# Лекция 3. Символьные строки и форматированный ввод-вывод.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», осенний семестр
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

# Ввод-вывод (I/O)

Операции **ввода-вывода** являются базовым процессом внутри которого рассматривается **корректная** передача информации от внешнего мира к обработчику информации и наоборот.
**Ввод** – данные/сигнал **получаемые (input)** обработчиком информации.
**Вывод** – данные/сигнал **посланные (output)** обработчиком информации.

- stdio.h (от англ. standard input/output header — стандартный заголовочный файл ввода-вывода)

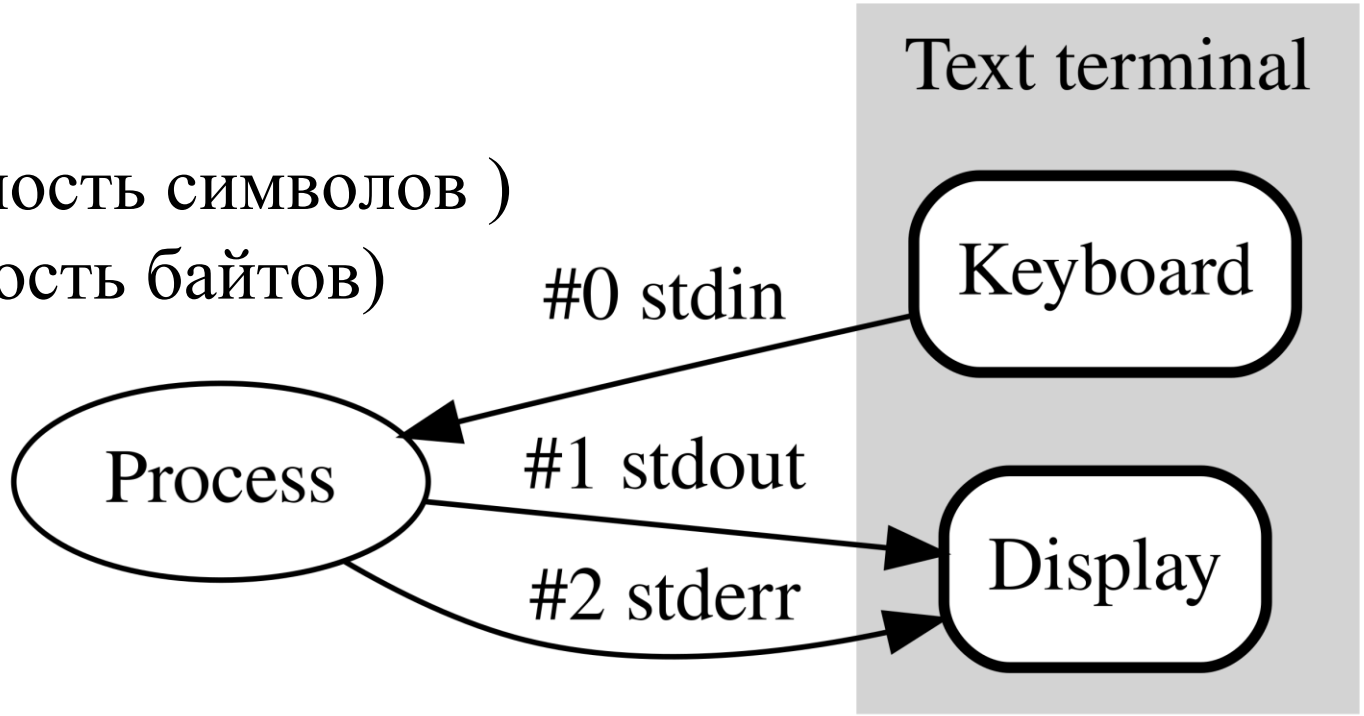
# Поток данных

Под **потоком данных** понимается набор информации, выходной или входной от:

- Устройства (Терминал)
- Файла\Сокета

Потоки так же могут быть:

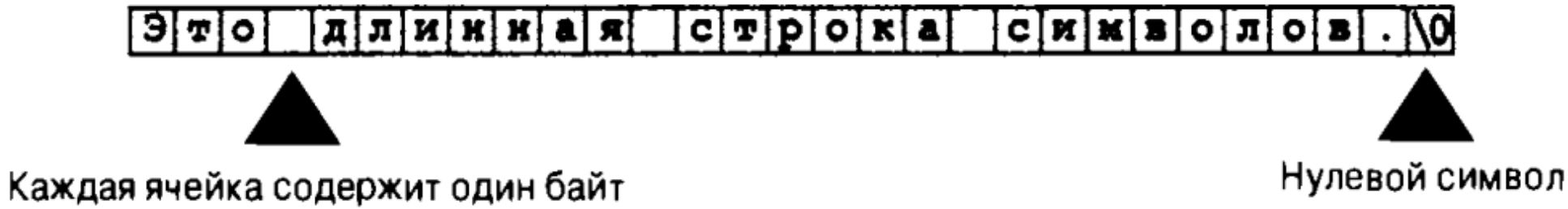
- Текстовым (Последовательность символов )
- Бинарным (Последовательность байтов)



# Символьная строка

~~str(‘Привет мир’)~~ – это Python нотация, где по сути любой объект это текстовый поток или бинарный, но речь не о нём.

Символьная строка в языке Си – это последовательность из одного или большего количества символов, которая заканчивается управляющим символом ‘\0’;
‘0\’ – называют нулевым символом строки



# Ошибка с написанием кавычек

В языке программирования Си одинарные и двойные кавычки имеют колоссальное отличие.
“Эта длинная строка символов.” – двойные кавычки определяют строковый литерал, и не считаются частью строки.

‘И’, ‘В’, ‘Т’ – одинарные кавычки идентифицируют 1 символ, который сопоставим с целочисленным типом данных char (1byte).

Тогда как же представлять строку данных?
Что за тип данных используется для представления строки?

# Строковый массив данных

Вместо специального типа данных в Си для хранения строковых данных применяются **массив** типа **char**.
Каждый символ по отдельности хранится в своей ячейки памяти в естественном виде, также как он и расположен в строке.
Из вышесказанного следует, что память для хранения строки конечного размера выделяется последовательно и единственным “куском”.
На окончание строки указывает символ ‘\0’

# Массивы

**Массив** – (массив переменных) несколько ячеек памяти, расположенные подряд, которые имеют один тип данных.

В случае строки символьных данных это **массив** элементов типа **char**

# Пример строки символьных данных

```
char symbol_str[29] = "Это длинная строка символов.";
```

`char` – указывает на тип данных.

`symbol_str`– название переменной.

`[]` – указывает, что мы используем массивы (оператор массива).

`29` – количество переменных типа char выделяемое в памяти последовательно.

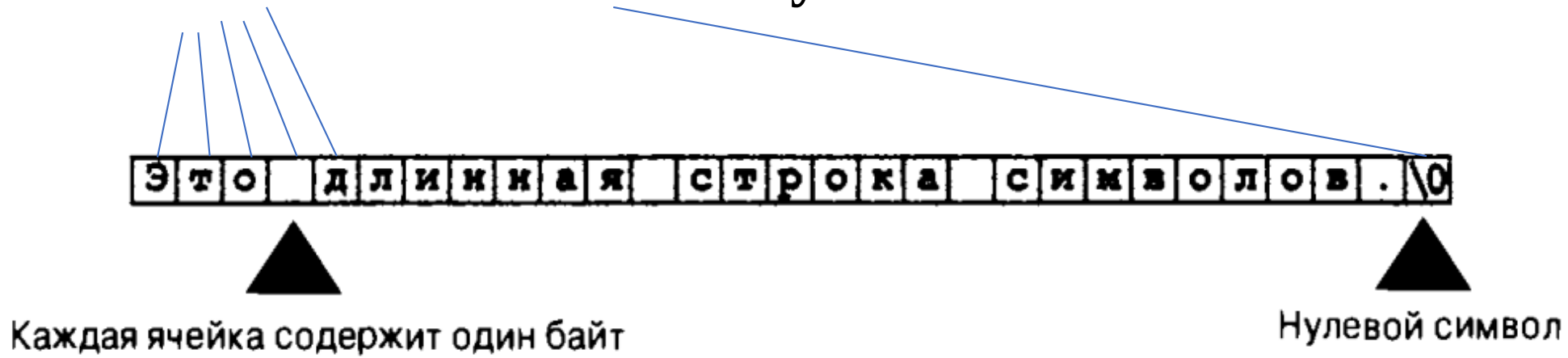
Строки в языке Си часто называют строки типа **Си-style**

# Длинна строки и размер в памяти

```
char symbol_str[29] = "Это длинная строка символов.";
```

**Если под строку выделено 29 byte, значит она может содержать не более 28 символов, плюс 1 нулевой символ.**

**Тип char ( 1byte)**
**1+1+1+1+1 … +1 = 29 byte**



# Ввод строки с использованием scanf()

```c
#include<stdio.h>

int main()
{
    char cstr[40];
    printf("Введите слово:");
    scanf("%s",cstr);
    printf("Ввод окончен\n");
    printf("Результат ввода: %s \n", cstr);
    return 0;
}
```

# Ввод строки с использованием scanf()

```
char cstr[40]; //Создаем переменную char[] размером 40 байт
```

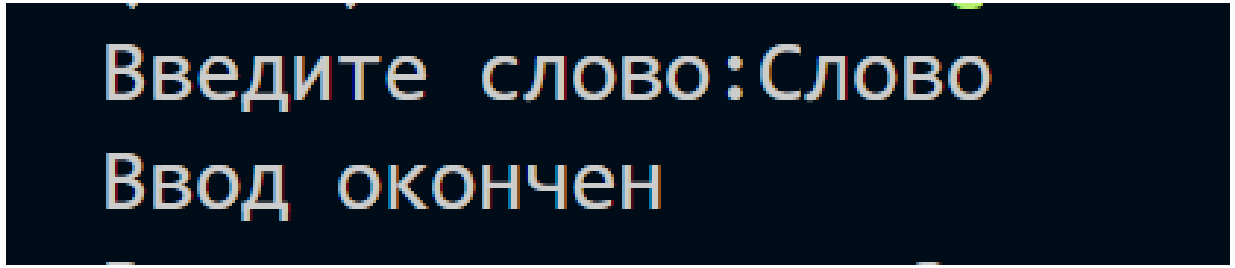
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | ? | t | ? | ? | ? | ? | ? | r | ? | n | Ё | ? | \0 | ? | ? | ? | ? | R | h | ? | \0 | ? | ? | ? | $ | # | ? | ? | ^ | ? | ? | ? | ? | ? | ? | ? |

В **памяти** при запуске программы будет зарезервировано 40 byte памяти из расчёта char = 1byte * 40.
Как узнать сколько весит тип данных ??? printf(“%d”,sizeof(char)) => 1

Что сейчас находится в памяти, человек и компьютер знать не могут. Здесь и сейчас из свободного **набора byte** выдана область размером 40 byte, “процессор не тратит время на их очистку, если программист об этом не просит”.

# Ввод строки с использованием scanf()

```
printf("Введите слово:");
scanf("%s",cstr);
printf("Ввод окончен\n");
```



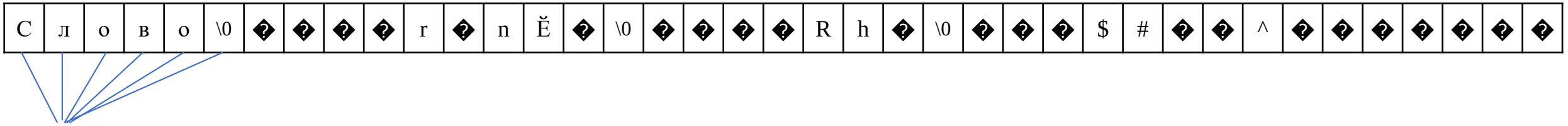

| С | л | о | в | о | \0 | ? | ? | ? | ? | r | ? | n | Ё | ? | \0 | ? | ? | ? | ? | R | h | ? | \0 | ? | ? | ? | $ | # | ? | ? | ^ | ? | ? | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

1. Вывод строкового литерала: “Введите слово:”
2. Выполнение программы остановлено.
3. Поток ввода ожидает введение строковых данных.
4. Поток ввода считает данные до ближайшего символа “ “ (пробел) или нажатия клавиши Enter.
5. Выполнение программы продолжается.
6. Вывод строкового литерала: “Ввод окончен”.

# Ввод строки с использованием scanf()

```
printf("Результат ввода: %s \n", cstr);
```

```
Результат ввода: Слово
```




1. Поток вывода, считает из памяти все символы от начала строки до ‘\0’.
2. Склеивает строковый литерал “Результат вывода:” и значения cstr.
3. Выводи информацию в терминал.

Знак ‘\0’ говорит о том, что полезная информация в строке закончилась.

# Разница между строкой Си-style и символом

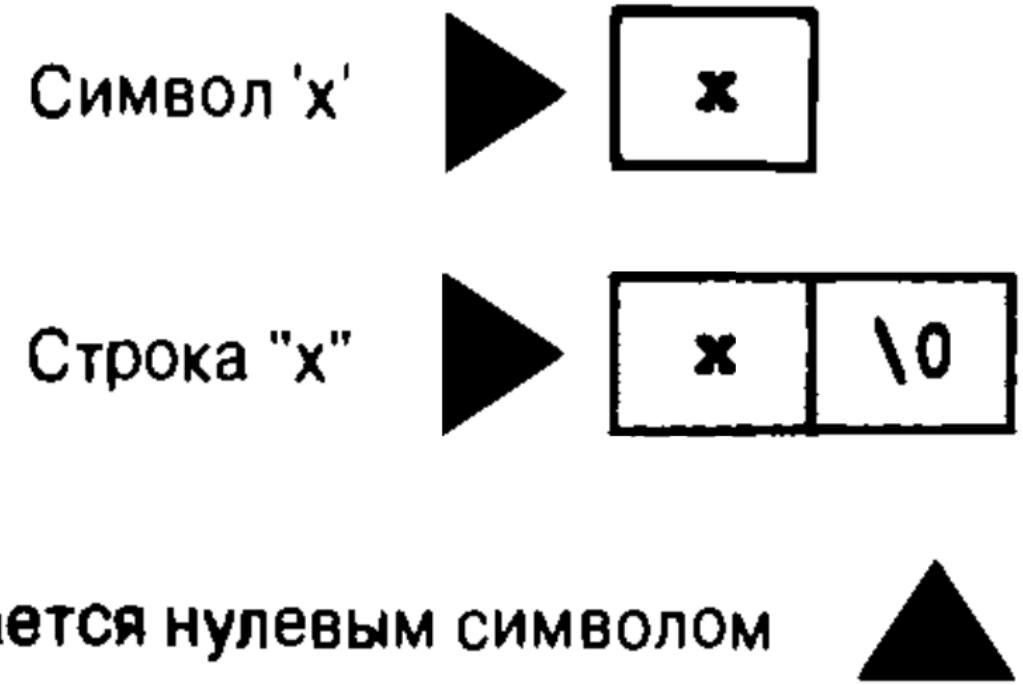



*Рис. 4.3. Символ 'x' и строка "x"*

# Узнать размер строки

Библиотека **string.h** содержит набор функции позволяющий работать со строками написанными в Си-style.

Для определения размера строки используется функция **strlen** *(от англ. string lenght, длинна строки)*, успешный результат выполнения возвращает целочисленное значение типа unsigned long.

```
printf("Строка занимает %zd символов и %zd ячеек
памяти.\n", strlen(cstr), sizeof cstr);
```

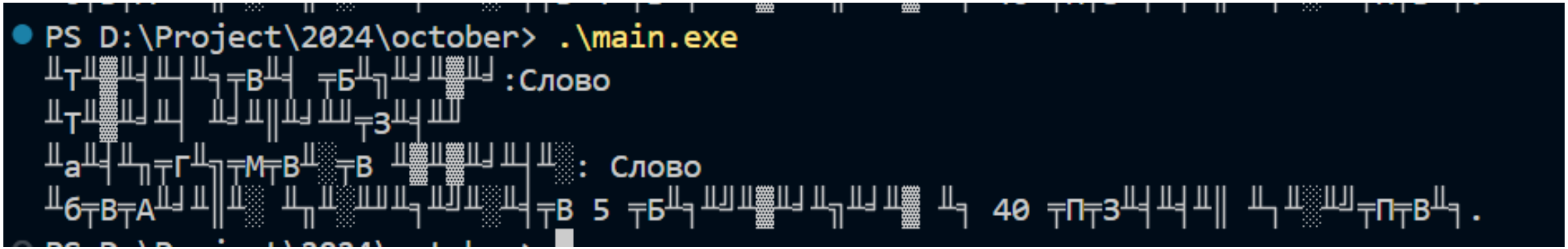
Какое число будет возвращено strlen(cstr)? А какое sizeof cstr ?

# Узнать размер строки - результаты

Компиляция на ОС GNU/Linux

```
usr:~/codec$ ./cstyle
Введите слово:Слово
Ввод окончен
Результат ввода: Слово
Строка занимает 10 символов и 40 ячеек памяти.
```

Компиляция на ОС Windows



# История и проблема кодировок (1)

1963/1967 - ASCII (7-битный стандарт)

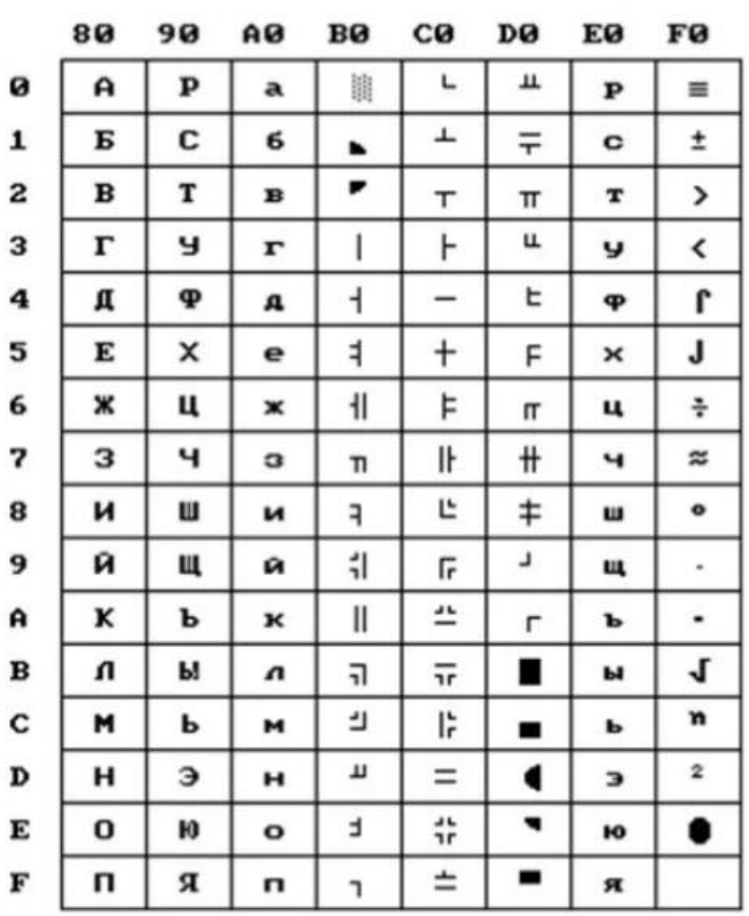
## ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [END OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

# История и проблема кодировок (2)

1970s–1980s - Расширенные 8-битные кодировки (code pages)

Windows (CMD) CP866 Linux Shell (KOI-8R) MAC Shell (MacCyrillic)



| | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|
| 0 | А | Р | а | ▒ | └ | ╨ | р | ≡ |
| 1 | Б | С | б | ▙ | ┴ | ╤ | с | ± |
| 2 | В | Т | в | ▛ | ┬ | ╥ | т | ≥ |
| 3 | Г | У | г | │ | ├ | ╙ | у | ≤ |
| 4 | Д | Ф | д | ┤ | ─ | ╘ | ф | ⌠ |
| 5 | Е | Х | е | ╡ | ┼ | ╒ | х | ⌡ |
| 6 | Ж | Ц | ж | ╢ | ╞ | ╓ | ц | ÷ |
| 7 | З | Ч | з | ╖ | ╟ | ╫ | ч | ≈ |
| 8 | И | Ш | и | ╕ | ╚ | ╪ | ш | ° |
| 9 | Й | Щ | й | ╣ | ╔ | ┘ | щ | ∙ |
| A | К | Ъ | к | ║ | ╩ | ┌ | ъ | · |
| B | Л | Ы | л | ╗ | ╦ | █ | ы | √ |
| C | М | Ь | м | ╝ | ╠ | ▄ | ь | ⁿ |
| D | Н | Э | н | ╜ | ═ | ▌ | э | ² |
| E | О | Ю | о | ╛ | ╬ | ▐ | ю | ■ |
| F | П | Я | п | ┐ | ╧ | ▀ | я | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | | 144 | ▒ | 160 | ─ | 176 | ╠ | 192 | ю | 208 | п | 224 | Ю | 240 | П |
| 129 | ¦ | 145 | ▒ | 161 | Ё | 177 | ╡ | 193 | а | 209 | я | 225 | А | 241 | Я |
| 130 | ┌ | 146 | ▓ | 162 | ┌ | 178 | ╢ | 194 | б | 210 | р | 226 | Б | 242 | Р |
| 131 | ┐ | 147 | ⌠ | 163 | ё | 179 | Ё | | | | | | | | |
| 132 | └ | 148 | ■ | 164 | ╒ | 180 | ╣ | | | | | | | | |
| 133 | ┘ | 149 | • | 165 | ╒ | 181 | ╣ | | | | | | | | |
| 134 | ├ | 150 | √ | 166 | ┐ | 182 | ╤ | | | | | | | | |
| 135 | ┤ | 151 | ≈ | 167 | ╖ | 183 | ╥ | | | | | | | | |
| 136 | ┬ | 152 | ≤ | 168 | ╕ | 184 | ╕ | | | | | | | | |
| 137 | ┴ | 153 | ≥ | 169 | └ | 185 | ┴ | | | | | | | | |
| 138 | ┼ | 154 | | 170 | ╚ | 186 | ╨ | | | | | | | | |
| 139 | ■ | 155 | ⌡ | 171 | ╚ | 187 | ╨ | | | | | | | | |
| 140 | ▄ | 156 | ° | 172 | ┘ | 188 | ╧ | | | | | | | | |
| 141 | ▐ | 157 | ² | 173 | ╜ | 189 | ╫ | | | | | | | | |
| 142 | ▌ | 158 | · | 174 | ╜ | 190 | ╫ | | | | | | | | |
| 143 | ▐ | 159 | ÷ | 175 | ╞ | 191 | ё | | | | | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П |
| 9 | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я |
| A | † | ° | ¢ | £ | § | • | ¶ | І | ® | © | ™ | Ђ | ђ | ≠ | Ѓ | ѓ |
| B | ∞ | ± | ≤ | ≥ | і | µ | ∂ | Ј | Є | є | Ї | ї | Љ | љ | Њ | њ |
| C | ј | Ѕ | ¬ | √ | ƒ | ≈ | Δ | « | » | … | | Ћ | ћ | Ќ | ќ | ѕ |
| D | – | — | “ | ” | ‘ | ’ | ÷ | „ | Ў | ў | Џ | џ | № | Ё | ё | я |
| E | а | б | в | г | д | е | ж | з | и | й | к | л | м | н | о | п |
| F | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | ¤ |

# История и проблема кодировок (3)

1987  Идея Unicode
1991 Основание Unicode Consortium
1992 Изобретение UTF-8 (Кен Томпсон, Роб Пайк)
1993 Первый стандарт Unicode (Unicode 1.0)
2000s–2010s Повсеместное внедрение UTF-8 (Linux, веб, мобильные ОС)

Он был задуман как совместимый с ASCII способ кодирования Unicode, позволяющий:

- представлять все символы Unicode (включая эмодзи, иероглифы и т.д.);
- при этом ASCII-символы (0–127) кодируются одним байтом, как в оригинальном ASCII;
- не требовать указания порядка байтов (в отличие от UTF-16/UTF-32);
- быть самосинхронизирующимся — можно найти начало символа, даже если начать читать с середины потока;

Тем самым переход от кодировок в 8 бит -> 16 бит

# История и проблема кодировок (4)

**UTF-8** — это переменная длина кодирования:
1 байт: 0xxxxxxx → ASCII (U+0000–U+007F)
2 байта: 110xxxxx 10xxxxxx → U+0080–U+07FF
3 байта: 1110xxxx 10xxxxxx 10xxxxxx → U+0800–U+FFFF
4 байта: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx → U+10000–U+10FFFF

Однако:
Си как язык программирования “не знает" про UTF-8 напрямую, он работает с байтами. (Стандарт C90 (ANSI C))

# История и проблема кодировок (5)

Но начиная с определённых стандартов, появились функции для работы с многобайтовыми и широкими символами, включая UTF-8.
Стандарт C95 (Amendment 1 к C90)

```c
#include <stdio.h>
#include <wchar.h>
#include <locale.h>
int main(void)
{
    setlocale(LC_ALL, "");
    char cstyle[] = "Пишу строку в C-style";
    wchar_t wcstyle[] = L"Пишу строку на расширенном C-style";
    printf("Пишу простой текст, без задней мысли\n");
    printf("%s\n", cstyle);
    wprintf(L"%ls", wcstyle);
    return 0;
}
```

# Константы с использованием препроцессора

**Константа** – именованное постоянное значение определенного типа данных, которое нельзя изменить в процессе выполнения команды, которое задаётся при помощи литерала.

Разница между **константой** и **литералом** присутствует, но ее сложно оценить.

**Литерал** – это значение, которое выражается как **само по себе**, а не значение **переменной**, тогда как **константа** – имеет имя, может быть использована в программе при вызове, определяется литералом.

# Константы с использованием препроцессора

**Шаблон:**

```
#define ИМЯ значение
```

**Примеры:**

```
#define RATE 0.5
#define APP_NAME "Application name"
```

**Константы на препроцессоре** принято описывать вне блока int main().

*Правилами хорошего тона* в программировании языка Си принято именовать константы с использованием только символов ВЕРХНЕГО регистра.

# Константы с модификатором const

**Шаблон:**

```
const тип ИМЯ = значение
```

**Примеры:**

```
const int RATE = 0.5;
const char[] APP_NAME = "Application name";
```

**Константы с модификатором** const принято описывать внутри блока int main().

*Правилами хорошего тона* в программировании языка Си принято именовать константы с использованием только символов ВЕРХНЕГО регистра.

# Примеры создания символьных строк

**Примеры:**

в блоке main()

```
char[17] app_name = "Application name";
char[] app_name = "Application name";

const char[17] APP_NAME = "Application name";
const char[] APP_NAME = "Application name";

printf(" Application name");
```

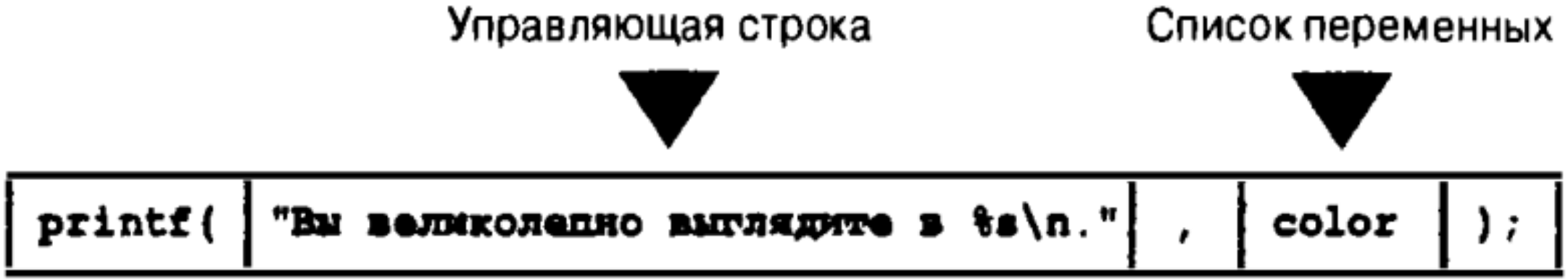
Вне блока main()

```
#define APP_NAME "Application name"
```

# printf() как функция вывода

**Формат использования функции `printf()` имеет вид:**

`printf(`*управляющая-строка, элемент1, элемент2, ...*`);`



# printf() как функция вывода

**Таблица 4.3. Спецификаторы преобразования и результирующий вывод**

| Спецификатор преобразования | Описание вывода |
|---|---|
| `%a` | Число с плавающей запятой, шестнадцатеричные цифры и p-запись (C99/C11) |
| `%A` | Число с плавающей запятой, шестнадцатеричные цифры и P-запись (C99/C11) |
| `%c` | Одиночный символ |
| `%d` | Десятичное целое число со знаком |
| `%e` | Число с плавающей запятой, экспоненциальное представление |
| `%E` | Число с плавающей запятой, экспоненциальное представление |
| `%f` | Число с плавающей запятой, десятичное представление |
| `%g` | В зависимости от значения использует `%f` или `%e`. Спецификатор `%e` применяется, если показатель степени меньше `-4` либо больше или равен указанной точности |

# printf() как функция вывода

| | |
|---|---|
| %G | В зависимости от значения использует %f или %E. Спецификатор %E применяется, если показатель степени меньше -4 либо больше или равен указанной точности |
| %i | Десятичное целое число со знаком (то же, что и %d) |
| %o | Восьмеричное целое число без знака |
| %p | Указатель |
| %s | Символьная строка |
| %u | Десятичное целое число без знака |
| %x | Шестнадцатеричное целое число без знака, используются шестнадцатеричные цифры 0f |
| %X | Шестнадцатеричное целое число без знака, используются шестнадцатеричные цифры 0F |
| %% | Знак процента |

# printf() как функция вывода

Функция **printf()** имеет возвращаемое целочисленное значение, которое хранит в себе **количество выведенных символов**.

```
int num;
num = printf("Кафедра вычислительных систем\n");
printf("%d\n", num);
return 0;
```

```
Кафедра вычислительных систем
57
```

Если в процессе вывода возникли ошибки, результатом возвращаемого значения будет **отрицательное число**.

# scanf() как функция ввода

Функция **scanf()** работает аналогично функции **printf()**.

**Формат использования функции** `scanf()` **имеет вид:**

```
scanf(управляющая-строка, элемент1, элемент2, ...);
```

Использование функции **scanf()** , для чтения переменной одного из базовых типов, **предварите** имя переменной символом &.
Использование функции **scanf()** для чтения строки в символьный массив, символ & не нужен.

# scanf() как функция ввода

| Спецификатор преобразования | Значение |
|---|---|
| %c | Интерпретирует введенные данные как символ |
| %d | Интерпретирует введенные данные как десятичное целое число со знаком |
| %e, %f, %g, %a | Интерпретирует введенные данные как число с плавающей запятой (%a появился в C99) |
| %E, %F, %G, %A | Интерпретирует введенные данные как число с плавающей запятой (%A появился в C99) |
| %i | Интерпретирует введенные данные как десятичное целое число со знаком |
| %o | Интерпретирует введенные данные как восьмеричное целое число со знаком |
| %p | Интерпретирует введенные данные как указатель (адрес) |
| %s | Интерпретирует введенные данные как строку. Ввод начинается с первого символа, не являющегося пробельным, и включает все символы до следующего пробельного символа |
| %u | Интерпретирует введенные данные как десятичное целое число без знака |
| %x, %X | Интерпретирует введенные данные как шестнадцатеричное целое число со знаком |

# scanf() как функция ввода

```
// input.c -- ситуации, когда должен использоваться символ &
#include <stdio.h>
int main(void)
{
    int age;          // переменная
    float assets;     // переменная
    char pet[30];     // строка

    printf("Введите информацию о своем возрасте, сумме в банке и любимом животном.\n");
    scanf("%d %f", &age, &assets);  // здесь должен быть указан символ &
    scanf("%s", pet);               // для строкового массива символ & не нужен
    printf("%d $%.2f %s\n", age, assets, pet);

    return 0;
}
```

```
Введите информацию о своем возрасте, сумме в банке и любимом животном.
38
92360.88 лама
38 $92360.88 лама
```

# scanf() как функция ввода

```
scanf("%d,%d", &n, &m);
```

```
88, 121
```

```
88,121
88  ,121
88 ,  121
```

и

```
88,
121
```

Правильное написание модификаторов и флагов, позволит добиться качественного и правильного ввода пользователем.
Но не обезопасит вашу программу от ошибок.

# scanf() как функция ввода

Функция **scanf()** – возвращает количество элементов, успешно считанных из терминала.

Если ни одного элемента не прочитано возвращается **0**.

Если же обнаружено условие “**конца файла**” (с англ. end-of-file), возвращает **EOF** и присвоится значение -1.

**EOF** – это особое условие, которое является индикацией того, что достигнут конец при чтении источника данных файл/сокет.

# Задачи на практику

Познакомиться с модификаторами и флагами для функции printf().
Познакомиться с модификаторами и флагами для функции scanf()
Познакомиться с модификатором * в функциях printf() и scanf()

Смотреть вопросы для самоконтроля (153 стр, page 94)
Упражнения по программированию (155-156 стр, page 97)

# Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)