

Лабораторная работа 2 (0010 = 2)

Представление данных в ЭВМ

Задание Л2.31.

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

1. `char *p1 = reinterpret_cast<char *>(p);`
2. `unsigned short *p2 = reinterpret_cast<unsigned short *>(p);`
3. `double *p3 = reinterpret_cast<double *>(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: `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?

При выводе в поток указателя `p1` (на однобайтовый целый тип) результат отличается от `p`, `p2`, `p3` — так как в C/C++ не определён строковый тип, вместо строк используются массивы однобайтовых целых и основанные на них классы. Соответственно, оператор `<<` для указателей на типы `char / unsigned char / int8_t / uint8_t` перегружен и печатает не адрес, а значения байтов в символьной форме, начиная с указанного адреса и до ближайшего нулевого байта. Для вывода такого указателя как адреса его надо преобразовать `reinterpret_cast` к нетипизированному `void *`. Так как в Л2.31 `reinterpret_cast<void *>(p1)` — это `p`, в поток имеет смысл вывести: `p`, `p2`, `p3` и `reinterpret_cast<void *>(p1+1)`, `p2 + 1`, `p3 + 1`.

При выводе с помощью `printf()` формат вывода определяется не типом аргумента, а форматной строкой; в формате `p` (вывод целого числа как адреса) можно вывести все указатели: `p`, `p1`, `p2`, `p3` и `p1 + 1`, `p2 + 1`, `p3 + 1`.

```

#include <iostream>
#include <iomanip>

void viewPointer(void* p){
    std::cout<<"address void *p is          "<<p<<std::endl;

    static char *p1 = reinterpret_cast<char *>(p);
    std::cout<<"address char *p1 is          "<<p<<std::endl;
    std::cout<<"address char *p1 +1 is        "<<reinterpret_cast<void *>(p1+1)<<std::endl;

    unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
    std::cout<<"address unsigned short *p2 is "<<p2<<std::endl;
    std::cout<<"address unsigned short *p2 +1 is "<<p2+1<<std::endl;

    double *p3 = reinterpret_cast<double *>(p);
    std::cout<<"address double *p3 is         "<<p3<<std::endl;
    std::cout<<"address double *p3 +1 is        "<<p3+1<<std::endl;

}

int main(){
    char a='g';
    void* p=reinterpret_cast<void *>(&a);
    viewPointer(p);

    return 0;
}

```

```

address void *p is          0x7fff0012527f
address char *p1 is         0x7fff0012527f
address char *p1 +1 is      0x7fff00125280
address unsigned short *p2 is 0x7fff0012527f
address unsigned short *p2 +1 is 0x7fff00125281
address double *p3 is       0x7fff0012527f
address double *p3 +1 is    0x7fff00125287

```

Задание Л2.32.

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`:

$*p1$, $*p2$, $*p3$. Можно ли рассчитать (и, соответственно, напечатать) $*p$? 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` (`long long`), `"0123456789abcdef"` (`char[]`).

При выводе в поток режим вывода целых чисел переключается манипуляторами `hex/dec/oct` (или методом `setf()` с одноимёнными флагами, или методом `setbase()`, принимающим аргументы 16, 10, 8); режим вывода действует на все целые числа до его смены. Так, для вывода всех в шестнадцатеричном виде — в начале `main()` напечатайте манипулятор `hex`. На вывод чисел с плавающей запятой манипуляторы `hex`, `oct`, `dec` никакого влияния не оказывают.

При выводе с помощью `printf()` используйте формат вывода X с соответствующим модификатором размера перед X .

```

#include <iostream>
#include <iomanip>

void printPointer(void* p){

    static char *p1 = reinterpret_cast<char *>(p);
    std::cout<<"value of char *p1 is          "<<*p1<<std::endl;
    std::cout<<"value of char *p1 +1 is        "<<*(p1+1)<<std::endl;

    unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
    std::cout<<"value of unsigned short *p2 is  "<<*p2<<std::endl;
    std::cout<<"value of unsigned short *p2 +1 is "<<*(p2+1)<<std::endl;

    double *p3 = reinterpret_cast<double *>(p);
    std::cout<<"value of double *p3 is          "<<*p3<<std::endl;
    std::cout<<"value of double *p3 +1 is        "<<*(p3+1)<<std::endl;

}

int main(){
    std::cout<<std::hex;
    char b[] ="0123456789abcdef";
    long long a=0x1122334455667788;


    std::cout<<"test value = 0x1122334455667788"<<std::endl;
    void* p=reinterpret_cast<void *>(&a);
    printPointer(p);


    std::cout<<std::endl;

    std::cout<<"test value = \"0123456789abcdef\""<<std::endl;
    p=reinterpret_cast<void *>(&b);
    printPointer(p);

    return 0;
}

```

```
test value = 0x1122334455667788
value of char *p1 is      
value of char *p1 +1 is    w
value of unsigned short *p2 is  7788
value of unsigned short *p2 +1 is 5566
value of double *p3 is      3.84141e-226
value of double *p3 +1 is    6.9532e-310
```

```
test value = "0123456789abcdef"
value of char *p1 is      
value of char *p1 +1 is    w
value of unsigned short *p2 is  3130
value of unsigned short *p2 +1 is 3332
value of double *p3 is      9.95833e-43
value of double *p3 +1 is    1.81795e+185
```

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:

1. прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);
2. обратный (младший байт по старшему адресу, порядок Motorola, Big-Endian, 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).

При выводе в поток: однобайтовые целочисленные переменные (не только `char / unsigned char`, но часто и `int8_t / uint8_t`) выводятся в поток в виде сим-

вола, код которого равен значению переменной (независимо от манипуляторов *hex/oct/dec*). Для вывода в поток значения однобайтовой переменной в виде шестнадцатеричного, десятичного или восьмеричного числа необходимо расширить это значение до двух- или более байтового типа. Для корректного шестнадцатеричного вывода байта необходимо дополнить его нулями (беззнаковое расширение), то есть преобразовать *unsigned char* в *unsigned short* или *unsigned* и т. п. Для преобразования значения в значение в C++ используется оператор `static_cast`, то есть для печати *i*-го байта `*(p1+i)` типа *unsigned char* следует вывести в поток: `static_cast(*(p1+i))`. Вместо `static_cast` можно использовать универсальное приведение в стиле C, но это не рекомендуется.

При выводе с помощью `printf ()` используйте формат `02hhX`; в этом случае никакое преобразование не нужно.

```

#include <iostream>
#include <iomanip>

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

```

```

int main(){

    std::cout<<"item 1"<<std::endl;
    int a=5;
    void* p=reinterpret_cast<void *>(&a);
    printDump(p,4);

    std::cout<<"item 2"<<std::endl;
    int b[]={5,-5,(int)0xFF007100};
    p=reinterpret_cast<void *>(&b);
    printDump(p,12);

    std::cout<<"item 3"<<std::endl;
    double c=1.5;
    p=reinterpret_cast<void *>(&c);
    printDump(p,8);

    return 0;
}

```

```

item 1
05 00 00 00
item 2
05 00 00 00 FB FF FF FF 00 71 00 FF
item 3
00 00 00 00 00 00 F8 3F

```

Бонус +2 балла за платформу. При подготовке к работе выполните измерения на платформе, где архитектура процессора отлична от x86/x86-64.

Mac os, процессор M1, архитектура ARM

item 1

05 00 00 00

item 2

05 00 00 00 FB FF FF FF 00 71 00 FF

item 3

00 00 00 00 00 00 F8 3F

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*:

- целочисленную беззнаковую 16-битную интерпретацию (*unsigned short*) в шестнадцатеричном представлении;
- целочисленную беззнаковую 16-битную интерпретацию (*unsigned short*) в двоичном представлении;
- 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);
- целочисленную знаковую 16-битную интерпретацию (*short*) в шестнадцатеричном представлении;
- целочисленную знаковую 16-битную интерпретацию (*short*) в двоичном представлении;
- целочисленную знаковую 16- битную интерпретацию (*short*) в десятичном представлении (для *printf* () используйте формат *d*).

Штраф –2 балла, если количество выводимых цифр двоичного представления отлично от количества бит в числе ($\frac{16}{4}=4$ для *print16()*) либо если количество выводимых цифр шестнадцатеричного представления отлично от количества тетрад в числе ($\frac{16}{4}=4$ для *print16()*).

Бонус +1 балл, если вывод *print16()* занимает одну строку (так на экран поместится больше чисел).

Бонус +2 балла, если при этом младшая цифра находится под младшей цифрой предыдущей строки (чего можно добиться заданием ширины поля вывода).

Для получения различных интерпретаций участка памяти по адресу p в C++ можно использовать преобразование указателя p в указатель на другой тип оператором `reinterpret_cast` с последующим разыменованием. Обратите внимание, что разыменованный указатель является lvalue (может стоять слева от оператора присваивания), то есть различные интерпретации можно использовать не только для просмотра, но и для изменения участка памяти по адресу p .

Так, целочисленная беззнаковая интерпретация 16 бит памяти по адресу p для (а) и (в) — `*(reinterpret_cast<unsigned short*>(p))`, знаковая для (г) и (е) — `*(reinterpret_cast<short*>(p))` (см. разделы 7.2 и 7.3).

Вместо `reinterpret_cast` можно использовать универсальное приведение в стиле C, но его использование в коде на C++ не рекомендуется.

Обратите внимание, что преобразование значения в значение оператором `static_cast` или приведением в стиле C не обеспечивает необходимого эффекта; хотя для целых типов одного размера *signed* ↔ *unsigned* преобразование значения в значение чаще всего приводит к тому же результату, что `reinterpret_cast` + разыменование, но уже для плавающей запятой это не так.

При выводе в поток двоичное (битовое) представление чисел можно получить, используя шаблон `std::bitset<N>`, где N — количество бит в представлении — необходимо задать вручную.

При выводе с помощью `printf()` в новейших версиях GCC можно использовать форматы b и B с соответствующим модификатором размера; в более старых, где b и B не поддерживаются, необходимо вручную выделять и выводить биты числа.

Отсутствие двоичного представления (б)/(д) не штрафует в том случае, если для студента не составляет труда прочитать шестнадцатеричное представление (а)/(г) и записать по нему двоичное для любого выбранного числа.

Проверьте работу функции `print16()` на 16-битных целочисленных переменных, принимающих следующие значения:

- минимальное целое 16-битное значение без знака;
- максимальное целое 16-битное значение без знака;
- минимальное целое 16-битное значение со знаком;

- максимальное целое 16-битное значение со знаком;
- значение x , соответствующее варианту (таблица Л2.1);
- значение y , соответствующее варианту (таблица Л2.1);

(запишите каждое из значений в 16 -битную целочисленную переменную и передайте её адрес функции).

- Убедитесь, что (б) и (д) — одно и то же двоичное представление.
- Убедитесь, что (а) и (г) — одно и то же шестнадцатеричное представление.

Шестнадцатеричный формат вывода для целочисленных переменных используется как компактная запись двоичного кода, а не альтернативное представление значения, поэтому результат совпадает для беззнаковой и знаковой целочисленных интерпретаций p (и равен беззнаковой интерпретации в шестнадцатеричной системе счисления).

Варианты значений

Таблица Л2.1

$(N^{\circ} - 1) \% 2 + 1$	Вариант
1	$x = 9, y = -9, z = 0x88776155$
2	$x = 5, y = -5, z = 0xFF007100$

Измените функцию `print16()` так, чтобы убрать дублирование (сохраните полный вариант как комментарий), и в дальнейшем пользуйтесь вариантом без дублей.

```

#include <iostream>
#include <iomanip>
#include <bitset>

void print16(void* p){
    unsigned short int a=*reinterpret_cast<unsigned short int*>(p);
    short b=(reinterpret_cast<short *>(p));

    std::bitset<16> a2{a};
    std::bitset<16> b2{(unsigned short)b};

    std::cout
    <<std::setw(10)<<std::hex<<a
    <<std::setw(18)<<a2
    <<std::setw(8)<<std::dec<<a
    <<std::setw(10)<<std::hex<<b
    <<std::setw(18)<<b2
    <<std::setw(8)<<std::dec<<b
    <<std::endl;

}

int main(){
    std::cout<<"\nМинимальное целое 16-битное значение без знака:"<<std::endl;
    unsigned short int a=0;
    void* p=reinterpret_cast<void *>(&a);
    print16(p);

    std::cout<<"\nМаксимальное целое 16-битное значение без знака:"<<std::endl;
    unsigned short int b=65535;
    p=reinterpret_cast<void *>(&b);
    print16(p);

    std::cout<<"\nМинимальное целое 16-битное значение со знаком:"<<std::endl;
    short int c=-32768;
    p=reinterpret_cast<void *>(&c);
    print16(p);

    std::cout<<"\nМаксимальное целое 16-битное значение со знаком:"<<std::endl;
    short int d=32767;
    p=reinterpret_cast<void *>(&d);
    print16(p);
}

```

```

std::cout<<"\nЗначение x = 5:"<<std::endl;
short int e=5;
p=reinterpret_cast<void *>(&e);
print16(p);

std::cout<<"\nЗначение y = -5:"<<std::endl;
short int f=-5;
p=reinterpret_cast<void *>(&f);
print16(p);
return 0;
}

```

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 (*f float*) и печатать:

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

При выводе в поток режим вывода чисел с плавающей запятой переключается манипуляторами *fixed/scientific*. Количество цифр после запятой задаётся манипулятором `setprecision()`.

При выводе с помощью `printf()` используйте форматы вывода *f* (так, `5.2f` — 5 цифр всего, 2 после запятой) и *e* с соответствующим модификатором размера. Проверьте работу `print32()` на 32- битных целочисленных переменных, принимающих следующие значения:

- минимальное целое 32-битное значение без знака;
- максимальное целое 32-битное значение без знака;
- минимальное целое 32-битное значение со знаком;
- максимальное целое 32-битное значение со знаком;
- целочисленное значение x , соответствующее варианту (таблица Л2.1);
- целочисленное значение y , соответствующее варианту (таблица Л2.1);

- целочисленное значение z , соответствующее варианту (таблица Л2.1);
аналогично `print16()`, а также 32 - битных переменных с плавающей запятой (*float*):
- *float*-значение x , соответствующее варианту (таблица Л2.1);
- *float*-значение y , соответствующее варианту (таблица Л2.1);
- *float*-значение z , соответствующее варианту (таблица Л2.1).

Сравните структуру целой переменной и *float*-переменной, имеющих равные значения (в частности, x). Сравните *float*-значения x и $y = -x$.

```

#include <iostream>
#include <iomanip>
#include <bitset>

void print32(void* p){
    unsigned int a=*reinterpret_cast<unsigned int*>(p);
    int b=(reinterpret_cast<int *>(p));
    float c=(reinterpret_cast<float *>(p));

    std::bitset<32> a2{a};
    std::bitset<32> b2{(unsigned )b};

    std::cout
    <<std::setw(9)<<std::hex<<a          //a
    <<std::setw(33)<<a2                //6
    <<std::setw(12)<<std::dec<<a        //B
    <<std::setw(12)<<std::dec<<b        //e

    <<std::setw(50)<<std::setprecision(4)<<std::fixed<<c      //ж
    <<std::setw(25)<<std::setprecision(4)<<std::scientific<<c  //з
    <<std::endl;

}

int main(){
    std::cout<<"\n"
    <<std::setw(9)<<"hex"
    <<std::setw(33)<<"binary"
    <<std::setw(12)<<"udec"
    <<std::setw(12)<<"dec"
    <<std::setw(50)<<"fixed"
    <<std::setw(25)<<"exp"
    <<std::endl;

    std::cout<<"\nМинимальное целое 32-битное значение без знака:"<<std::endl;
    unsigned int a=0;
    void* p=reinterpret_cast<void *>(&a);
    print32(p);

    std::cout<<"\nМаксимальное целое 32-битное значение без знака:"<<std::endl;
    unsigned int b=4294967295 ;
    p=reinterpret_cast<void *>(&b);

```

```
print32(p);
```

```
std::cout<<"\nМинимальное целое 32-битное значение со знаком:"<<std::endl;
int c=-2147483648;
p=reinterpret_cast<void *>(&c);
print32(p);
```

```
std::cout<<"\nМаксимальное целое 32-битное значение со знаком:"<<std::endl;
int d=2147483647;
p=reinterpret_cast<void *>(&d);
print32(p);
```

```
std::cout<<"\nЦелочисленное значение x = 5:"<<std::endl;
int e=5;
p=reinterpret_cast<void *>(&e);
print32(p);
```

```
std::cout<<"\nЦелочисленное значение y = -5:"<<std::endl;
int f=-5;
p=reinterpret_cast<void *>(&f);
print32(p);
```

```
std::cout<<"\nЦелочисленное значение z = 0xFF007100:"<<std::endl;
unsigned int g=0xFF007100;
p=reinterpret_cast<void *>(&g);
print32(p);
```

```
std::cout<<"\nfloat-значение x = 5:"<<std::endl;
float h=5;
p=reinterpret_cast<void *>(&h);
print32(p);
```

```
std::cout<<"\nfloat-значение y = -5:"<<std::endl;
h=-5;
p=reinterpret_cast<void *>(&h);
print32(p);
```

```
std::cout<<"\nfloat-значение z = 0xFF007100:"<<std::endl;
h=0xFF007100;
p=reinterpret_cast<void *>(&h);
print32(p);
```

```
    return 0;  
}
```

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 *double*-значениях *x*, *y*, *z*.

Штраф –2 балла, если для *print32()* либо *print64()* количество выводимых цифр двоичного представления отлично от количества бит в числе либо количество выводимых цифр шестнадцатеричного — от количества тетрад.

Бонус +1 балл, если вывод *print32()* (и *print64()*, если она реализована) занимает одну строку.

Бонус +2 балла, если при этом младшая цифра находится под младшей цифрой предыдущей строки.


```

#include <iostream>
#include <iomanip>
#include <bitset>

void print64(void* p){
    unsigned long long int a=*reinterpret_cast<unsigned long long int*>(p);
    long long int b=(reinterpret_cast<int long long*>(p));
    double d=(reinterpret_cast<double *>(p));
    std::bitset<64> a2{a};
    std::bitset<64> b2{(unsigned)b};

    std::cout
    <<std::setw(17)<<std::hex<<a           //a
    <<std::setw(65)<<a2                     //b
    <<std::setw(22)<<std::dec<<a           //B
    <<std::setw(22)<<std::dec<<b           //E
    <<std::setw(50)<<std::fixed<<d        //ж
    <<std::setw(25)<<std::scientific<<d
    <<std::endl;

}

int main(){
    std::cout<<"\n"
    <<std::setw(17)<<"hex"
    <<std::setw(65)<<"binary"
    <<std::setw(22)<<"udec"
    <<std::setw(22)<<"dec"
    <<std::setw(50)<<"fixed"
    <<std::setw(25)<<"exp"
    <<std::endl;

    std::cout<<"\nМинимальное целое 64-битное значение без знака:"<<std::endl;
    unsigned int a=0;
    void* p=reinterpret_cast<void *>(&a);
    print64(p);

    std::cout<<"\nМаксимальное целое 64-битное значение без знака:"<<std::endl;
    unsigned long long b = 18446744073709551615ULL;
    p=reinterpret_cast<void *>(&b);
    print64(p);

```

```
std::cout<<"\nМинимальное целое 64-битное значение со знаком:"<<std::endl;
long long int c=-9223372036854775807LL-1;
p=reinterpret_cast<void *>(&c);
print64(p);
```

```
std::cout<<"\nМаксимальное целое 64-битное значение со знаком:"<<std::endl;
long long int d=9223372036854775807;
p=reinterpret_cast<void *>(&d);
print64(p);
```

```
std::cout<<"\nЦелочисленное значение x = 5:"<<std::endl;
long long int e=5;
p=reinterpret_cast<void *>(&e);
print64(p);
```

```
std::cout<<"\nЦелочисленное значение y = -5:"<<std::endl;
long long int f=-5;
p=reinterpret_cast<void *>(&f);
print64(p);
```

```
std::cout<<"\nЦелочисленное значение z = 0xFF007100:"<<std::endl;
unsigned int g=0xFF007100;
p=reinterpret_cast<void *>(&g);
print64(p);
```

```
std::cout<<"\ndouble-значение x = 5:"<<std::endl;
double h=5.0;
p=reinterpret_cast<void *>(&h);
print64(p);
```

```
std::cout<<"\ndouble-значение y = -5:"<<std::endl;
h=-5.0;
p=reinterpret_cast<void *>(&h);
print64(p);
```

```
std::cout<<"\ndouble-значение z = 0xFF007100:"<<std::endl;
h=0xFF007100;
p=reinterpret_cast<void *>(&h);
print64(p);
```

```
return 0;
```

```
}
```

Задание Л2.37. Бонус +2 балла для пар, обязательное для троек.

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.

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, инициализированной строкой при создании — размер массива равен размеру строки, а полный размер массива в байтах можно определить оператором *sizeof*.

```

#include <iostream>
#include <cstring>
#include <locale.h>

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

int main() {
    setlocale(LC_ALL, "russian");

    char str1[] = "jzyx";
    char str2[] = "ёяюэ";

    std::cout << "Хранение в памяти строки \"jzyx\":\n";
    printDump(str1, strlen(str1) + 1); // Учитываем завершающий нулевой символ
    std::cout << "Хранение в памяти строки \"ёяюэ\":\n";
    printDump(str2, strlen(str2) + 1); // Учитываем завершающий нулевой символ

    // Широкие строки
    wchar_t str3[] = L"jzyx";
    wchar_t str4[] = L"ёяюэ";

    std::cout << "Хранение в памяти широкой строки L\"jzyx\":\n";
    printDump(str3, wcslen(str3) * sizeof(wchar_t) + sizeof(wchar_t)); // Учитываем завершающий
    std::cout << "Хранение в памяти широкой строки L\"ёяюэ\":\n";
    printDump(str4, wcslen(str4) * sizeof(wchar_t) + sizeof(wchar_t)); // Учитываем завершающий

    return 0;
}

```

Microsoft 64

6A 7A 79 78 00

Хранение в памяти строки "ёяюэ":

B8 FF FE FD 00

Хранение в памяти широкой строки L"jzyx":

6A 00 7A 00 79 00 78 00 00 00

Хранение в памяти широкой строки L"ёяюэ":

51 04 4F 04 4E 04 4D 04 00 00

MINGV gcc

Хранение в памяти строки "jzyx":

6A 7A 79 78 00

Хранение в памяти строки "ёяюэ":

D1 91 D1 8F D1 8E D1 8D 00

Хранение в памяти широкой строки L"jzyx":

6A 00 7A 00 79 00 78 00 00 00

Хранение в памяти широкой строки L"ёяюэ":

51 04 4F 04 4E 04 4D 04 00 00

Linux gcc

Хранение в памяти строки "jzyx":

6A 7A 79 78 00

Хранение в памяти строки "ёяюэ":

D1 91 D1 8F D1 8E D1 8D 00

Хранение в памяти широкой строки L"jzyx":

6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00 00

Хранение в памяти широкой строки L"ёяюэ":

51 04 00 00 4F 04 00 00 4E 04 00 00 4D 04 00 00 00 00 00 00

Linux clang

Хранение в памяти строки "jzyx":

6A 7A 79 78 00

Хранение в памяти строки "ёяюэ":

D1 91 D1 8F D1 8E D1 8D 00

Хранение в памяти широкой строки L"jzyx":

6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00 00

Хранение в памяти широкой строки L"ёяюэ":

51 04 00 00 4F 04 00 00 4E 04 00 00 4D 04 00 00 00 00 00 00

Хранение в памяти строки "jzyx":

6A 7A 79 78 00

Хранение в памяти строки "ёяюэ":

D1 91 D1 8F D1 8E D1 8D 00

Хранение в памяти широкой строки L"jzyx":

6A 00 00 00 7A 00 00 00 79 00 00 00 78 00 00 00 00 00

Хранение в памяти широкой строки L"ёяюэ":

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

Л2.1. Дополнительные бонусные и штрафные баллы

-3 балла за утечку памяти (выделенные, но не освобождённые блоки динамической памяти).

Л2.3. Вопросы

1. Как представляются целые числа со знаком и без знака?

Целые числа со знаком представляются в виде чисел, которые могут быть как положительными, так и отрицательными.

В компьютерной технике применяются три формы записи целых чисел со знаком: прямой код, обратный код, дополнительный код. (могут занимать 16264 байта. Старший бит указывает на знак числа)

Целые числа без знака представляются только положительными числами. (занимают 1-2 байта)<

2. Как перевести число в дополнительный код?

1. Перевести числа в двоичную систему
2. Написать обратный код этого числа
3. Прибавить единицу к младшему разряду<

3. Для чего нужно знать порядок следования байтов на вашем компьютере?

При работы с данными из других систем могут возникнуть проблемы с чтением и работой с полученной информацией. (Например, передача данных между 2мя компьютерами. 1й компьютер записывает данные "слева-направо" Big-endian, 2й компьютер читает данные "справа-налево" Little-endian. Решение данной проблемы - применение дополнительной конвертации данных).

4. Всякая ли последовательность из N битов ($N \in \{16, 32, 64\}$) может быть рассмотрена как N -битное беззнаковое целое число в натуральном двоичном коде (беззнаковом коде ЭВМ)? Единственное ли беззнаковое значение можно сопоставить этой последовательности?

Да, является. (N -битное беззнаковое целое число принимает значения от 0 до 2^N).
Да, единственное.

5. Каждое ли целочисленное значение $x \in [0, 2^N)$ имеет своё представление в натуральном двоичном коде? Единственная ли последовательность из N битов соответствует каждому значению x ?

Каждое значение x имеет свое представление в натуральном двоичном коде. При этом любая последовательность из N битов соответствует только одному единственному числу из $[0, 2^N)$.

- Натуральный двоичный код - обычное беззнаковое представление числа в двоичной системе исчисления.<

6. Всякая ли последовательность из N битов может быть рассмотрена как N -битное знаковое целое число в дополнительном коде (знаковом коде ЭВМ)? Единственное ли знаковое значение можно сопоставить этой последовательности?

- Дополнительный код положительного числа совпадает с прямым кодом. (знаковый бит - 0) Для отрицательного числа дополнительный код образуется путем получения обратного кода и добавлением к младшему разряду единицы.
Да, всякое. Каждый набор битов дает 1 определенное число.

7. Каждое ли целочисленное значение $x \in [-2^{N-1}, 2^{N-1})$ имеет своё представление в дополнительном коде? Единственная ли последовательность из N битов соответствует каждому значению x ?

При использовании алгоритма преобразования числа в дополнительный код из предыдущего вопроса очевидно, что Каждое ли целочисленное значение $x \in [-2^{N-1}, 2^{N-1})$ имеет своё представление

8. Всякая ли последовательность из N битов ($N \in \{32, 64\}$) может быть рассмотрена как N -битное значение с плавающей запятой? Всегда ли это значение — число?

9. Каждое ли вещественное значение $x \in [\min, \max]$ имеет своё представление в коде с плавающей запятой стандарта IEEE 754 (*float/double* ЭВМ)?

10. Как располагаются в памяти элементы массива?

Элементы массива располагаются в памяти последовательно друг за другом в соответствии с их индексами.

11. Как найти размер массива, зная размер элемента и их количество?

размер элемента * количество

12. Как представляется символьная информация в компьютере в кодах ASCII, расширениях ASCII и различных кодировках Unicode?

13. Как хранятся русские буквы в «классических» и «широких» строках?

15. Как строковые функции libc (stdlib) определяют конец строки?

16. Сколько символов (для узких строк — узких символов *char*, для широких — *wchar_t*) необходимо для представления строки из пяти латинских букв? Цифр? Русских букв? Зависит ли ответ от платформы?