# Get Next Line

Чтение строки из `fd` слишком утомительно

*Резюме: Цель этого проекта - заставить вас кодировать функцию, которая возвращает строку, заканчивающуюся новой строкой, читаемую из файлового дескриптора.*

# Вступление

Теперь вы начинаете понимать, что будет сложно читать данные из файлового дескриптора, если вы заранее не знаете его размер. Какого размера должен быть ваш буфер? Сколько раз вам нужно читать дескриптор файла, чтобы получить данные?

Совершенно нормально и естественно, что вы, как программист, захотите прочитать `строку`, которая заканчивается разрывом строки из файлового дескриптора. Например, каждая команда, которую вы вводите в своей оболочке, или каждая строка, читаемая из плоского файла.

Благодаря проекту `get_next_line` вы, наконец, сможете написать функцию, которая позволит вам читать строку, заканчивающуюся символом новой строки из файлового дескриптора. Вы сможете добавить эту функцию в свой `libftif`, как вы считаете нравится и, что самое главное, используйте его во всех будущих проектах, которые потребуют этого.

# Глава I.

## Цели

Этот проект не только позволит вам добавить очень удобную функцию в вашу коллекцию, но также позволит вам изучить очень интересную новую концепцию программирования на C: `статические переменные`.

Вы также получите более глубокое понимание распределения, происходят ли они в памяти стека или в памяти кучи, манипулирования и жизненного цикла буфера, неожиданной сложности, связанной с использованием одной или нескольких статических переменных.

Ваше уважение к нормам и правилам написания кода повысит точность вашего программирования. Мы также подозреваем, что ваш подход к кодированию изменится, когда вы обнаружите, что начальное состояние переменной в функции может варьироваться в зависимости от вызова этой самой функции.

# Глава II.

## Общие инструкции

- Ваш проект должен быть написан в соответствии с нормами написания кода. Если у вас есть бонусные файлы/функции, то они тоже будут включены в проверку норм, и вы получите `0`, если внутри есть ошибка нормы.
- Ваши функции не должны завершаться неожиданно (`segmentation fault`, `bus error`, `double free` и т.д.), За исключением неопределенного поведения. Если это произойдет, ваш проект будет считаться нефункциональным и во время оценки получит `0`.
- Все пространство памяти, выделенное кучей, должно быть должным образом освобождено при необходимости. Утечки памяти недопустимы.
- Если субъект потребует этого, то вы должны представить `Makefile`, который скомпилирует ваши исходные файлы в требуемый вывод с флагами `-Wall`, `-Wextra` и `-Werror`. Ваш `Makefile` не должен использовать повторное связывание.
- Ваш `Makefile` должен как минимум содержать правила: `$(NAME)`, `all`, `clean`, `fclean` и `re`.
- Чтобы получить бонусы в свой проект, вы должны включить правило `bonus` в свой `Makefile`, который добавит все различные заголовки, библиотеки или функции, запрещенные в основной части проекта. Бонусы должны быть в другом файле `file_bonus.{c/h}`. Оценка обязательной и бонусной части проводится отдельно.
- Если ваш проект позволяет вам использовать вашу библиотеку `libft`, вы должны скопировать ее исходный код и связанный с ним файл Makefile в папку `libft` вместе с соответствующим файлом `Makefile`. `Makefile` вашего проекта должен скомпилировать библиотеку, используя свой `Makefile`, а затем скомпилировать проект.
- Мы рекомендуем вам создавать программы тестирования для вашего проекта, даже если эту работу не нужно будет отправлять и оценивать. Это даст вам возможность легко проверить свою работу и работу коллег. Вы найдете эти тесты особенно полезными во время защиты. Действительно, во время защиты вы можете использовать свои тесты и/или тесты партнера, которого вы оцениваете.
- Отправьте свою работу в назначенный репозиторий `git`. Оцениваться будет только работа в репозитории `git`. Если `Deepthought` назначен для оценки вашей работы, это будет сделано после ваших оценок коллег. Если во время выставления оценок `Deepthought's` в каком-либо разделе вашей работы произойдет ошибка, оценка будет остановлена.

# Глава III.

## Обязательная часть

| **Имя программмы** | **get_next_line** |
|---|---|
| Прототип | int get_next_line(int fd, char **line); |
| Файлы для сдачи | get_next_line.c, get_next_line_utils.c, get_next_line.h |
| Параметры | #1. файловый дескриптор для чтения; #2. Значение, куда это будет прочитано. |
| Возвращаемое значение | (1) Если строка была прочитана, (0) Если был достигнут EOF, (-1) Если случилась ошибка. |
| Внешние функции. | read, malloc, free |
| Описание | Напишите функцию, которая возвращает строку, прочитанную из файлового дескриптора, без символа новой строки. |

- Вызов вашей функции `get_next_line` в цикле позволит вам читать текст, доступный в файловом дескрипторе, по одной строке за раз, пока не будет достигнут конец файла `EOF`(End of File).
- `Строкой` мы называем последовательность символов, которые заканчиваются на символе `\n` (код `ascii 0x0a`) или на конце файла `EOF`(End of File).
- Убедитесь, что ваша функция хорошо себя ведет при чтении из файла и при чтении из стандартного ввода.
- Ваша собственная библиотека `libft` не разрешена для этого проекта. Вы должны добавить файл `get_next_line_utils.c`, который будет содержать функции, необходимые для работы `get_next_line`.
- Ваша программа должна компилироваться с флагом `-D BUFFER_SIZE=xx.`, который будет использоваться в качестве размера буфера для вызовов чтения в вашей `get_next_line`. Это значение может и будет изменено вашими оценщиками или программой `moulinnette`.
- Компиляция будет производиться следующим образом: `gcc -Wall -Wextra -Werror -D BUFFER_SIZE=32 get_next_line.c get_next_line_utils.c`.
- Ваш `read` должен использовать `BUFFER_SIZE`, определенный во время компиляции, для чтения из файла или из чтения из стандартного потока ввода `stdin`.

- В заголовочном файле `get_next_line.h` у вас должен быть как минимум прототип функции `get_next_line` и макрос `BUFF_SIZE`, позволяющий выбрать размер буфера чтения для функции чтения. Это значение будет изменено во время защиты, чтобы оценить силу вашей функции.
- Мы считаем, что `get_next_line` имеет неопределенное поведение, если между двумя вызовами один и тот же файловый дескриптор переключается на другой файл до того, как на первом `fd` будет достигнут `EOF`.
- `lseek` является недопустимой функцией. Чтение файла необходимо производить только один раз.
- Наконец, мы считаем, что `get_next_line` имеет неопределенное поведение при чтении из двоичного файла. Однако при желании вы можете сделать такое поведение последовательным.
- Глобальные переменные запрещены.

---

> Ваша функция все еще работает, если значение `BUFFER_SIZE` равно `9999`? А если значение `BUFFER_SIZE` равно `1`? А если значение `10000000`? Ты знаешь почему?

---

---

> Вы должны стараться читать как можно меньше каждый раз при вызове `get_next_line`. Если вы встретили новую строку, вам нужно вернуть текущую строку. Не читайте файл целиком - обрабатывайте каждую строку.

---

---

> Не сдавайте свой проект без тестирования. Есть много тестов, которые нужно запустить, охватите свои базы. Попробуйте прочитать из файла, из перенаправления, из стандартного ввода. Как ваша программа ведет себя, когда вы отправляете новую строку в стандартный вывод? А `CTRL-D`?

---

---

> Эта статья может оказаться полезной для тех, кто умеет читать:
> https://latedev.wordpress.com/2012/12/04/all-about-eof/

---

---

> Хорошим началом было бы узнать, что такое статическая переменная
> https://en.wikipedia.org/wiki/Static_variable5

---

# Глава IV.

## Бонусная часть

Проект `get_next_line` прост и оставляет очень мало места для бонусов, но я уверен, что у вас много воображения. Если вы отлично справились с обязательной частью, непременно выполните эту бонусную часть, чтобы идти дальше. Повторяю, бонус не будет учитываться, если обязательная часть не идеальна. Сдавайте все 3 исходных файла с `_bonus` для этой части.

- Преуспейте в `get_next_line` с единственной статической переменной.
- Создайте возможность управлять несколькими файловыми дескрипторами с помощью вашей `get_next_line`. Например, если файловые дескрипторы `3`, `4` и `5` доступны для чтения, вы можете вызвать `get_next_line` один раз на `3`, один раз на `4`, еще раз на `3`, затем еще раз на `5` и т. Д. Без потери потока чтения на каждом из дескрипторов.