

# Отчёт по лабораторной работе №2 Представление данных в ЭВМ

Группа 8 Лазу Игорь, Меркулов Сергей

ОС: MSWindows

Компилятор: gccversion13.2.0

Разрядность сборки: 64 бит

Архитектура процессора: 64 бит

Назначение платформы: общее

**Задание Л2.з1.** Разработайте функцию `void viewPointer(void * p)`, которая принимает нетипизированный указатель  $p$ , преобразует его в типизированные: а) `char *p1 = reinterpret_cast(p)`; б) `unsigned short *p2 = reinterpret_cast(p)`; в) `double *p3 = reinterpret_cast(p)`; и печатает  $p$ ,  $p1$ ,  $p2$ ,  $p3$  (не значения по этим адресам, а сами адреса). Убедитесь, что  $p$ ,  $p1$ ,  $p2$ ,  $p3$  — один и тот же адрес, то есть что оператор `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:  $p + 1$ ,  $p2 + 1$ ,  $p3 + 1$ . 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?

Код:

```
void viewPointer(void * p){
    char *p1 = reinterpret_cast<char *>(p);
    unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
    double *p3 = reinterpret_cast<double *>(p);
    printf("p = %p\n", p);
    printf("p1 = %p\n", p1);
    printf("p2 = %p\n", p2);
    printf("p3 = %p\n", p3);
    printf("All adresses are equal\n\n");

    printf("p1 = %p, p1+1 = %p, size of type: %u\n", p1, p1 + 1, sizeof(*p1));
    printf("p2 = %p, p2+1 = %p, size of type: %u\n", p2, p2 + 1, sizeof(*p2));
    printf("p3 = %p, p3+1 = %p, size of type: %u\n", p3, p3 + 1, sizeof(*p3));
}
```

## Результат:

```
p = 0061FF18
p1 = 0061FF18
p2 = 0061FF18
p3 = 0061FF18
All addresses are equal
```

```
p1 = 0061FF18, p1+1 = 0061FF19, size of type: 1
p2 = 0061FF18, p2+1 = 0061FF1A, size of type: 2
p3 = 0061FF18, p3+1 = 0061FF20, size of type: 8
```

## Какова разница между $p$ и $p + 1$ в байтах?

- Разница между  $p$  и  $p+1$  в байтах соответствует размеру типа данных самой переменной.

**Задание Л2.з2.** Разработайте функцию `void printPointer(void *p)`, которая принимает нетипизированный указатель  $p$ , преобразует его 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 функции `printPointer()` на значениях `0x1122334455667788` (`longlong`), `"0123456789abcdef"` (`char[]`)

## Код

```
void printPointer(void *p){
    char *p1 = reinterpret_cast<char *>(p);
    unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
    double *p3 = reinterpret_cast<double *>(p);
    printf("*p1 = %c, *(p1 + 1) = %c\n", *p1, *(p1 + 1));
    printf("*p2 = %x, *(p2 + 1) = %x\n", *p2, *(p2 + 1));
    printf("*p3 = %f, *(p3 + 1) = %f\n", *p3, *(p3 + 1));
}
```

## Результат

```
*p1 = 0, *(p1 + 1) = 0
*p2 = 7788, *(p2 + 1) = 5566
*p3 = 0.000000, *(p3 + 1) = 0.000000
*p1 = 0, *(p1 + 1) = 1
*p2 = 3130, *(p2 + 1) = 3332
*p3 = 0.000000, *(p3 + 1) = 0.000000
```

Можно ли рассчитать (и, соответственно, напечатать)  $*p$ ?

Да, ведь нам известен адрес переменной

**Задание Л2.33.** Разработайте функцию `void printDump(void * p, size_t N)`, которая принимает нетипизированный указатель `p`, преобразует его в типизированный указатель на байт `unsigned char * p1` 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

С помощью `printDump()` определите и выпишите в отчёт, как хранятся в памяти компьютера в программе на C/C++:

– целое число `x` (типа `int`; таблица Л2.1); по результату исследования определите порядок следования байтов в словах для вашего процессора:

а) прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);

б) обратный (младший байт по старшему адресу, порядок Motorola, BigEndian, от старшего к младшему);

– массив из трёх целых чисел (статический или динамический, но не высокоуровневый контейнер) с элементами `x, y, z`;

– число с плавающей запятой `y` (типа `double`; таблица Л2.1).

## Вариант 2

```
void printDump(void *p, size_t N){
    unsigned char *p1 = reinterpret_cast<unsigned char *>(p);
    for(int byte = 0; byte < N; byte++){
        printf("%02hhX ", *(p1 + byte));
    }
}
```

## Порядок прямой

**Задание Л2.34.** Изучите, как интерпретируется одна и та же область памяти, если она рассматривается как знаковое или беззнаковое целое число, а также — как одно и то же число записывается в различных системах счисления. Для этого на языке C/C++ разработайте функцию `void print16(void * p)`, которая печатает для области памяти по заданному адресу `p`:

а) целочисленную беззнаковую 16-битную интерпретацию (`unsigned short`) в шестнадцатеричном представлении;

б) целочисленную беззнаковую 16-битную интерпретацию (`unsigned short`) в двоичном представлении;

в) целочисленную беззнаковую 16-битную интерпретацию (*unsigned short*) в десятичном представлении (для *printf()* используйте формат *u*: она умеет выводить любые целые в любом представлении, и требуется явное указание);

г) целочисленную знаковую 16-битную интерпретацию (*short*) в шестнадцатеричном представлении;

д) целочисленную знаковую 16-битную интерпретацию (*short*) в двоичном представлении;

е) целочисленную знаковую 16-битную интерпретацию (*short*) в десятичном представлении (для *printf()* используйте формат *d*).

## Результат

```
Print16 for minimum unsigned 16 bit number
a)0000 b)0000000000000000 c)0000 d)0000 e)0000000000000000 f)0000
Print16 for maximum unsigned 16 bit number
a)ffff b)1111111111111111 c)65535 d)ffff e)1111111111111111 f)-001
Print16 for minimum signed 16 bit number
a)8000 b)1000000000000000 c)32768 d)8000 e)1000000000000000 f)-32768
Print16 for maximum signed 16 bit number
a)7fff b)0111111111111111 c)32767 d)7fff e)0111111111111111 f)32767
Print16 for x = 5
a)0005 b)00000000000000101 c)0005 d)0005 e)00000000000000101 f)0005
Print16 for y = -5
a)ffffb b)1111111111111011 c)65531 d)ffffb e)1111111111111011 f)-005
```

**Задание Л2.35.** Разработайте на языке C/C++ функцию `void print32(void *p)`, аналогичную `print16()` для размера 32 (каждое из дублирующихся представлений — шестнадцатеричное (а) и (г), двоичное (б) и (д) — выводить один раз). Кроме целочисленных интерпретаций, `print32()` должна рассматривать память по адресу *p* как 32-битное число с плавающей запятой («вещественное») одинарной точности (*float*) и печатать:

ж) 32-битную интерпретацию с плавающей запятой (*float*) в представлении с фиксированным количеством цифр после запятой;

з) 32-битную интерпретацию с плавающей запятой (*float*) в экспоненциальном представлении.

## Результат


```
Print32 for minimum unsigned 32 bit number
a)00000000 b)00000000000000000000000000000000 c)00000000 d)00000000 e)00000000000000000000000000000000 f)00000000 g) 0.00 e)0.00e+00
Print32 for maximum unsigned 32 bit number
a)ffffffff b)11111111111111111111111111111111 c)4294967295 d)ffffffff e)11111111111111111111111111111111 f)-00000001 g) -nan e) -nan
Print32 for minimum signed 32 bit number
a)80000000 b)10000000000000000000000000000000 c)2147483648 d)80000000 e)10000000000000000000000000000000 f)-2147483648 g)-0.00 e)-0.00e+00
Print32 for maximum signed 32 bit number
a)7fffffffff b)01111111111111111111111111111111 c)2147483647 d)7fffffffff e)01111111111111111111111111111111 f)2147483647 g) nan e) nan
Print32 for int x = 5
a)00000005 b)00000000000000000000000000000101 c)00000005 d)00000005 e)00000000000000000000000000000101 f)00000005 g) 0.00 e)7.01e-45
Print32 for int y = -5
a)fffffffb b)11111111111111111111111111111011 c)4294967291 d)fffffffb e)11111111111111111111111111111011 f)-00000005 g) -nan e) -nan
Print32 for int z = 0xFF007100
a)ff007100 b)11111110000000000111000100000000 c)4278219008 d)ff007100 e)11111110000000001110001000000000 f)-16748288 g)-170727913005483667253711249801085976576.00 e
Print32 for float x = 5
a)40a00000 b)01000000010100000000000000000000 c)1084227584 d)40a00000 e)01000000010100000000000000000000 f)1084227584 g) 5.00 e)5.00e+00
Print32 for float y = -5
a)c0a00000 b)11000000010100000000000000000000 c)3231711232 d)c0a00000 e)11000000010100000000000000000000 f)-1063256064 g)-5.00 e)-5.00e+00
Print32 for float z = 0xFF007100
a)4f7f0071 b)01001111011111110000000001110001 c)1333723249 d)4f7f0071 e)01001111011111110000000001110001 f)1333723249 g)4278219008.00 e)4.28e+09
```

**Задание Л2.36.** Бонус +2 балла. Разработайте на языке C/C++ функцию *print64()*, аналогичную *print32()* для размера 64 бита и, соответственно, числа с плавающей запятой двойной точности, *double*. Аналогично Л2.35, проверьте *print64()* на граничных целочисленных 64-битных значениях, целочисленных значениях *x*, *y*, *z* и *doub*-значениях *x*, *y*, *z*.


## Результат

```
Print64 for minimum unsigned 64 bit number
a)0000000000000000 b)0000000000000000000000000000000000000000000000000000 c)0000000000000000 d)0000000000000000 e)0000000000000000000000000000000000000000000000000000 f)0000000000000000 g) 0.00 e)0.00e+00
Print64 for maximum unsigned 64 bit number
a)ffffffffffffffff b)00000000000000000000000000000000000000000000000011111111111111111111111111111111111 c)18446744073709551615 d)ffffffffffffffff e)18446744073709551615
Print64 for minimum signed 64 bit number
a)8000000000000000 b)0000000000000000000000000000000000000000000000000000 c)9223372036854775808 d)8000000000000000 e)8000000000000000
Print64 for maximum signed 64 bit number
a)7fffffffffffffff b)00000000000000000000000000000000000000000000000011111111111111111111111111111111111 c)9223372036854775807 d)7fffffffffffffff e)9223372036854775807
Print64 for long long x = 5
a)0000000000000005 b)0000000000000000000000000000000000000000000000000000 c)0000000000000005 d)0000000000000005 e)0000000000000005 f)0000000000000005 g) 0.00 e)0.00e+00
Print64 for long long y = -5
a)fffffffbfffffffb b)00000000000000000000000000000000000000000000000011111111111111111111111111111111111 c)18446744073709551611 d)fffffffbfffffffb e)18446744073709551611
Print64 for long long z = 0xFF007100
a)00000000ff007100 b)000000000000000000000000000000000000000000000000111111100000000111000100000000 c)0000004278219008 d)00000000ff007100 e)00000000ff007100 f)00000000ff007100 g) 0.00 e)0.00e+00
Print64 for double x = 5
a)4014000000000000 b)0000000000000000000000000000000000000000000000000000 c)4617315517961601024 d)4014000000000000 e)4014000000000000 f)4014000000000000 g) 5.00 e)5.00e+00
Print64 for double y = -5
a)c014000000000000 b)0000000000000000000000000000000000000000000000000000 c)13840687554816376832 d)c014000000000000 e)c014000000000000 f)c014000000000000 g) -5.00 e)-5.00e+00
Print64 for double z = 0xFF007100
a)41efe00e20000000 b)0000000000000000000000000000000000000000000000000000 c)4751262483170197504 d)41efe00e20000000 e)41efe00e20000000 f)41efe00e20000000 g) -5.00 e)-5.00e+00
Print64 for float x = 5
a)40a00000 b)01000000010100000000000000000000 c)1084227584 d)40a00000 e)01000000010100000000000000000000 f)1084227584 g) 5.00 e)5.00e+00
Print64 for float y = -5
a)c0a00000 b)11000000010100000000000000000000 c)3231711232 d)c0a00000 e)11000000010100000000000000000000 f)-1063256064 g)-5.00 e)-5.00e+00
Print64 for float z = 0xFF007100
a)4f7f0071 b)01001111011111110000000001110001 c)1333723249 d)4f7f0071 e)01001111011111110000000001110001 f)1333723249 g)4278219008.00 e)4.28e+09
```


**Задание Л2.37.** С помощью функции *printDump()* задания Л2.33 определите и выпишите в отчёт, как хранятся в памяти на платформах из таблицы Л1.1: – строки "jzyx" и "ёяюэ" из *char*; при выборе количества отображаемых байтов *N* учитывайте всю длину строки (включая завершающий нулевой символ), а не только видимые буквы; – «широкие» строки L"jzyx" и L"ёяюэ" из *wchar\_t*; при выборе *N* учитывайте всю длину строки. Результаты оформите в отчёте в виде таблицы. На MS Windows возможна (если файл исходного кода сохранён в однобайтовой кодировке windows-1251) ситуация, когда литерал L"ёяюэ" не воспринимается компилятором как корректная широкая строка. Поставьте в соответствующих ячейках отчёта прочерки.

```
Output of x86-64 gcc 13.2 (Compiler #1) ✎ ✕
A ▾ ☐ Wrap lines  Select all

ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  6A 7A 79 78 00
  D1 91 D1 8F D1 8E D1 8D 00
  6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00 00
  51 04 00 00 4F 04 00 00 4E 04 00 00 4D 04 00 00 00 00 00
```

```
Output of x86-64 clang 17.0.1 (Compiler #1) ✎ ✕
A ▾ ☐ Wrap lines  Select all

ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  6A 7A 79 78 00
  D1 91 D1 8F D1 8E D1 8D 00
  6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00 00
  51 04 00 00 4F 04 00 00 4E 04 00 00 4D 04 00 00 00 00 00
```

```
Output of x86-64 icc 2021.10.0 (Compiler #1) ✎ ✕
▾ ☐ Wrap lines  Select all
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in the se
ASM generation compiler returned: 0
  ▾ remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in the se
Execution build compiler returned: 0
Program returned: 0
  6A 7A 79 78 00
  D1 91 D1 8F D1 8E D1 8D 00
  6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00 00
  D1 00 00 00 91 00 00 00 D1 00 00 00 8F 00 00 00 D1 00 00 00 8E 00 00 00 D1 00 00 00 8D 00 00 00 00 00 00 00
```

MinGW32

```
6A 7A 79 78 00
D1 91 D1 8F D1 8E D1 8D 00
6A 00 7A 00 79 00 78 00 00 00
51 04 4F 04 4E 04 4D 04 00 00
```

MinGW64

```
[Running] cd "c:\Users\User\Documents\Workspace\2K.25\EVM\Lab2\2-lab\" && g++ Task_7.cpp -o Task_7 && "c:\Users\User\Documents\Workspace\2K.25\EVM\Lab2\2-lab\Task_7
6A 7A 79 78 00
D1 91 D1 8F D1 8E D1 8D 00
6A 00 7A 00 79 00 78 00 00 00
51 04 4F 04 4E 04 4D 04 00 00
```

Windows64

```
6A 7A 79 78 00
B8 FF FE FD 00
6A 00 7A 00 79 00 78 00 00 00
51 04 4F 04 4E 04 4D 04 00 00
```

Windows32

```
6A 7A 79 78 00
B8 FF FE FD 00
6A 00 7A 00 79 00 78 00 00 00
51 04 4F 04 4E 04 4D 04 00 00
```

	GNU/Linux 64			MS Windows 64			
Типы данных	GCC (64)	Clang (64)	Intel (64)	GCC (MinGW 32)	GCC (MinGW64)	Windows 64	Windows 32
Порядок	прямой	прямой	прямой	прямой	прямой	прямой	прямой

Ответы на вопросы в конце

- 1) Числа без знака представлены в двоичном виде. В знаковых числах первый бит отвечает за знак
- 2) Нужно инвертировать прямой двоичный код и прибавить единицу. Если число отрицательное, то бит, отвечающий за знак, не инвертируется
- 3)Порядок следования байтов определяет то, как хранятся числа в системе. Это важно знать, при работе с данными или их обменом.
- 4) Да, всякая последовательность может быть трактована как без знаковое целое число. Это потому что натуральный двоичный код является прямым отображением двоичных цифр на десятичные значения. Есть только 1 трактовка, т.к. каждая последовательность из Nбит имеет ровно одно значение в натуральном двоичном коде.
- 5) Да, в общем виде можно записать  $2^N - 1$  чисел. Да, можно единственным способом, т.к. существует взаимно однозначное соответствие между двоичным кодом и натуральными числами.



6) Да, всякая. Да, единственная.

7) Да, каждое целочисленное значение имеет своё представление в дополнительном коде. Это следует из того, что дополнительный код представляет все числа из указанного диапазона, используя Nбит. Да, единственное, т.к. существует взаимно однозначное соответствие.

8) Нет, не всякая последовательность из Nбитов может быть рассмотрена как N-битное значение с плавающей точкой. Это потому, что код с плавающей точкой стандарта IEEE 754 имеет определённую структуру, состоящую из знакового бита, порядка и мантииссы. Некоторые последовательности из Nбитов не соответствуют ни одному из значений с плавающей запятой, например, все единицы и все нули. Нет, не всегда это значение – число. Некоторые значения с плавающей точкой представляют специальные случаи, такие как бесконечность или не число

9) Нет, не каждое вещественное значение  $x$   $[\min, \max]$  имеет своё представление в коде с плавающей запятой стандарта IEEE 754. Это потому, что код с плавающей запятой имеет ограниченную точность и диапазон, и не может представить все вещественные числа с произвольной точностью. Некоторые вещественные числа могут быть округлены или приближены к ближайшему представимому значению с плавающей запятой, что может привести к потере точности или ошибкам округления.

10) Элементы массива располагаются в памяти последовательно, в порядке возрастания индексов. Например, если есть массив `int a[5] = {1, 2, 3, 4, 5};`, то его элементы будут занимать пять смежных ячеек памяти, где `a[0]` будет находиться в первой ячейке, `a[1]` во второй, и так далее. Размер каждой ячейки памяти зависит от типа данных массива. Например, если тип данных `int` занимает 4 байта, то каждый элемент массива `a` будет занимать 4 байта, и весь массив будет занимать 20 байт.

11) Необходимо умножить количество элементов на их размер.

12) Символьная информация в компьютере представляется в виде числовых кодов, соответствующих определенным символам. Код ASCII (American Standard Code for Information Interchange) является одним из самых распространенных кодов, который использует 7 бит для представления 128 символов, включая латинские буквы, цифры, знаки препинания и управляющие символы. Расширения ASCII используют 8 бит для представления 256 символов, добавляя дополнительные символы, такие как акцентированные буквы, специальные знаки и символы других языков. Кодировки Unicode являются современными стандартами, которые позволяют представлять более миллиона символов из разных письменных систем, включая кириллицу, иероглифы, эмодзи и многие другие. Существуют разные форматы Unicode, такие как UTF-8, UTF-16 и UTF-32, которые используют разное количество бит для кодирования символов.

13) Русские буквы в «классических» строках хранятся в виде однобайтовых символов, используя одну из расширенных кодировок ASCII, таких как CP1251, KOI8-R или ISO 8859-5. В этих кодировках русские буквы занимают верхнюю половину таблицы ASCII, начиная с кода 128. Например, буква А имеет код 192 в CP1251, 225 в KOI8-R и 161 в ISO 8859-5. Русские буквы в «широких» строках хранятся в виде двухбайтовых символов, используя одну из кодировок Unicode, таких как UTF-16 или UTF-32. В этих кодировках русские буквы имеют коды в диапазоне от U+0400 до U+04FF, соответствующем кириллическому блоку Unicode. Например, буква А имеет код U+0410 в UTF-16 и UTF-32.

14) Строковые функции `libc (stdlib)` определяют конец строки по специальному символу `\0`

15) Для представления строки из пяти латинских букв необходимо пять символов для узких строк и пять символов для широких строк. Это потому, что латинские буквы имеют одинаковые коды в ASCII и Unicode и занимают один байт в узких строках и два байта в широких строках. Например, строка "Hello" в узкой строке представляется как 48 65 6c 6c 6F, а в широкой строке как 00 48 00 65 00 6c 00 6c 00 6F. Для представления строки из пяти цифр также необходимо пять символов для



узких строк и пять символов для широких строк. Это потому, что цифры также имеют одинаковые коды в ASCII и Unicode и занимают один байт в узких строках и два байта в широких строках. Например, строка "12345" в узкой строке представляется как 31 32 33 34 35 , а в широкой строке как e0 31 00 32 00 33 00 34 00 35 . Для представления строки из пяти русских букв необходимо пять символов для узких строк и пять символов для широких строк, если используется кодировка Unicode. Это потому, что русские буквы имеют двухбайтовые коды в Unicode и занимают два байта в узких строках и четыре байта в широких строках. Например, строка "Привет" в узкой строке представляется как De 9F D1 80 D0 B8 D0 B2 D0 B5 D1 82 , а в широкой строке как 9F 04 80 04 41 04 32 04 35 04 82 04 . Однако, если используется одна из расширенных кодировок ASCII, то для представления строки из пяти русских букв необходимо десять символов для узких строк и пять символов для широких строк. Это потому, что русские буквы имеют однобайтовые коды в расширенных кодировках ASCII и занимают один байт в узких строках и два байта в широких строках. Например, строка "Привет" в узкой строке представляется как CF F0 E8 E2 E5 F2 в CP1251, а в широкой строке как CF 00 F0 00 E8 00 E2 00 E5 00 F2 00 . Ответ зависит от платформы, так как разные платформы могут использовать разные кодировки для представления символьной информации.