**Отчёт по лабораторной работе №2Представление данных в ЭВМ**

**Группа 8 Лазу Игорь, Меркулов Сергей**

**ОС: MSWindows**

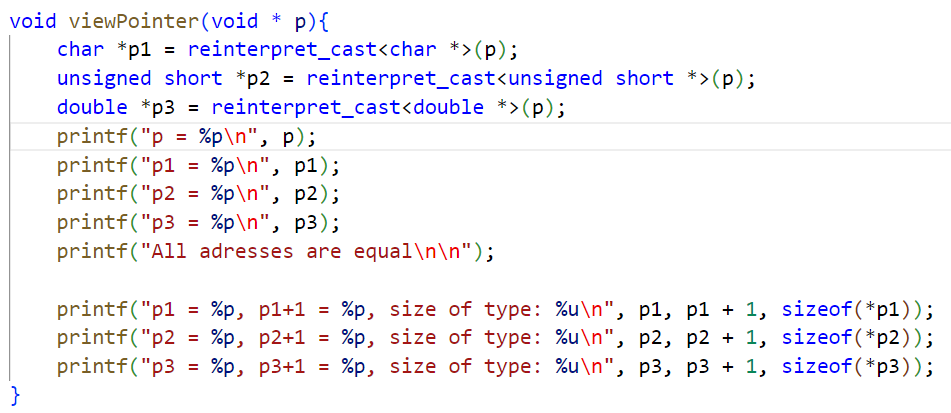
**Компилятор: gccversion13.2.0**

**Разрядность сборки: 64 бит**

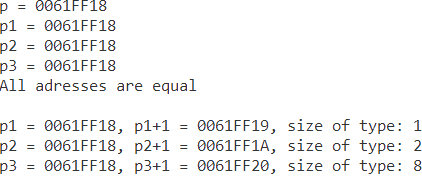
**Архитектура процессора: 64 бит**

**Назначение платформы: общее**

**Задание Л2.з1.** Разработайте функцию void𝑣𝑖𝑒𝑤𝑃𝑜𝑖𝑛𝑡𝑒𝑟(void \* 𝑝), которая принимает нетипизированный указатель 𝑝, преобразует его в типизированные: а) char \*p1 = reinterpret\_cast(p); б) unsignedshort \*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: 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?

**Код: **

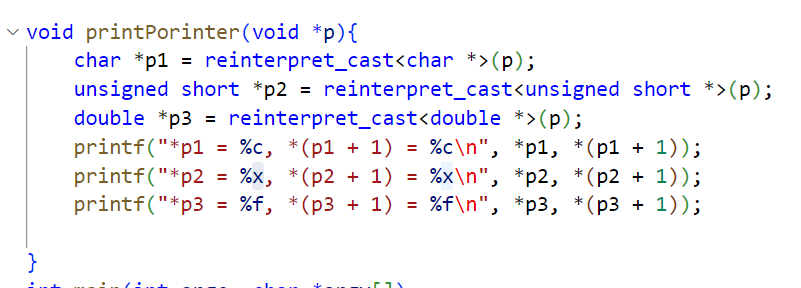
**Результат:**

****

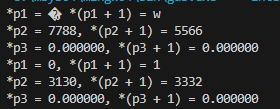
**Какова разница между p и p + 1 в байтах?**

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

**Задание Л2.з2.**Разработайте функцию void𝑝𝑟𝑖𝑛𝑡𝑃𝑜𝑖𝑛𝑡𝑒𝑟(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** 

**Результат**

****

**Можно ли рассчитать (и, соответственно, напечатать) \*P?**

Да, ведь нам известен адрес переменной

**Задание Л2.з3.**Разработайте функцию void𝑝𝑟𝑖𝑛𝑡𝐷𝑢𝑚𝑝(void \* 𝑝, size\_t𝑁), которая принимает нетипизированный указатель 𝑝, преобразует его в типизированный указатель на байт 𝑢𝑛𝑠𝑖𝑔𝑛𝑒𝑑𝑐ℎ𝑎𝑟 \* 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:

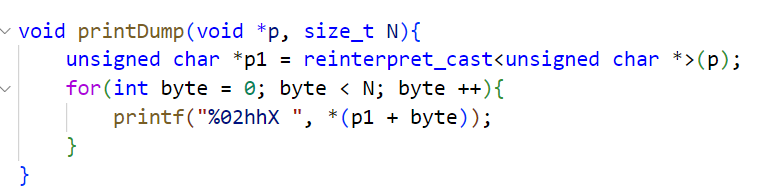
а) прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);

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**

****

**Порядок прямой**

**Задание Л2.з4.** 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;

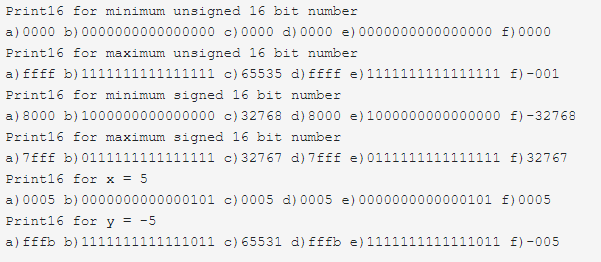
в) целочисленную беззнаковую 16-битную интерпретацию (𝑢𝑛𝑠𝑖𝑔𝑛𝑒𝑑 𝑠ℎ𝑜𝑟𝑡) в десятичном представлении (для 𝑝𝑟𝑖𝑛𝑡𝑓() используйте формат 𝑢: она умеет выводить любые целые в любом представлении, и требуется явное указание);

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**

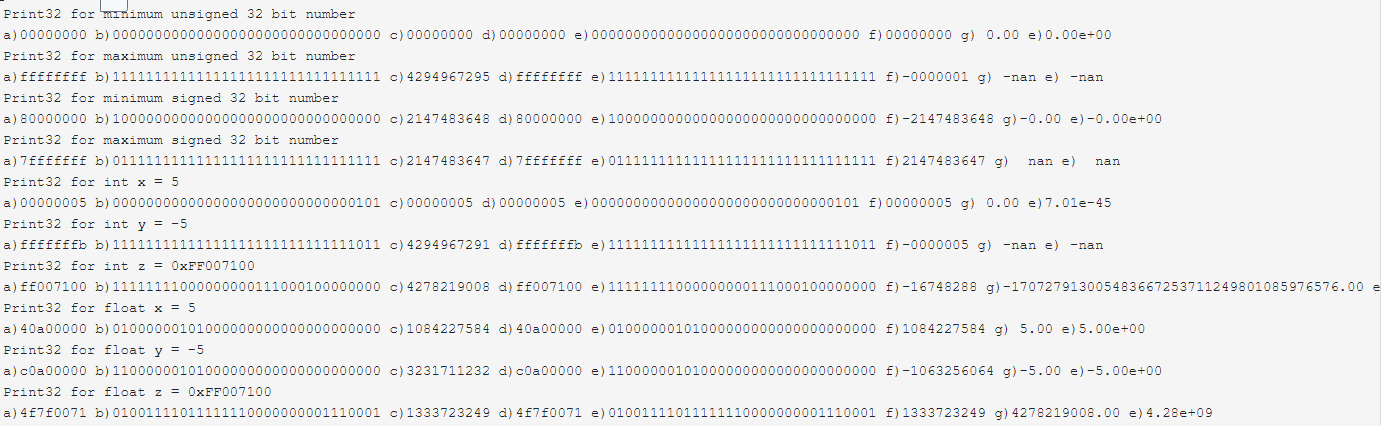
****

**Задание Л2.з5.** 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-битноe числo с плавающей запятой («вещественноe») одинарной точности (𝑓𝑙𝑜𝑎𝑡) и печатать:

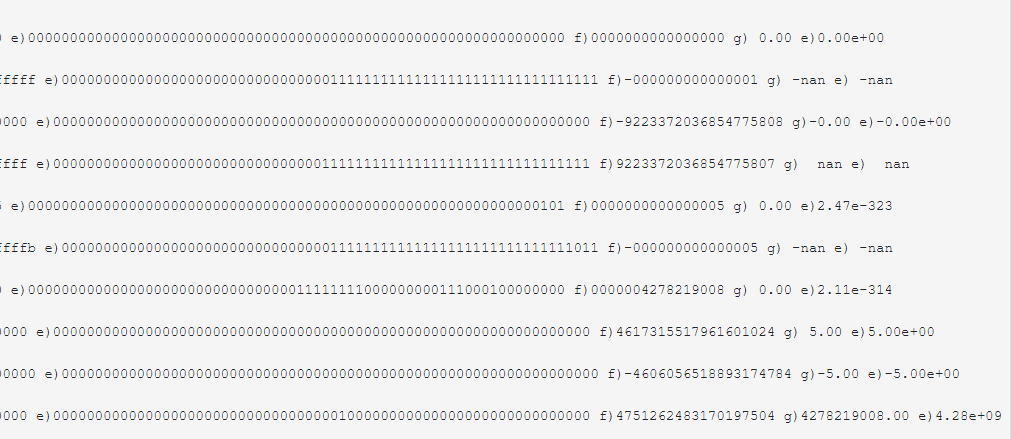
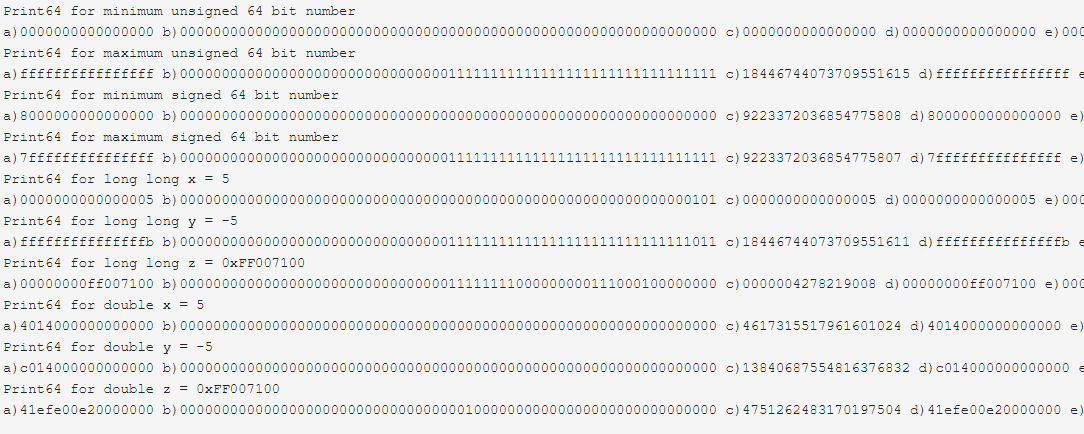
ж) 32-битную интерпретацию с плавающей запятой (𝑓𝑙𝑜𝑎𝑡) в представлении с фиксированным количеством цифр после запятой;

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

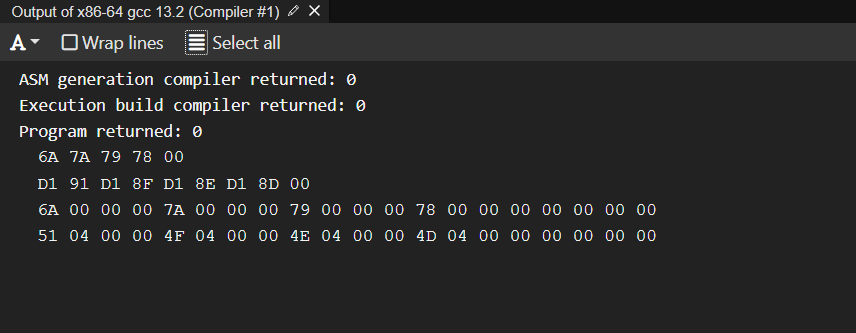
**Результат**

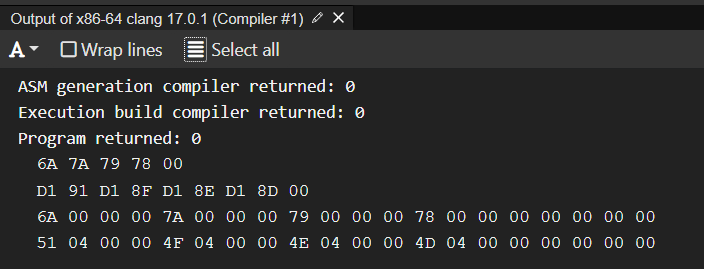
****

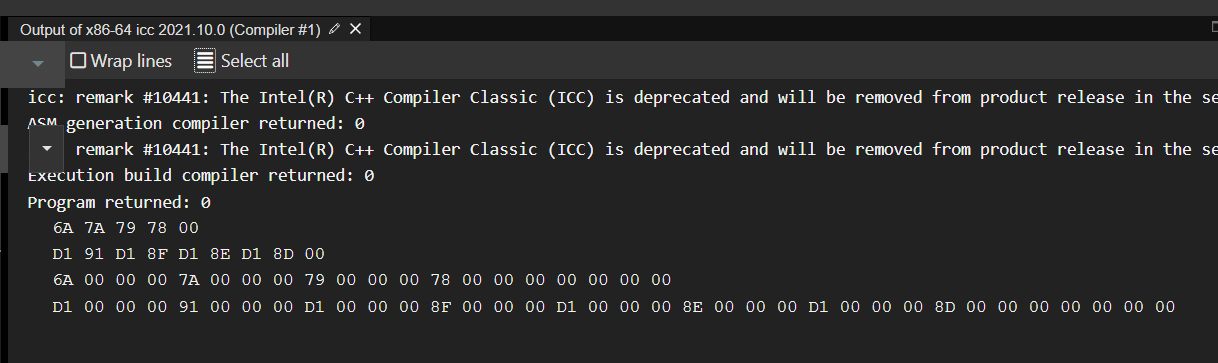
**Задание Л2.з6.** Бонус +2 балла. Разработайте на языке C/C++ функцию 𝑝𝑟𝑖𝑛𝑡64(), аналогичную 𝑝𝑟𝑖𝑛𝑡32() для размера 64 бита и, соответственно, числа с плавающей запятой двойной точности, 𝑑𝑜𝑢𝑏𝑙𝑒. Аналогично Л2.з5, проверьте 𝑝𝑟𝑖𝑛𝑡64() на граничных целочисленных 64-битных значениях, целочисленных значениях 𝑥, 𝑦, 𝑧 и 𝑑𝑜𝑢𝑏-значениях 𝑥, 𝑦, 𝑧.

**Результат**

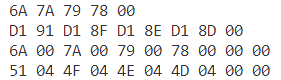
**Задание Л2.з7.**С помощью функции 𝑝𝑟𝑖𝑛𝑡𝐷𝑢𝑚𝑝() задания Л2.з3 определите и выпишите в отчёт, как хранятся в памяти на платформах из таблицы Л1.1: – строки "jzyx" и "ёяюэ" из 𝑐ℎ𝑎𝑟; при выборе количества отображаемых байтов 𝑁 учитывайте всю длину строки (включая завершающий нулевой символ), а не только 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.



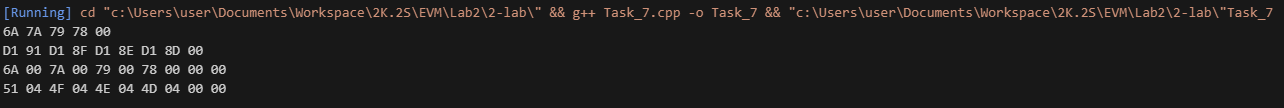




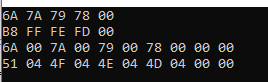
MinGW32



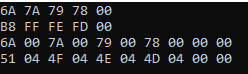
MinGW64



Windows64



Windows32



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 точной стандарта IEEE 754 имеет определённую структуру, состоящую из знакового бита, порядка и мантиссы. 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 х [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 впервой ячейке, а[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 а 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 6с 6с 6F , а в широкой строке как [00 48 00 65 00](tel:00%2048%2000%2065%2000) 6C 00 6C 00 6F . Дляпредставления 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](tel:31%2032%2033%2034%2035) , а в широкой строке как е0 [31 00 32 00 33](tel:31%2000%2032%2000%2033) 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](tel:04%2080%2004%2041%2004) [32 04 35 04 82](tel:32%2004%2035%2004%2082) 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.