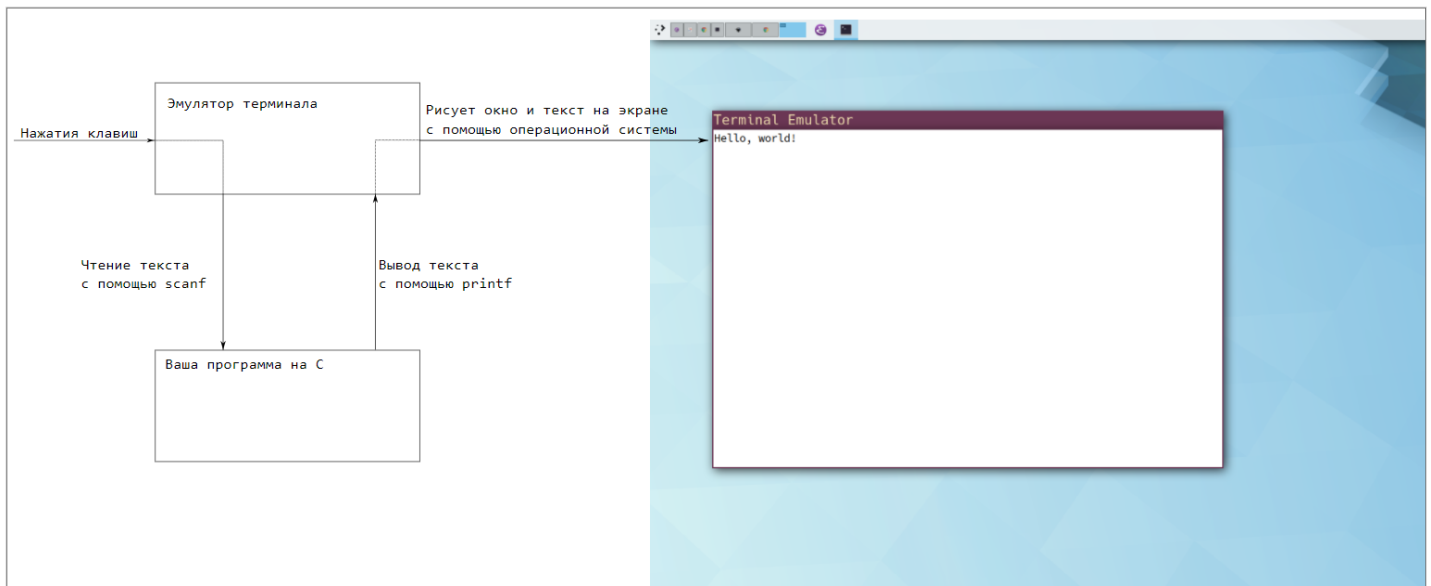


# Ввод и вывод данных из программ

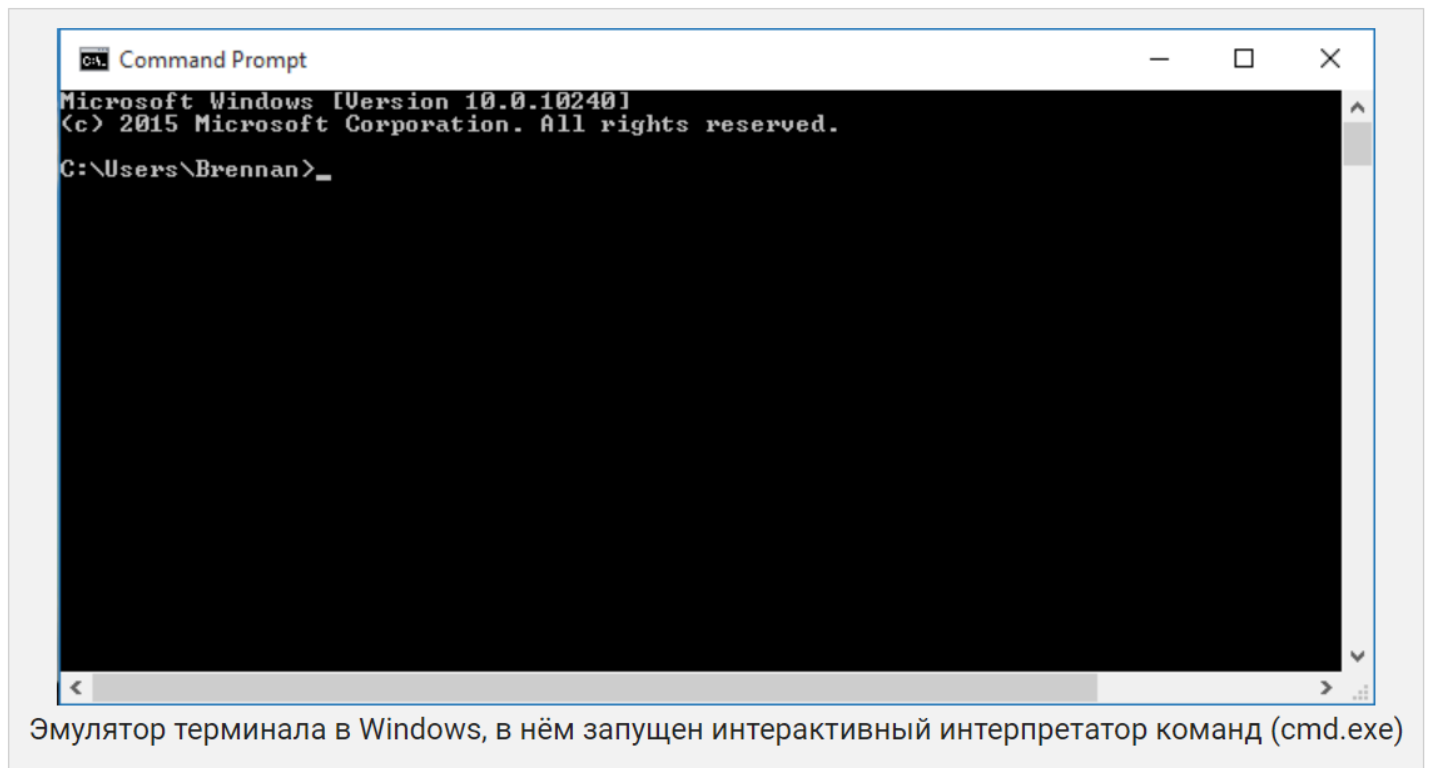
Программы могут по-разному взаимодействовать с пользователем и вообще с "внешним миром". В этом уроке нас интересуют основы ввода и вывода в С, очень просто и в контексте выполнения программ на ваших персональных компьютерах. Этот урок даже не про сам язык – он про то, как начать понимать среду, в которой программа выполняется. Мы опишем ситуацию с точки зрения пользователя, почти как [чёрный ящик](#).

Те простые программы на С, которые мы пишем, являются [консольными приложениями](#), то есть взаимодействуют с пользователем через текст. В них нет графического интерфейса: окон, кнопок, красиво нарисованных полей для ввода текста и т.д. Такие программы запускаются внутри [эмулятора терминала](#), графического приложения, которое умеет считывать нажатия клавиш и рисовать текст поверх одноцветного окна. Запустить программу внутри него значит, что эмулятор терминала будет обрабатывать нажатия клавиш от пользователя и пересылать их в программу, а также собирать весь вывод программы и рисовать его на экране.

На этом скриншоте вы видите один из эмуляторов терминала в Linux, внутри него запущена программа, которая выводит `Hello, world!`



В составе ОС Windows есть специальное консольное приложение `cmd.exe`. Вот как оно выглядит во встроенном эмуляторе терминала в Windows:



Приложение `cmd.exe` является интерпретатором команд: вы можете интерактивно с ним взаимодействовать посылая ему текстовые команды.

При запуске `cmd.exe` выводит:

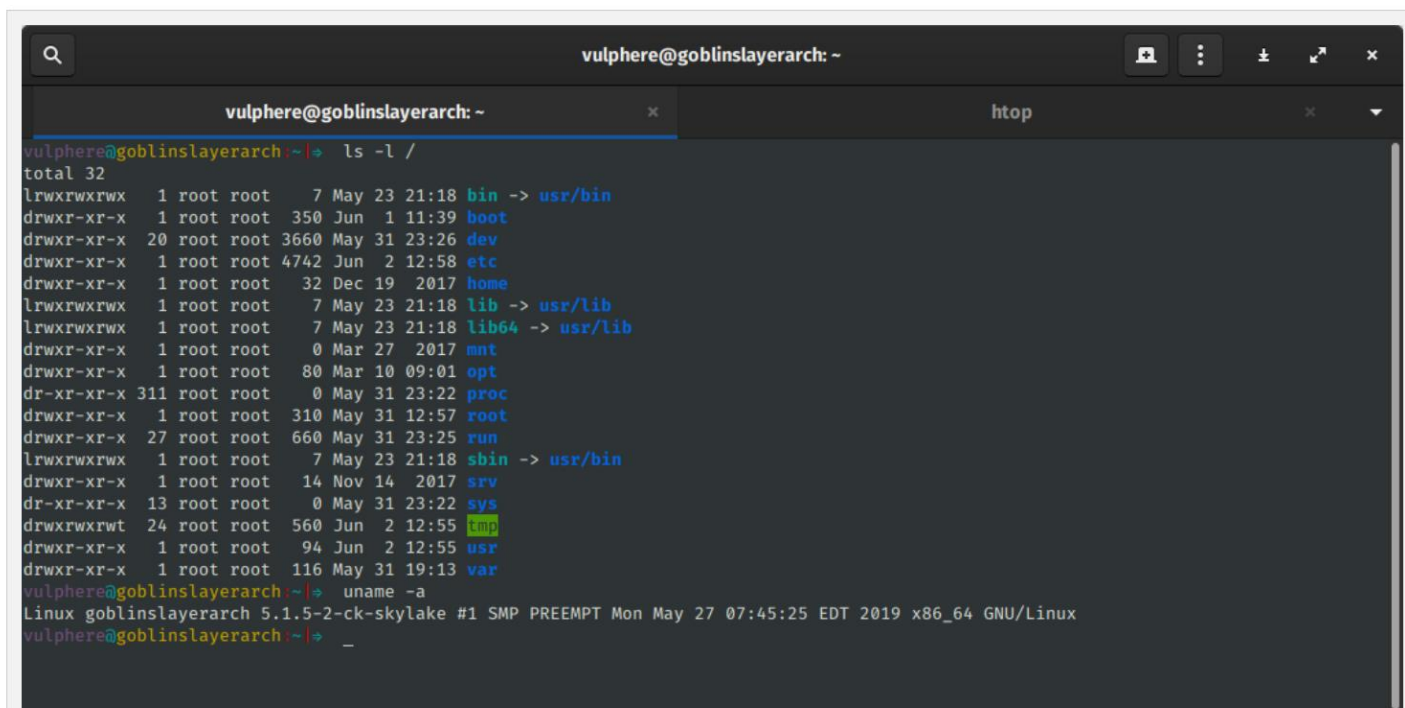
- приглашение (Microsoft Windows (Version ... ));
- директорию, где мы находимся в данный момент (C:\Users\Brennan);

Интерпретатор команд на схеме выше был бы там же, где и "Ваша программа на C". Если бы мы сами писали `cmd.exe`, то могли бы вывести приглашение и текущую директорию через вызов функции `printf`. Эмулятор терминала принимает текст от `cmd.exe` и рисует этот текст на экране.

Затем `cmd.exe` ожидает, что пользователь введёт какие-то текстовые команды. Эмулятор терминала позволяет вводить эти команды с помощью клавиатуры и передавать в `cmd.exe` введённый текст. Мы бы читали текст с помощью `scanf`.

На каждую команду `cmd.exe` среагирует соответствующим образом; команды могут перемещать вас по файловой системе, запускать программы, и многое другое. Вы можете запустить и вашу собственную программу из `cmd.exe`, перейдя в директорию с её исполняемым файлом и введя её имя, или же запустить её напрямую через проводник.

В Linux и macOS эмулятор терминала и интерпретатор команд часто используются в похожей связке. Отметим, что интерпретаторов команд там больше: если в Windows это, как правило, всего лишь `cmd.exe` и [PowerShell](#), то в Linux и macOS это могут быть `sh`, `bash`, `zsh`, `ksh`, `fish` и многие другие.



```
vulphere@goblinslayerarch: ~  
vulphere@goblinslayerarch:~$ ls -l /  
total 32  
lrwxrwxrwx 1 root root 7 May 23 21:18 bin -> usr/bin  
drwxr-xr-x 1 root root 350 Jun 1 11:39 boot  
drwxr-xr-x 20 root root 3660 May 31 23:26 dev  
drwxr-xr-x 1 root root 4742 Jun 2 12:58 etc  
drwxr-xr-x 1 root root 32 Dec 19 2017 home  
lrwxrwxrwx 1 root root 7 May 23 21:18 lib -> usr/lib  
lrwxrwxrwx 1 root root 7 May 23 21:18 lib64 -> usr/lib  
drwxr-xr-x 1 root root 0 Mar 27 2017 mnt  
drwxr-xr-x 1 root root 80 Mar 10 09:01 opt  
dr-xr-xr-x 311 root root 0 May 31 23:22 proc  
drwxr-xr-x 1 root root 310 May 31 12:57 root  
drwxr-xr-x 27 root root 660 May 31 23:25 run  
lrwxrwxrwx 1 root root 7 May 23 21:18 sbin -> usr/bin  
drwxr-xr-x 1 root root 14 Nov 14 2017 srv  
dr-xr-xr-x 13 root root 0 May 31 23:22 sys  
drwxrwxrwt 24 root root 560 Jun 2 12:55 tmp  
drwxr-xr-x 1 root root 94 Jun 2 12:55 usr  
drwxr-xr-x 1 root root 116 May 31 19:13 var  
vulphere@goblinslayerarch:~$ uname -a  
Linux goblinslayerarch 5.1.5-2-ck-skylake #1 SMP PREEMPT Mon May 27 07:45:25 EDT 2019 x86_64 GNU/Linux  
vulphere@goblinslayerarch:~$
```

Эмулятор терминала в Linux (Gnome), в нём запущен интерактивный интерпретатор команд (чаще всего это bash)

Ещё раз, тезисно:

- В эмуляторе терминала запускаются консольные приложения; эмулятор рисует на экране окно и текст в нём, а также позволяет его считывать.
- Интерпретаторы команд могут запускаться внутри терминалов; они считывают команды через терминал и могут выводить текст в ответ на них. Но они не рисуют текст на экране.
- Другие консольные приложения не отличаются от интерпретаторов команд: они также могут запускаться в эмуляторе терминала и через него рисовать на экране текст и запрашивать у пользователя ввести текст с клавиатуры.
- Можно запустить в эмуляторе терминала интерпретатор команд, такой, как `cmd` или `bash`, а затем выполнить в нём команду запуска приложения. Такое приложение тоже будет выводить текст через эмулятор терминала.

Вы можете компилировать и запускать программы прямо в браузере используя бесплатный сервис [OnlineGDB](#). В нём вы вводите программу в текстовое поле, запускаете её и видите, какой текст она хочет выводить на экран с помощью `printf`. Программа также может запрашивать текст у пользователя; в таком случае необходимо ввести его в поле внизу.

[Запустите следующий пример в OnlineGDB](#); вам потребуется нажать на кнопку Run и ввести число. Затем программа напечатает строку `The number was:` и введённое вами число.

```
// Эта функция считывает одно число со входа
```

```
// Пока нам не нужно знать, как она устроена
```

```
int read_int() {
```

```
int x;

scanf("%d", &x);

return x;

}
```

```
int main()

{

    printf("Enter a number: \n");

    printf("The number was: %d\n", read_int() );

    return 0;

}
```

[Запустить пример в OnlineGDB](#)

Здесь есть функция `read_int`, которую пока нам не нужно понимать; воспринимайте её как "чёрный ящик", способ попросить пользователя ввести число с клавиатуры.

В других средах разработки обычно есть возможность запустить программу и взаимодействовать с ней через встроенный эмулятор терминала.



The screenshot shows the JetBrains Clion IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, and Help. The main editor window displays a C program in `main.c` with the following code:

```
1  #include <stdio.h>
2  // Эта функция считывает одно число со входа
3  // Пока нам не нужно знать, как она устроена
4  int read_int() {
5      int x;
6      scanf( format: "%d", &x);
7      return x;
8  }
9
10
11 int main()
12 {
13     printf( format: "Enter a number: \n");
14     printf( format: "The number was: %d\n", read_int() );
15
16     return 0;
17 }
```

Below the editor, the 'Run' window shows the execution of the program. The output is as follows:

```
/home/sayon/CLionProjects/untitled1/cmake-build-debug/untitled1
Enter a number:
42
The number was: 42

Process finished with exit code 0
```

Среда разработки: JetBrains Clion. Так выглядит результат работы программы с вводом "42" во встроенном в Clion эмуляторе терминала.

## Вывод и отрисовка текста

Как мы увидели на примере интерпретатора команд и эмулятора терминала, *вывод данных* и *отрисовка текста на экране* это не одно и то же:

1. Программа, запущенная внутри эмулятора терминала, иницирует вывод текста.
2. Эмулятор терминала этот текст рисует на экране.

Получается, что можно выводить текст не только в терминал? Это верное предположение.

У каждой работающей в Windows или Linux программы есть три специальных *стандартных потока данных* ([IO streams](#)):

- Поток 0 это поток ввода (`stdin`)
- Поток 1 это поток вывода (`stdout`)
- Поток 2 это поток ошибок (`stderr`)

При запуске программы внутри терминала эти потоки автоматически подсоединяются следующим образом:

- `stdin` подсоединён к вводу с клавиатуры;

- `stdout` подсоединён к выводу в терминал;
- `stderr` тоже подсоединён к выводу в терминал.

При вводе данных в программу мы получаем их, как правило, через `stdin`; вывод с помощью `printf` происходит в `stdout`.

## Перенаправление потоков

Можно перенаправить эти потоки, например, подсоединив к ним файлы. Вот несколько примеров, которые работают в командной строке Linux (когда вы запускаете интерпретатор вроде `bash` в эмуляторе терминала):

```
# Ввод в программу из файла input.txt, вывод по-прежнему на экран
```

```
./myprogram < input.txt
```

```
# Ввод в программу с клавиатуры, вывод в файл output.txt
```

```
./myprogram > output.txt
```

```
# Ввод из файла, вывод в файл
```

```
./myprogram < input.txt > output.txt
```

```
# Соединить вывод myprogram и ввод программы wc
```

```
./myprogram | wc
```

```
# wc это стандартная утилита для подсчёта количества строчек, слов и т.д.
```

Что выводить в поток ошибок? Простое правило такое: результаты работы программы выводятся в `stdout`, а информация о том, как они получены, выводится в `stderr`. Это включает в себя ошибки (не получилось послать сообщение по сети, кончилась память), диагностические сообщения (выполнение какой-то функции заняло слишком много времени, необходимо обновить библиотеку, к серверу подсоединился новый пользователь и т.д.).

Важно разделять эту информацию на два потока: это позволяет перенаправить `stdout` и `stderr` в разные файлы.

```
# Ввод в программу с клавиатуры, вывод в файл output.txt, поток ошибок выводится в терминал
```

```
./myprogram > output.txt
```

```
# Ввод в программу с клавиатуры, вывод в файл output.txt, поток ошибок выводится в errors.txt
```

```
./myprogram > output.txt 2>errors.txt
```

```
# Ввод в программу с клавиатуры, вывод в файл output.txt, поток ошибок сливается с потоком вывода
```

```
./myprogram 2>&1 > output.txt
```

В потоки можно вводить/выводить любые данные, не только текст.