с плавающей точкой)
    printf("*p3 = %g\n", *p3);
    printf("*(p3+1) = %g\n", *(p3 + 1));
}
```

Задание Л2.№3

Разработайте функцию void $printDump$(void *p,size_t N), которая преобразует нетипизированный указатель p в указатель $p1$ на байт ($char/unsigned\ char$) и печатает шестнадцатеричные значения N байтов, начиная с этого адреса: *$p1$, *($p1$ + 1), ... * ($p1$ + (N − 1)) — шестнадцатеричный дамп памяти.

Каждый байт должен выводиться в виде двух шестнадцатеричных цифр, байты разделяются пробелом (спецификатор «%02hhX »).

Исследуйте при помощи $printDump$(), как хранятся в памяти компьютера:

a) целое число x (типа int; таблица Л2.2); по результату исследования определите порядок следования байтов в словах для вашего процессора: — прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему); — обратный (младший байт по старшему адресу, порядок Motorola, BigEndian, от старшего к младшему);
b) число с плавающей запятой x (типа $double$; таблица Л2.2).
c) строки "abcdef" и "абвгде" (массив из $char$; при выборе N учитывайте всю длину строки, а не только видимые буквы);
d) «широкие» строки L"abcdef" и L"абвгде" (массив из $wchar\_t$; при выборе N учитывайте всю длину строки).

Длину строки в элементах $char/wchar\_t$ без завершающего нуля можно получить при помощи функций $strlen()/wcslen()$; после чего, зная размер элемента и то, что завершающий ноль занимает один элемент, можно вычислить размер строки в байтах. Если строки хранятся в статическом массиве без явного указания размера, инициализированном строкой при создании — размер строки равен размеру массива (можно определить sizeof).

Бонус +2 балла. Выполните в) и г) на всех платформах таблицы Л2.1.
Бонус +2 балла за платформу. Выполните измерения на платформе, где архитектура процессора отлична от x86/amd64.

1. Платформы и компиляторы, рассмотренные в анализе:

ОС	Архитектура	Компилятор	Разрядность сборки
Windows	x86_64 (64)	MinGW (GCC) 16.0.2	64
Linux	x86_64 (64)	GCC 14.2.0	64
Linux	x86_64 (64)	Clang/LLVM 20.1.0	64
Windows	x86_64 (64)	MS Visual C++ 1940	64
Windows	x86 (32)	MS Visual C++ 1940	32

2. Полученные результаты:

ОС / Компилятор	Тип данных	Представление в памяти (байты)	Комментарий
Windows / Clang/LLVM 16.0.2 x86_64 (64)	int x = 0x12345678	78 56 34 12	Little-Endian
	double d = 12345.7	A1 F8 31 E6 D6 1C C8 40	Little-Endian IEEE 754
	Строка "abcdef"	61 62 63 64 65 66 00	ASCII + завершающий нуль
	Строка "абвгде"	D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00	UTF-8
	Широкая строка L"abcdef"	61 00 62 00 63 00 64 00 65 00 66 00 00 00	wchar_t 2 байта, UTF-16LE
	Широкая строка L"абвгде"	30 04 31 04 32 04 33 04 34 04 35 04 00 00	wchar_t 2 байта, UTF-16LE

ОС / Компилятор	Тип данных	Представление в памяти (байты)	Комментарий
Linux / GCC 14.2.0 x86_64 (64)	int x = 0x12345678	78 56 34 12	Little-Endian
	double d = 12345.7	A1 F8 31 E6 D6 1C C8 40	Little-Endian IEEE 754
	Строка "abcdef"	61 62 63 64 65 66 00	ASCII + завершающий нуль
	Строка "абвгде"	D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00	UTF-8
	Широкая строка L"abcdef"	61 00 00 00 62 00 00 00 63 00 00 00 64 00 00 00 65 00 00 00 66 00 00 00 00 00 00 00	wchar_t 4 байта, UTF-32LE
	Широкая строка L"абвгде"	30 04 00 00 31 04 00 00 32 04 00 00 33 04 00 00 34 04 00 00 35 04 00 00 00 00 00 00	wchar_t 4 байта, UTF-32LE
Linux / Clang/LLVM 20.1.0 x86_64 (64)	int x = 0x12345678	78 56 34 12	Little-Endian
	double d = 12345.7	A1 F8 31 E6 D6 1C C8 40	Little-Endian IEEE 754
	Строка "abcdef"	61 62 63 64 65 66 00	ASCII + завершающий нуль
	Строка "абвгде"	D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00	UTF-8
	Широкая строка L"abcdef"	61 00 00 00 62 00 00 00 63 00 00 00 64 00 00 00 65 00 00 00 66 00 00 00 00 00 00 00	wchar_t 4 байта, UTF-32LE
	Широкая строка L"абвгде"	30 04 00 00 31 04 00 00 32 04 00 00 33 04 00 00 34 04 00 00 35 04 00 00 00 00 00 00	wchar_t 4 байта, UTF-32LE
Windows / MS Visual C++ 19.40 x86_64 (64)	int x = 0x12345678	78 56 34 12	Little-Endian
	double d = 12345.7	A1 F8 31 E6 D6 1C C8 40	Little-Endian IEEE 754
	Строка "abcdef"	61 62 63 64 65 66 00	ASCII + завершающий нуль
	Строка "абвгде"	D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00	UTF-8
	Широкая строка L"abcdef"	61 00 62 00 63 00 64 00 65 00 66 00 00 00	wchar_t 2 байта, UTF-16LE
	Широкая строка L"абвгде"	30 04 31 04 32 04 33 04 34 04 35 04 00 00	wchar_t 2 байта, UTF-16LE
Windows / MS Visual C++ 19.40 x86 (32)	int x = 0x12345678	78 56 34 12	Little-Endian
	double d = 12345.7	A1 F8 31 E6 D6 1C C8 40	Little-Endian IEEE 754
	Строка "abcdef"	61 62 63 64 65 66 00	ASCII + завершающий нуль
	Строка "абвгде"	D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00	UTF-8
	Широкая строка L"abcdef"	61 00 62 00 63 00 64 00 65 00 66 00 00 00	wchar_t 2 байта, UTF-16LE
	Широкая строка L"абвгде"	30 04 31 04 32 04 33 04 34 04 35 04 00 00	wchar_t 2 байта, UTF-16LE

3. Анализ представления данных в памяти по заданию

a) Целое число int x = 0x12345678

Во всех случаях байты хранятся в порядке: 78 56 34 12
Это соответствует **Little-Endian** (прямой порядок Intel), где младший байт лежит по младшему адресу.

---

b) Число с плавающей точкой double d = 12345.7

Во всех случаях байты хранятся одинаково: A1 F8 31 E6 D6 1C C8 40
Это также соответствует **Little-Endian** порядку хранения для 64-битных чисел с плавающей точкой.

---

c) Строки в памяти

- Строка "abcdef" (массив char): 61 62 63 64 65 66 00
- Строка "абвгде" (массив char в UTF-8): D0 B0 D0 B1 D0 B2 D0 B3 D0 B4 D0 B5 00

  Здесь видим, что кириллические символы кодируются в UTF-8 по 2 байта на символ, плюс завершающий нулевой байт.

---

d) Широкие строки L"abcdef" и L"абвгде" (массив wchar_t)

- Windows (MSVC и Clang/LLVM):
  61 00 62 00 63 00 64 00 65 00 66 00 00 00
  И
  30 04 31 04 32 04 33 04 34 04 35 04 00 00
  Здесь wchar_t — 2 байта (UTF-16LE), каждый символ занимает 2 байта, младший байт идет первым.
- Linux (GCC и Clang/LLVM):
  61 00 00 00 62 00 00 00 63 00 00 00 64 00 00 00 65 00 00 00 66 00 00 00 00 00 00 00
  и
  30 04 00 00 31 04 00 00 32 04 00 00 33 04 00 00 34 04 00 00 35 04 00 00 00 00 00 00
  Здесь wchar_t — 4 байта (UTF-32LE), каждый символ занимает 4 байта, младший байт идет первым.

4. Итог

Пункт	Вывод по исследованию
a)	Порядок байтов для int — **Little-Endian** (Intel, прямой порядок) на всех платформах
b)	Для double также Little-Endian, байты хранятся в порядке от младшего к старшему
c)	Строки char[] хранятся в UTF-8, символы ASCII — по 1 байту, кириллица — по 2 байта
d)	wchar_t на Windows — 2 байта (UTF-16LE), на Linux — 4 байта (UTF-32LE), порядок Little-Endian

Листинг:

```c
#include <stdio.h>
#include <string.h>
#include <wchar.h>
#include <locale.h>

void printDump(void *p, size_t N) {
    unsigned char *p1 = (unsigned char *)p;
    for (size_t i = 0; i < N; i++) {
        printf("%02hhX", p1[i]);
        if (i + 1 < N) printf(" ");
    }
    printf("\n");
}

int main() {
    // Установка локали для корректного вывода wchar_t
    setlocale(LC_ALL, "");

    // a) int x
    int x = 0x12345678;
    printf("int x = 0x12345678, хранение в памяти (байты):\n");
    printDump(&x, sizeof(x));
    // Если младший байт в начале (0x78), значит Little-Endian

    // б) double x
    double d = 12345.6789;
    printf("double d = %g, хранение в памяти (байты):\n", d);
    printDump(&d, sizeof(d));

    // в) строки "abcdef" и "абвгде"
    char str1[] = "abcdef";
    char str2[] = "абвгде";  // зависит от кодировки исходного файла и консоли
    printf("Строка \"abcdef\" (включая завершающий нуль):\n");
    printDump(str1, sizeof(str1));  // sizeof включает '\0'
    printf("Строка \"абвгде\" (включая завершающий нуль):\n");
    printDump(str2, sizeof(str2));

    // г) широкие строки L"abcdef" и L"абвгде"
    wchar_t wstr1[] = L"abcdef";
    wchar_t wstr2[] = L"абвгде";
    printf("Широкая строка L\"abcdef\" (включая завершающий нуль):\n");
    printDump(wstr1, sizeof(wstr1));
    printf("Широкая строка L\"абвгде\" (включая завершающий нуль):\n");
    printDump(wstr2, sizeof(wstr2));

    return 0;
}
```

Задание Л2.№4.

Напечатайте, используя (limits.h) C, минимальные и максимальные значения 8-битных целых типов $char$ и $unsigned\ char$, 16-битных $short$ и $unsigned\ short$, 32-битных int и $unsigned$, 64-битных $long\ long$ и $unsigned\ long\ long$.

1. **Сколько различных значений может принимать переменная беззнакового N- битного типа? Знакового N-битного? Связано ли это как-то со значением N и как именно? Л2. Представление данных в ЭВМ 347**
   - Беззнаковый N-битный тип может принимать 2^N различных значений, от 0 до 2^N-1.
   - Знаковый N-битный тип может принимать 2^N различных значений, от -2^{N-1} до $2^{N-1}-1$.
2. **Каждое ли целое число $x \in [0, 2^N)$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [0, 2^N)$? Взаимно однозначно ли это соответствие?** – Да, да, соответствие взаимно однозначно.
3. **Каждое ли целое число $x \in [-2^{N-1}, 2^{N-1})$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [-2^{N-1}, 2^{N-1})$? Взаимно однозначно ли это соответствие?** - Да, да.

Напечатайте минимальные и максимальные значения min и max 32-битного типа с плавающей запятой $float$, 64-битного $double$.

1. **Каждое ли вещественное число $x \in$ [min, max] имеет своё представление с плавающей запятой стандарта IEEE 754 ($float/double$ ЭВМ)?** Нет. Между числами с плавающей точкой существуют промежутки — не все вещественные числа в этом диапазоне могут быть представлены точно.
2. **Всякая ли последовательность ξ из N битов ($N \in \{32, 64\}$) может быть рассмотрена как N-битное значение с плавающей запятой? Всегда ли это значение — число?** Да, любая битовая последовательность интерпретируется как некоторое число с плавающей точкой, но не всегда это число — могут быть специальные значения (NaN, ±∞).
3. **Каких чисел больше: 32-битных целых ($int/unsigned$) или 32-битных с плавающей запятой ($float$)? 64-битных целых ($long\ long/unsigned\ long\ long$) или 64-битных с плавающей запятой ($double$)? Значения ±∞ числами не являются, нечисла nan — тем более**.
   - Целые числа — это просто все возможные комбинации битов, которые однозначно соответствуют числам, поэтому целых чисел всегда больше, чем чисел с плавающей запятой того же битового размера. Это связано с тем, что числа с плавающей запятой используют часть битов для экспоненты и имеют особые зарезервированные значения (±∞, NaN), а также не представляют все возможные значения в диапазоне, а только дискретные точки.

```
Задание №4
========================================================================
Целочисленные типы:
char:                   min = -128, max = 127
unsigned char:          min = 0, max = 255
short:                  min = -32768, max = 32767
unsigned short:         min = 0, max = 65535
int:                    min = -2147483648, max = 2147483647
unsigned int:           min = 0, max = 4294967295
long long:              min = -9223372036854775808, max = 9223372036854775807
unsigned long long:     min = 0, max = 18446744073709551615

Плавающие типы:
float:                  min = 1.175494e-38, max = 3.402823e+38
double:                 min = 2.225074e-308, max = 1.797693e+308

========================================================================
```

Рис. 2: Ввод диапазона допустимых значения для каждого стандартного типа данных

Листинг:

Файл task2_4.c:

```c
#ifndef task2_4_H
#define task2_4_H

#include <stdio.h>
#include <limits.h>
#include <float.h>

void run_task2_4()
{
    printf("\nЗадание №4\n");
    printf("========================================================================\n");

    printf("Целочисленные типы:\n");

    printf("char:                   min = %d, max = %d\n", CHAR_MIN, CHAR_MAX);
    printf("unsigned char:          min = 0, max = %u\n", UCHAR_MAX);

    printf("short:                  min = %d, max = %d\n", SHRT_MIN, SHRT_MAX);
    printf("unsigned short:         min = 0, max = %u\n", USHRT_MAX);

    printf("int:                    min = %d, max = %d\n", INT_MIN, INT_MAX);
    printf("unsigned int:           min = 0, max = %u\n", UINT_MAX);

    printf("long long:              min = %lld, max = %lld\n", LLONG_MIN, LLONG_MAX);
    printf("unsigned long long:     min = 0, max = %llu\n", ULLONG_MAX);

    printf("\nПлавающие типы:\n");
    printf("float:                  min = %e, max = %e\n", FLT_MIN, FLT_MAX);
    printf("double:                 min = %e, max = %e\n", DBL_MIN, DBL_MAX);

    printf("\n========================================================================\n");
}

#endif
```

Задание Л2.№5.

Исследуйте внутреннее представление 16-битных чисел.

Для этого на C/C++ разработайте void $print16$(void * p) (одну: С или П).

С-Л2.№5 для вывода при помощи стандартной библиотеки C (рекомендуется): функция $print16$() интерпретирует p как адрес 16-битного целого числа x типа $short/unsigned\ short$ и печатает x при помощи $printf$():

С-а) в шестнадцатеричном представлении;

С-б) в двоичном представлении: если не поддерживается формат b — напечатать биты (x & (1 << i)) != 0, от старшего i = 15 и до i = 0;

С-в) в десятичном беззнаковом представлении;

С-г) в десятичном знаковом представлении.

Необходимый вид вывода $print16$() для значений 13 и 0x8000 по адресу p:

000D 0000000000001101 13 +13

8000 1000000000000000 32768 -32768

Для $print16$(), а также последующих $print32$(), $print64$(), и любых x и y младшая цифра любого представления y всегда должна печататься под младшей цифрой соответствующего представления x.

П-Л2.№5 для вывода в потоки (крайне не рекомендуется, но и не штрафуется).

Так как вывод в поток различает $short$ и $unsigned\ short$, функция $print16$() интерпретирует адрес p дважды:

— как адрес 16-битного беззнакового целого ux типа $unsigned\ short$;

— и как адрес 16-битного знакового целого sx типа $short$;

и печатает оба ux и sx во всех доступных представлениях:

П-а) ux в шестнадцатеричном представлении;

П-б) ux в двоичном представлении (шаблон std::bitset);

П-в) ux в десятичном представлении;

П-г) sx в шестнадцатеричном представлении;

П-д) sx в двоичном представлении (шаблон std::bitset);

П-е) sx в десятичном представлении.

Не забывайте, что манипуляторы dec/hex действуют на все числа до отмены, а $setw$(int w) — только на одно следующее.

Убедитесь в процессе исследования, что (П-б) и (П-д) — одно и то же двоичное представление; (П-а) и (П-г) — одно и то же шестнадцатеричное представление. Если у вас (П-б)≠(П-д) или (П-а)≠(П-г) — ищите ошибку.

Сократите вывод П-Л2.№5 до вида, аналогичного С-Л2.№5.

Штраф –1 балл, если версия задания С-Л2.№5 реализована при помощи потоков или П-Л2.№5 при помощи стандартной библиотеки С.

Штраф –1 балл, если вывод $print$16() занимает более одной строки. Исследуйте при помощи $print$16() 16-битные целочисленные переменные типа $short/unsigned\ short$, принимающие значения:

— минимальное и максимальное целое беззнаковое 16-битные значения;

— минимальное и максимальное целое знаковое 16-битные значения;

— целочисленные x, y, a, b, соответствующие варианту (таблица Л2.2).

Как представляются в памяти ЭВМ знаковые целые значения?

Обычно в памяти знаковые целые числа хранятся в формате дополнительного кодая

Для 16-битного числа это значит:

- Старший бит (бит 15) — знак: 0 — положительное, 1 — отрицательное.
- Отрицательные числа хранятся как дополнительный код от соответствующего положительного числа.

Как различаются целочисленные x и $y = -x$? Как связаны двоичное представление (С-б) и шестнадцатеричное (С-а)?

- Если x — положительное число, то -x — это инверсия битов x с добавлением 1
- Например, для x = 9 (0000 0000 0000 1001), -x = -9 (1111 1111 1111 0111 + 1 = 1111 1111 1111 1000).
- Каждые 4 бита двоичного числа соответствуют одной шестнадцатеричной цифре.
- Например, двоичное 0000 0000 0000 1101 соответствует шестнадцатеричному 000D.

Как по шестнадцатеричному (С-а) записать двоичное для любого выбранного числа?

- Каждую шестнадцатеричную цифру заменяем на 4-битный двоичный эквивалент.
- Например, 8 в hex — 1000 в бинарном, 0 — 0000, D — 1101.

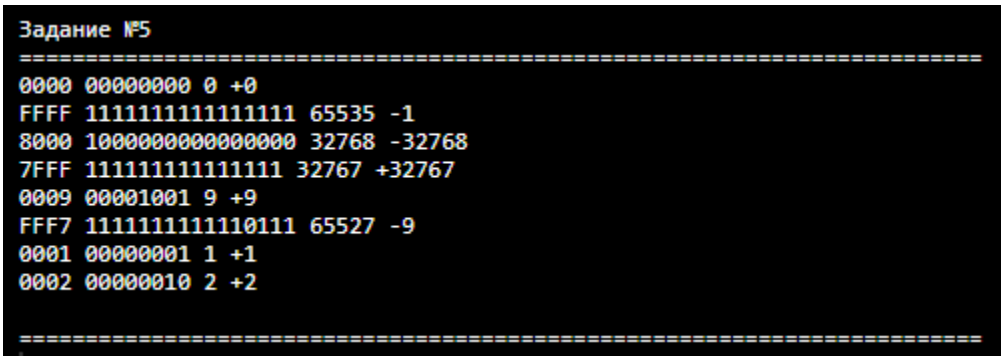

Рис. 3: Вывод функции $print$16()

Листинг:

Файл task2_5.c:

```c
#ifndef task2_5_H
#define task2_5_H

#include <stdio.h>

#define toUshort *(unsigned short*)
#define toshort *(short*)

void print16(void *p);

void run_task2_5()
{
    printf("\nЗадание №5\n");
    printf("================================================================================\n");

    unsigned short umin = 0x0000;          // минимальное беззнаковое
    unsigned short umax = 0xFFFF;          // максимальное беззнаковое
    short smin = (short)0x8000;            // минимальное знаковое (-32768)
    short smax = 0x7FFF;                   // максимальное знаковое (32767)

    print16(&umin);
    print16(&umax);
    print16(&smin);
    print16(&smax);

    // Вариант 17: (17-1)%2 + 1 = 1
    short x = 9;
    short y = -9;
    short a = 1;
    short b = 2;

    print16(&x);
    print16(&y);
    print16(&a);
    print16(&b);


    printf("\n================================================================================\n");
}

void print16(void *p) {
    // Интерпретируем указатель как unsigned short и short
    // Вывод шестнадцатеричного представления беззнакового числа (4 символа с ведущими нулями) %04X
    // Вывод двоичного представления (от старшего бита к младшему) %08b
    // Вывод десятичного беззнакового числа %u
    // Вывод десятичного знакового числа с явным знаком %+d
    printf("%04X %08b %u %+d \n", toUshort p,  toUshort p, toUshort p,  toshort p);
}

#endif
```

Задание Л2.№6.

Исследуйте внутреннее представление 32-битных чисел — целых $int/unsigned$ и с плавающей запятой $float$.

Для этого на C/C++ разработайте void $print32$(void * p) (одну: С или П).
С-Л2.№6 для стандартной библиотеки С: $print32$() интерпретирует p дважды:
— как адрес 32-битного целого числа x типа $int/unsigned$;
— как адрес 32-битного числа с плавающей запятой fx типа $float$;
и печатает x в представлениях (С-а)–(С-г), а fx:
С-д) в шестнадцатеричном экспоненциальном представлении;
С-е) в десятичном экспоненциальном представлении;
С-ж) в представлении с десятичной запятой.

Необходимый вид вывода $print32$() для значений 13 и 0x80000000 по адресу p:
0000000D 00000000000000000000000000001101 13 +13 +0X1.A0P-146 +1.82e-44 +0.00 80000000
10000000000000000000000000000000 2147483648 -2147483648 -0X0.00P+0 -0.00e+00 -0.00

Если ширина терминала позволяет вывести (С-а)–(С-ж) в одну строку — это необходимо сделать; иначе вторая строка должна иметь отступ.

Исследуйте при помощи $print32$() 32-битные переменные — целочисленные типа $int/unsigned$ и с плавающей запятой типа $float$:
— минимальное и максимальное целое беззнаковое 32-битные значения;
— минимальное и максимальное целое знаковое 32-битные значения;
— целочисленные x, y, a, b, c, d, соответствующие варианту (таблица Л2.2);
— $float$-значения x, y, a, b, c, d, соответствующие варианту (таблица Л2.2);

```
Задание №6
================================================================================
00000000 00000000 0 +0 +0x0p+0 +0.000000e+00 +0.00
FFFFFFFF 11111111111111111111111111111111 4294967295 -1 -nan -nan -nan
80000000 10000000000000000000000000000000 2147483648 -2147483648 -0x0p+0 -0.000000e+00 -0.00
7FFFFFFF 11111111111111111111111111111111 2147483647 +2147483647 +nan +nan +nan
00000009 00001001 9 +9 +0x1.2p-146 +1.261169e-44 +0.00
FFFFFFF7 11111111111111111111111111110111 4294967287 -9 -nan -nan -nan
00000001 00000001 1 +1 +0x1p-149 +1.401298e-45 +0.00
00000002 00000010 2 +2 +0x1p-148 +2.802597e-45 +0.00
00BC614E 10111100011000010100111O 12345678 +12345678 +0x1.78c29cp-126 +1.729998e-38 +0.00
075BCD15 11101011011110011010001O101 123456789 +123456789 +0x1.b79a2ap-113 +1.653600e-34 +0.00
41100000 10000010001000000000000000000000 1091567616 +1091567616 +0x1.2p+3 +9.000000e+00 +9.00
C1100000 11000001000100000000000000000000 3239051264 -1055916032 -0x1.2p+3 -9.000000e+00 -9.00
3F800000 11111110000000000000000000000000 1065353216 +1065353216 +0x1p+0 +1.000000e+00 +1.00
40000000 10000000000000000000000000000000 1073741824 +1073741824 +0x1p+1 +2.000000e+00 +2.00
4B3C614E 10010110011110001100001010011110 1262248270 +1262248270 +0x1.78c29cp+23 +1.234568e+07 +12345678.00
4CEB79A3 10011001110101101111001101000011 1290500515 +1290500515 +0x1.d6f346p+26 +1.234568e+08 +123456792.00

================================================================================
```

Рис. 4: Вывод функции $print32$()

Листинг:

Файл task2_6.c:

```c
#ifndef task2_6_H
#define task2_6_H

#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>

#define toUInt *(uint32_t*)
#define toInt *(int32_t*)
#define toFloat *(float*)

void print32(void *p);

void run_task2_6()
{
    printf("\nЗадание №6\n");
    printf("================================================================================\n");

    uint32_t umin = 0x00000000;          // минимальное беззнаковое
    uint32_t umax = 0xFFFFFFFF;          // максимальное беззнаковое
    int32_t smin = 0x80000000;           // минимальное знаковое (-2147483648)
    int32_t smax = 0x7FFFFFFF;           // максимальное знаковое (2147483647)

    print32(&umin);
    print32(&umax);
    print32(&smin);
    print32(&smax);

    // Вариант 17: (17-1)%2 + 1 = 1
    int32_t x = 9;
    int32_t y = -9;
    int32_t a = 1;
    int32_t b = 2;
    int32_t c = 12345678;
    int32_t d = 123456789;

    float fx = 9.0f;
    float fy = -9.0f;
    float fa = 1.0f;
    float fb = 2.0f;
    float fc = 12345678.0f;
    float fd = 123456789.0f;

    print32(&x);
    print32(&y);
    print32(&a);
    print32(&b);
    print32(&c);
    print32(&d);

    print32(&fx);
    print32(&fy);
    print32(&fa);
    print32(&fb);
    print32(&fc);
    print32(&fd);

    printf("\n================================================================================\n");
}

void print32(void *p) {
    printf("%08X %08b %u %+d %+a %+e %+.2f \n", toFloat p, toFloat p, toFloat p, toInt p, toUInt p,
toUInt p, toUInt p);
}

#endif
```

Задание Л2.№7.

Аналогично Л2.№6, разработайте void $print64$(void * p), аналогичную $print32$() для размера 64 бита. Необходимый вид вывода $print64$() для значений 13 и 0x8000000000000000: 000000000000000D 0000000000000000000000000000000000000000000000000000000000001101 13 +13 +0X0.00P-1022 +6.42e-323 +0.00 8000000000000000 1000000000000000000000000000000000000000000000000000000000000000 9223372036854775808 -9223372036854775808 -0X0.00P+0 -0.00e+00 -0.00 Если ширина терминала позволяет вывести данные в одну или хотя бы в две строки — это необходимо сделать; иначе вторая и третья должна иметь отступ Исследуйте $print64$() переменные $long\ long/unsigned\ long\ long$ и $double$: — целочисленные x, a, соответствующие варианту (таблица Л2.2); — $double$-значения x, a, соответствующие варианту (таблица Л2.2). Чем различается одно и то же значение (x или a), записанное в $float$ и $double$? В каком интервале $double$-чисел больше: [+0, +2) или [+2, +∞)? Штраф −4 балла, если для любой $printWW$(), где $WW \in \{16, 32, 64\}$: — порядок представлений отличается от указанного; — представления не соответствует друг другу (а команда, вместо того, чтобы задать вопрос преподавателю — так и сдаёт); — в частности, двоичное (С-б) явно несообразно шестнадцатеричному (С-а); или формат вывода отличается от указанного в худшую сторону, в частности: — количество выводимых цифр двоичного представления (С-б) отлично от количества бит WW в числе; — а выводимых цифр шестнадцатеричного (С-а) — от количества тетрад $WW/4$; — младшая цифра не под младшей предыдущего вызова $printWW$(); — вывод менее компактен, чем указано (делать компактнее при сохранении информативности — можно и нужно).

```
Задание №6
========================================================================
Целочисленные значения:
x_ll = 9:
0000000000000009 1001 9 +9 +0x0.0000000000009p-1022 +4.446591e-323 +0.00

y_ll = -9:
FFFFFFFFFFFFFFF7 11111111111111111111111111110111 18446744073709551607 -9 -nan -nan -nan

a_ull = 1:
0000000000000001 1 1 +1 +0x0.0000000000001p-1022 +4.940656e-324 +0.00

b_ull = 2:
0000000000000002 10 2 +2 +0x0.0000000000002p-1022 +9.881313e-324 +0.00

c_ull = 12345678:
0000000000BC614E 101111000110000101001110 12345678 +12345678 +0x0.0000000bc614ep-1022 +6.099575e-317 +0.00

d_ull = 123456789:
00000000075BCD15 111010110111100110100010101 123456789 +123456789 +0x0.00000075bcd15p-1022 +6.099576e-316 +0.00

Double значения:
x_d = 9.0:
4022000000000000 0 4621256167635550208 +4621256167635550208 +0x1.2p+3 +9.000000e+00 +9.00

a_d = 1.0:
3FF0000000000000 0 4607182418800017408 +4607182418800017408 +0x1p+0 +1.000000e+00 +1.00

Сравнение float и double для 9.0:
float:  41100000 (hex), 9.0000000 (decimal)
double: 4022000000000000 (hex), 9.000000000000000 (decimal)


========================================================================
```

Рис. 5: Вывод функции $print64$()

Листинг:

Файл task2_7.c:

```c
#ifndef task2_7_H
#define task2_7_H

#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>

#define toUInt64(p) (*(uint64_t*)(p))
#define toInt64(p)  (*(int64_t*)(p))
#define toDouble(p) (*(double*)(p))
```

```
11.
12. void print64(void *p);
13.
14. void run_task2_7()
15. {
16.     printf("\nЗадание №6\n");
17.     printf("=============================================================================\n");
18.
19.     // 1. Целочисленные значения long long и unsigned long long
20.     long long x_ll = 9LL;
21.     long long y_ll = -9LL;
22.     unsigned long long a_ull = 1ULL;
23.     unsigned long long b_ull = 2ULL;
24.     unsigned long long c_ull = 12345678ULL;
25.     unsigned long long d_ull = 123456789ULL;
26.
27.     printf("Целочисленные значения:\n");
28.
29.     printf("x_ll = 9:\n");
30.     print64(&x_ll);
31.     printf("\n");
32.
33.     printf("y_ll = -9:\n");
34.     print64(&y_ll);
35.     printf("\n");
36.
37.     printf("a_ull = 1:\n");
38.     print64(&a_ull);
39.     printf("\n");
40.
41.     printf("b_ull = 2:\n");
42.     print64(&b_ull);
43.     printf("\n");
44.
45.     printf("c_ull = 12345678:\n");
46.     print64(&c_ull);
47.     printf("\n");
48.
49.     printf("d_ull = 123456789:\n");
50.     print64(&d_ull);
51.     printf("\n");
52.
53.     // 2. Double значения x и a
54.     double x_d = 9.0;
55.     double a_d = 1.0;
56.
57.     printf("Double значения:\n");
58.
59.     printf("x_d = 9.0:\n");
60.     print64(&x_d);
61.     printf("\n");
62.
63.     printf("a_d = 1.0:\n");
64.     print64(&a_d);
65.     printf("\n");
66.
67.     // 3. Различия одного и того же значения в float и double
68.     float f_val = 9.0f;
69.     double d_val = 9.0;
70.
71.     printf("Сравнение float и double для 9.0:\n");
72.     printf("float:  %08X (hex), %.7f (decimal)\n", *(uint32_t*)&f_val, f_val);
73.     printf("double: %016llX (hex), %.15f (decimal)\n", *(unsigned long long*)&d_val, d_val);
74.     printf("\n");
75.
76.     printf("\n=============================================================================\n");
77. }
78.
79. void print64(void *p) {
80.     printf("%016llX %b %llu %+lld %+a %+e %+.2f\n", toUInt64(p), toUInt64(p), toUInt64(p),
toInt64(p), toDouble(p), toDouble(p), toDouble(p));
81. }
82. #endif
```

Задание Л2.№8.

Разработайте функцию $c16to32$(void *p), которая принимает адрес 16-битной целочисленной переменной, печатает её значение $print16$(), расширяет это значение до 32 бит двумя способами:

— как знаковое ($short \rightarrow int$);

— как беззнаковое ($unsigned\ short \rightarrow unsigned\ int$).

и для каждого способа печатает результат $print32$().

Явное расширение в C++ выполняется static_cast; неявное — в частности, при присваивании. Если источник и приёмник различаются не только размером, но и знаковостью — неопределённое поведение C/C++. Проверьте её работу на значениях m и n (таблица Л2.3).

```
Задание №8
========================================================================
Для m = 21:
Исходное 16-битное значение:
0015 00010101 21 +21
Знаковое расширение до 32 бит:
00000015 00010101 21 +21 +0x1.5p-145 +2.942727e-44 +0.00
Беззнаковое расширение до 32 бит:
00000015 00010101 21 +21 +0x1.5p-145 +2.942727e-44 +0.00

Для n = -37:
Исходное 16-битное значение:
FFDB 1111111111011011 65499 -37
Знаковое расширение до 32 бит:
FFFFFFDB 11111111111111111111111111011011 4294967259 -37 -nan -nan -nan
Беззнаковое расширение до 32 бит:
0000FFDB 1111111111011011 65499 +65499 +0x1.ffb6p-134 +9.178365e-41 +0.00

========================================================================
```

Рис. 6: Вывод функции c16to32()

Листинг:

Файл task2_8.c:

```c
#ifndef task2_8_H
#define task2_8_H

#include <stdio.h>
#include <stdint.h>

void print16(void *p);
void print32(void *p);
void print64(void *p);

void c16to32(void *p);

void run_task2_8()
{
    printf("\nЗадание №8\n");
    printf("================================================================================\n");

    short m = 21;
    short n = -37;

    printf("Для m = 21:\n");
    c16to32(&m);
    printf("\n");

    printf("Для n = -37:\n");
    c16to32(&n);
    printf("\n");

    printf("\n================================================================================\n");
}

// Функция c16to32
void c16to32(void *p) {
    short val16 = *(short*)p;

    printf("Исходное 16-битное значение:\n");
    print16(&val16);

    // Расширение знаковое
    int val32_signed = (int)val16;

    printf("Знаковое расширение до 32 бит:\n");
    print32(&val32_signed);

    // Расширение беззнаковое
    unsigned int val32_unsigned = (unsigned int)(unsigned short)val16;

    printf("Беззнаковое расширение до 32 бит:\n");
    print32(&val32_unsigned);
}

#endif
```

Задание Л2.№9.

Разработайте функцию $ab16$(void * p), которая принимает адрес 16-битной целочисленной переменной x и выполняет над её копиями операции (а1)–(б6). Оригинал, лежащий по адресу p, должен оставаться неизменным; для каждой операции исходным является значение x, а не результат предыдущей. Исходное значение x и каждый результат печатается $print16$(). Сопоставьте результаты (аi) и (бi) — вначале на значении $x = m$, затем $x = n$ (таблица Л2.3).

а1) беззнаковое умножение на 2;

а2) знаковое умножение на 2;

а3) беззнаковое деление на 2;

а4) знаковое деление на 2;

а5) расчёт остатка от беззнакового деления на 16;

а6) округление вниз до числа, кратного 16 (беззнаковое);

б1) беззнаковый сдвиг влево на 1 бит;

б2) знаковый сдвиг влево на 1 бит;

б3) беззнаковый сдвиг вправо на 1 бит; б

б4) знаковый сдвиг вправо на 1 бит;

б5) расчёт x & 15; б6) расчёт x & −16.

Если Л2.№9 реализуется на чистом C/C++, то беззнаковые и знаковые операции (умножение/деление/сдвиги) записываются одним и тем же оператором.

Таким образом, адрес p необходимо интерпретировать дважды:

— как адрес 16-битного беззнакового целого ux типа $unsigned\ short$;

— и как адрес 16-битного знакового целого sx типа $short$.

Одни и те же знаки операций C/C++:

— для ux обозначают беззнаковые операции (если операция бинарная, типа * или /, то и второй операнд должен быть того же типа $unsigned\ short$);

— для sx — знаковые (бинарные — для sx и другого $short$).

Также можно забежать вперёд и реализовать операции как ассемблерные вставки — на уровне ассемблера беззнаковые и знаковые операции выполняются разными командами: $mul/imul$, shl/sal, $div/idiv$, shr/sar.

```
Задание №9
========================================================================
Результаты для m = 21:
Исходное значение:
0015 00010101 21 +21
а1) беззнаковое умножение на 2: 002A 00101010 42 +42
а2) знаковое умножение на 2: 002A 00101010 42 +42
а3) беззнаковое деление на 2: 000A 00001010 10 +10
а4) знаковое деление на 2: 000A 00001010 10 +10
а5) остаток от беззнакового деления на 16: 0005 00000101 5 +5
а6) округление вниз до числа, кратного 16: 0010 00010000 16 +16
б1) беззнаковый сдвиг влево на 1 бит: 002A 00101010 42 +42
б2) знаковый сдвиг влево на 1 бит: 002A 00101010 42 +42
б3) беззнаковый сдвиг вправо на 1 бит: 000A 00001010 10 +10
б4) знаковый сдвиг вправо на 1 бит: 000A 00001010 10 +10
б5) расчёт x & 15: 0005 00000101 5 +5
б6) расчёт x & -16: 0010 00010000 16 +16

Результаты для n = -37:
Исходное значение:
FFDB 1111111111011011 65499 -37
а1) беззнаковое умножение на 2: FFB6 1111111110110110 65462 -74
а2) знаковое умножение на 2: FFB6 1111111110110110 65462 -74
а3) беззнаковое деление на 2: 7FED 111111111101101 32749 +32749
а4) знаковое деление на 2: FFEE 1111111111101110 65518 -18
а5) остаток от беззнакового деления на 16: 000B 00001011 11 +11
а6) округление вниз до числа, кратного 16: FFD0 1111111111010000 65488 -48
б1) беззнаковый сдвиг влево на 1 бит: FFB6 1111111110110110 65462 -74
б2) знаковый сдвиг влево на 1 бит: FFB6 1111111110110110 65462 -74
б3) беззнаковый сдвиг вправо на 1 бит: 7FED 111111111101101 32749 +32749
б4) знаковый сдвиг вправо на 1 бит: FFED 1111111111101101 65517 -19
б5) расчёт x & 15: 000B 00001011 11 +11
б6) расчёт x & -16: FFD0 1111111111010000 65488 -48

========================================================================
```

Рис. 7: Вывод функции ab16 ()

Листинг:

Файл task2_9.c:

```c
#ifndef task2_9_H
#define task2_9_H

#include <stdio.h>
#include <stdint.h>

void print16(void *p);

void ab16(void *p);

void run_task2_9()
{
    printf("\nЗадание №9\n");
    printf("================================================================================\n");

    short m = 21;
    short n = -37;

    printf("Результаты для m = 21:\n");
    ab16(&m);
    printf("\n");

    printf("Результаты для n = -37:\n");
    ab16(&n);

    printf("\n================================================================================\n");
}

void ab16(void *p) {
    short sx = *(short*)p;                      // знаковое 16-битное
    unsigned short ux = *(unsigned short*)p;    // беззнаковое 16-битное

    printf("Исходное значение:\n");
    print16(p);

    printf("а1) беззнаковое умножение на 2: ");
    print16(&(unsigned short){ux * 2});

    printf("а2) знаковое умножение на 2: ");
    print16(&(short){sx * 2});

    printf("а3) беззнаковое деление на 2: ");
    print16(&(unsigned short){ux / 2});

    printf("а4) знаковое деление на 2: ");
    print16(&(short){sx / 2});

    printf("а5) остаток от беззнакового деления на 16: ");
    print16(&(unsigned short){ux % 16});

    printf("а6) округление вниз до числа, кратного 16: ");
    print16(&(unsigned short){ux & ~0xF});

    printf("б1) беззнаковый сдвиг влево на 1 бит: ");
    print16(&(unsigned short){ux << 1});

    printf("б2) знаковый сдвиг влево на 1 бит: ");
    print16(&(short){sx << 1});

    printf("б3) беззнаковый сдвиг вправо на 1 бит: ");
    print16(&(unsigned short){ux >> 1});

    printf("б4) знаковый сдвиг вправо на 1 бит: ");
    print16(&(short){sx >> 1});

    printf("б5) расчёт x & 15: ");
    print16(&(short){sx & 15});

    printf("б6) расчёт x & -16: ");
    print16(&(short){sx & (-16)});
}

#endif
```