Libft

Ваша первая собственная библиотека

Резюме: Цель этого проекта - написать библиотеку на C, перегруппировывающую обычные функции, которые вам будет разрешено использовать во всех ваших других проектах.

Глава I.

Предисловие

Программирование на `C` может быть очень утомительным, если у человека нет доступа к этим очень полезным стандартным функциям. Этот проект дает вам возможность переписать эти функции, понять их и научиться их использовать. Эта библиотека поможет вам во всех ваших будущих проектах на языке `C`.

В рамках этого проекта мы также даем вам возможность расширить список функций своими собственными. Найдите время, чтобы расширить свою библиотеку `libft` в течение года.

Глава II.

Основные 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 превратить бонусы в свой проект, вы должны включить бонус правила 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	**libft.a**
Файлы для сдачи	*.c, libft.h, Makefile
Makefile	Да
Внешние функции.	Подробно ниже
Авторизованый Libft	Не применим
Описание	Напишите свою собственную библиотеку, содержащую отрывки из важных функций для вашего кура.

III.1 Технические предупреждения

- Ваш файл `libft.h` может содержать макросы и определения типов, если это необходимо.
- Строка ВСЕГДА должна заканчиваться на `'\0'`, даже если она не включена в описание функции, конечно если явно не указано другое.
- Запрещено использовать глобальные переменные.
- Если вам нужны подфункции для написания сложной функции, вы должны определить эти подфункции как статические, чтобы не публиковать их вместе с вашей библиотекой. Было бы неплохо сделать это и в ваших будущих проектах.

> Перейдите по этой ссылке, чтобы узнать больше о статических функциях: https://codingfreak.blogspot.com/2010/06/static-functions-in-c.html

- Отправьте все файлы в корень вашего репозитория.
- Вы должны использовать команду `ar` для создания вашей библиотеки, использование команды `libtool` запрещено.

- Вы должны обращать внимание на свои типы и разумно использовать приведение типов, когда это необходимо, особенно когда задействован тип `void *`. Вообще говоря, избегайте неявных приведений.

Пример:

```
char    *str;

str = malloc(42 * sizeof(*str));            /* Неверно ! Malloc возвращает
void* (неявное приведение) */
str = (char *) malloc(42 * sizeof(*str));   /* Верно ! (явное приведение
типов) */
```

III.2 Часть 1 - Функции `libc`

В этой первой части вы должны перекодировать набор функций `libc`, как определено в их `man`. Ваши функции должны будут представлять тот же прототип и поведение, что и исходные. Имена ваших функций должны иметь префикс `ft_`. Например, `strlen` становится `ft_strlen`.

> Некоторые прототипы функций, которые необходимо перекодировать, используют квалификатор `restrict`. Это ключевое слово является частью стандарта `c99`. Поэтому запрещено включать его в ваши прототипы и компилировать с флагом `-std = c99`.

Вы должны перекодировать следующие функции. Этим функциям не нужны внешние функции:

- memset
- strrchr
- bzero
- strnstr
- memcpy
- strncmp
- memccpy
- atoi
- memmove
- isalpha
- memchr
- isdigit
- memcmp
- isalnum
- strlen
- isascii
- strlcpy
- isprint
- strlcat
- toupper
- strchr
- tolower

Вы также должны перекодировать следующие функции, используя функцию `malloc`:

- calloc
- strdup

III.2 Часть 2 - Дополнительные функции

Во второй части вы должны закодировать набор функций, которые либо не включены в `libc`, либо включены в другую форму. Некоторые из этих функций могут быть полезны для написания функций Части 1.

Имя функции	**ft_substr**
Прототип	char *ft_substr(char const *s, unsigned int start, size_t len);
Файлы для сдачи	-
Параметры	#1. Строка, из которой создается подстрока. #2. Начальный индекс подстроки в строке ’s’. #3. Максимальная длина подстроки.
Возвращаемое значение	Подстрока. `NULL`, если выделение не удалось.
Внешние функции.	malloc
Описание	Выделяет(с помощью `malloc`) и возвращает подстроку из строки `s`. Подстрока начинается с индекса `start` и имеет максимальный размер `len`.

Имя функции	**ft_strjoin**
Прототип	char *ft_strjoin(char const *s1, char const *s2);
Файлы для сдачи	-
Параметры	#1.Строка префикса. #2. Строка суффикса.
Возвращаемое значение	Новая строка. `NULL`, если выделение не удалось.
Внешние функции.	malloc
Описание	Выделяет (с помощью `malloc`) и возвращает новую строку, которая является результатом объединения `s1` и `s2`.

Имя функции	**ft_strtrim**
Прототип	char *ft_strtrim(char const *s1, char const *set);
Файлы для сдачи	-
Параметры	#1.Обрезаемая строка. #2.Контрольный набор символов для обрезки.
Возвращаемое значение	Новая строка. `NULL`, если выделение не удалось.
Внешние функции.	malloc
Описание	Выделяет (с помощью `malloc`) и возвращает копию `s1` с символами, указанными в `set`, удаленными из начала и конца строки.

Имя функции	**ft_split**
Прототип	char **ft_split(char const *s, char c);
Файлы для сдачи	-
Параметры	#1. Строка, которую нужно разделить. #2. Символ-разделитель.
Возвращаемое значение	Массив новых строк, полученных в результате разделения. NULL, если выделение не удалось.
Внешние функции.	malloc, free
Описание	Выделяет (с помощью `malloc`) и возвращает массив строк, полученных путем разделения `s` с использованием символа `c` в качестве разделителя. Массив должен заканчиваться указателем NULL.

Имя функции	**ft_itoa**
Прототип	char *ft_itoa(int n);
Файлы для сдачи	-
Параметры	#1. Целое число для преобразования.
Возвращаемое значение	Строка, представляющая целое число. NULL, если размещение не удается.
Внешние функции.	malloc
Описание	Выделяет (с помощью `malloc`) и возвращает строку, представляющую целое число, полученное в качестве аргумента. Отрицательные числа нужно обрабатывать.

Имя функции	**ft_strmapi**
Прототип	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
Файлы для сдачи	-
Параметры	#1. Строка, по которой выполняется итерация. #2. Функция, применяемая к каждому
Возвращаемое значение	Строка, созданная последовательным применением 'f'. Возвращает NULL, если выделение не удалось.
Внешние функции.	malloc
Описание	Применяет функцию `f` к каждому символу строки `s` для создания новой строки (с помощью `malloc`) в результате последовательных применений `f`.

Имя функции	**ft_putchar_fd**
Прототип	void ft_putchar_fd(char c, int fd);
Файлы для сдачи	-
Параметры	#1. Символ для вывода. #2. Дескриптор файла для записи в поток.
Возвращаемое значение	-
Внешние функции.	write
Описание	Выводит символ `c` в указанный файловым дескриптором поток.

Имя функции	**ft_putstr_fd**
Прототип	void ft_putstr_fd(char *s, int fd);
Файлы для сдачи	-
Параметры	#1. Строка для вывода. #2. Дескриптор файла для записи в поток.
Возвращаемое значение	-
Внешние функции.	write
Описание	Выводит строку `s` в указанный файловым дескриптором поток.

Имя функции	**ft_putendl_fd**
Прототип	void ft_putendl_fd(char *s, int fd);
Файлы для сдачи	-
Параметры	#1. Строка для вывода. #2. Дескриптор файла для записи в поток.
Возвращаемое значение	-
Внешние функции.	write
Описание	Выводит строку `s` в указанный файловым дескриптором поток, за которой следует новая строка.

Имя функции	**ft_putnbr_fd**
Прототип	void ft_putnbr_fd(int n, int fd);
Файлы для сдачи	-
Параметры	#1. Число для вывода. #2. Дескриптор файла для записи в поток.
Возвращаемое значение	-
Внешние функции.	write
Описание	Выводит целое число n в указанный файловым дескриптором поток.

Глава IV.

Бонусная часть

Если вы успешно выполнили обязательную часть, вам будет интересно продолжить ее. Вы можете рассматривать этот последний раздел как бонусные баллы.

Наличие функций для управления памятью и строками очень полезно, но вскоре вы обнаружите, что функции для управления списками еще более полезны.

Вы будете использовать следующую структуру для представления элементов вашего списка. Эта структура должна быть добавлена в ваш файл `libft.h`.

Команда `make bonus` добавит бонусные функции в библиотеку `libft.a`.

Вам не нужно добавлять `_bonus` к файлам `.c` и заголовку в этой части. Добавляйте `_bonus` только к файлам, содержащим ваши собственные бонусные функции.

```
typedef struct  s_list
{
    void            *content;
    struct s_list   *next;
}               t_list;
```

Вот описание полей структуры `t_list`:

- `content`: содержит адрес данных. `void *` позволяет хранить адреса любых данных.
- `next`: содержит адрес следующего элемента списка связаных между собой структур или `NULL`, если это последний элемент.

Следующие функции позволят вам легко использовать ваши списки.

Имя функции	**ft_lstnew**
Прототип	t_list *ft_lstnew(void *content);
Файлы для сдачи	-
Параметры	#1. Контент для создания нового элемента.
Возвращаемое значение	Новый элемент.
Внешние функции.	malloc
Описание	Выделяет (с помощью malloc) и возвращает новый элемент. Переменная `content` инициализируется значением параметра `content`. Переменная `next` инициализируется значением `NULL`.

Имя функции	**ft_lstadd_front**
Прототип	void ft_lstadd_front(t_list **lst, t_list *new);
Файлы для сдачи	-
Параметры	#1. Адрес указателя на первую ссылку списка. #2. Адрес указателя на элемент, который нужно добавить в список.
Возвращаемое значение	-
Внешние функции.	-
Описание	Добавляет элемент `new` в начало списка.

Имя функции	**ft_lstsize**
Прототип	int ft_lstsize(t_list *lst);
Файлы для сдачи	-
Параметры	#1. Начало списка.
Возвращаемое значение	Длина списка.
Внешние функции.	-
Описание	Подсчитывает количество элементов в списке.

Имя функции	**ft_lstlast**
Прототип	t_list *ft_lstlast(t_list *lst);
Файлы для сдачи	-
Параметры	#1. Начало списка.
Возвращаемое значение	Последний элемент списка.
Внешние функции.	-
Описание	Возвращает последний элемент списка.

Имя функции	**ft_lstadd_back**
Прототип	void ft_lstadd_back(t_list **lst, t_list *new);
Файлы для сдачи	-
Параметры	#1. Адрес указателя на первую ссылку списка. #2. Адрес указателя на элемент, который нужно добавить в список.
Возвращаемое значение	-
Внешние функции.	-
Описание	Добавляет элемент `new` в конец списка.

Имя функции	**ft_lstdelone**
Прототип	void ft_lstdelone(t_list *lst, void (*del)(void*));
Файлы для сдачи	-
Параметры	#1. Элемент, который нужно освободить. #2. Адрес функции, используемой для удаления содержимого.
Возвращаемое значение	-
Внешние функции.	free
Описание	Принимает в качестве параметра элемент и освобождает память содержимого элемента с помощью функции `del`, заданной в качестве параметра, и освобождает элемент. Память следующего элемента списка не должна освобождаться.

Имя функции	**ft_lstclear**
Прототип	void ft_lstclear(t_list **lst, void (*del)(void*));
Файлы для сдачи	-
Параметры	#1. Адрес указателя на элемент. #2. Адрес функции, используемой для удаления содержимого элемента.
Возвращаемое значение	-
Внешние функции.	free
Описание	Удаляет и освобождает память данного элемента и всех его последователей, используя функцию `del` и `free`. Наконец, указатель на список должен быть установлен в `NULL`.

Имя функции	**ft_lstiter**
Прототип	void ft_lstiter(t_list *lst, void (*f)(void *));
Файлы для сдачи	-
Параметры	#1. Адрес указателя на элемент. #2. Адрес функции, используемой для перебора списка.
Возвращаемое значение	-
Внешние функции.	-
Описание	Итерирует список `lst` и применяет функцию `f` к содержимому каждого элемента.

Имя функции	**ft_lstmap**
Прототип	t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));
Файлы для сдачи	-
Параметры	#1. Адрес указателя на элемент. #2. Адрес функции, используемой для перебора списка. #3. Адрес функции, используемой для удаления содержимого элемента при необходимости.
Возвращаемое значение	Новый список. `NULL`, если выделение не удалось, .
Внешние функции.	malloc, free
Описание	Итерирует список `lst` и применяет функцию `f` к содержимому каждого элемента. Создает новый список в результате последовательных применений функции `f`. Функция `del` используется для удаления содержимого элемента, если это необходимо.

Если вы успешно выполнили как обязательный, так и бонусный разделы этого проекта, мы рекомендуем вам добавить другие функции, которые, по вашему мнению, могут быть полезны для расширения вашей библиотеки. Например, версия `ft_strsplit`, которая возвращает список вместо массива, функция `ft_lstfold`, аналогичная функции `reduce` в `Python` и функции `List.fold_left` в `OCaml` (остерегайтесь утечки памяти!). Вы можете добавлять функции для управления массивами, стеками, файлами, картами, хэш-таблицами и т.д.

«Предел - ваше воображение.»