**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

**Кафедра программного обеспечения вычислительной техники и автоматизированных систем**

**Курсовая работа**

**Разработка и реализация библиотеки строчных функций на языке программирования C**

Выполнил:

Студент группы: ВТ – 231

Масленников Д. А.

Принял:

Лукьянов А.М.

Белгород

2024

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение..................................................................................................................... | 3 |
| Постановка задачи..................................................................................................... | 3 |
| Описание решения задачи......................................................................................... | 4 |
| Ожидаемый результат............................................................................................... | 4 |
| Техническая информация.......................................................................................... | 4 |
| Основная часть........................................................................................................... | 5 |
| Вывод……………………………………………………………………………….. | 16 |

**Введение**

Целью данной курсовой работы является разработка собственной библиотеки строковых функций, которая обеспечит надежный и эффективный инструментарий для работы с текстовыми данными. Основными задачами исследования являются:

* Анализ существующих реализаций строковых функций в стандартной библиотеке Си
* Разработка собственных функций для обработки и трансформации строк
* Обеспечение высокой производительности и безопасности реализованных функций
* Тестирование и верификация созданной библиотеки

Актуальность работы обусловлена постоянно растущими требованиями к качеству и эффективности программного обеспечения. Создание оптимизированной библиотеки строковых функций позволит разработчикам получить надежный инструмент для манипуляции строками с минимальными накладными расходами.

Методологической основой работы послужили принципы низкоуровневого программирования на языке Си, методы оптимизации алгоритмов обработки данных и подходы к обеспечению безопасности при работе с памятью.

**ПОСТАНОВКА ЗАДАЧИ**

## Задачей курсовой работы будет разработка и реализация библиотеки функций для работы со строками на языке СИ, тестирование написанных функций. Функции должны быть протестированы автоматически, чтобы исключить какие-либо недоработки. Вот функции которые я хочу реализовать:

* Функции работы с памятью
* Реализация поиска байта в памяти (*find\_byte*)
* Разработка функции сравнения памяти (*compare\_memory*)
* Создание функции заполнения памяти (*fill\_memory*)
* Реализация копирования памяти (*copy\_memory*)
* Функции работы со строками
* Конкатенация строк с ограничением (*concat\_strings\_limited*)
* Определение длины строки (*get\_string\_length*)
* Поиск первого вхождения символа (*find\_first\_char*)
* Копирование строк с ограничением (*copy\_string\_limited*)
* Сравнение строк с ограничением (*compare\_strings\_limited*)
* Расширенные строковые операции
* Поиск подстроки (*find\_substring*)
* Разделение строк по разделителю (*split\_string*)
* Конвертация регистра (*convert\_to\_uppercase, convert\_to\_lowercase*)
* Вставка подстроки (*insert\_string*)
* Очистка краев строки (*trim\_string\_edges*)
* Функции подсчета и поиска
* Подсчет совпадающих символов (*count\_matching\_chars*)
* Подсчет несовпадающих символов (*count\_not\_matching\_chars*)
* Поиск первого совпадающего символа (*find\_first\_matching\_char*)
* Поиск последнего вхождения символа (*find\_last\_char*)

**ОПИСАНИЕ РЕШЕНИЯ ЗАДАЧИ**

Для реализации библиотеки строчных функций будут использованы два файла: заголовочный фал (*string\_lib.h*) и файлл реализации (*string\_lib.c*). Заголовочный файл содержит прототипы всех функций библиотеки, а файл реализации содержит их реализацию. Так же будут написаны тесты при помощи библиотек *check.h*, которые проверят правильность работы функций. Тесты будут храниться в отдельном файле *tests.c*.

## Ожидаемые результаты

* Надежная библиотека строковых функций
* Высокая производительность
* Минимальное использование ресурсов
* Расширяемость и возможность интеграции в различные проекты

**ТЕХНИЧЕСКАЯ ИНФОРМАЦИЯ**

Библиотека строчных функций будет реализована на языке программирования С. В процессе создания будут использоваться стандартные библиотеки языка C, тесты будут реализованы при помощи библиотеки *check.h*.

**ОСНОВНАЯ ЧАСТЬ**

**Импорт библиотек и создание заголовочного файла**

В самом начале нужно импортировать необходимые нам библиотеки. После этого проходит проверка системы, в зависимости от чего задается размер беззнакового целого *size\_*t

#include <ctype.h>

#include <locale.h>

#include <math.h>

#include <stdarg.h>

#include <stdbool.h>

#include <stdint.h>

#include <stdlib.h>

#include <wchar.h>

#if defined(\_WIN64) || defined(\_\_x86\_64\_\_) || defined(\_\_ppc64\_\_)

typedef uint64\_t size\_t; // 64-битная система

#else

typedef uint32\_t size\_t; // 32-битная система

#endif

**Константы и параметры окна**

Задается переменная *ASCII\_SHIFT* и присваивается значение 32, чтобы использовать ее для операции сдвига

**Функции**

Далее задаем тип, название, и переменные, которые будет принимать функция. На этом заканчивается работа с заголовочным файлом.

// Находит первое вхождение символа в блоке памяти указанной длины

extern void \*find\_byte(const void \*str, int symbol, size\_t n);

// Сравнивает два блока памяти указанной длины

extern int compare\_memory(const void \*str1, const void \*str2, size\_t n);

// Заполняет блок памяти указанным символом на заданную длину

extern void \*fill\_memory(void \*str, int symbol, size\_t n);

// Копирует блок памяти из одного места в другое

extern void \*copy\_memory(void \*dest, const void \*src, size\_t n);

// Конкатенирует (объединяет) строки с ограничением максимальной длины

extern char \*concat\_strings\_limited(char \*dest, const char \*src, size\_t n);

// Вычисляет длину строки

extern size\_t get\_string\_length(const char \*str);

// Находит первое вхождение указанного символа в строке

extern char \*find\_first\_char(const char \*str, int symbol);

// Копирует строку с ограничением максимальной длины

extern char \*copy\_string\_limited(char \*dest, const char \*src, size\_t n);

// Подсчитывает количество несовпадающих символов между двумя строками

extern size\_t count\_not\_matching\_chars(const char \*str1, const char \*str2);

// Сравнивает строки с ограничением максимальной длины

extern int compare\_strings\_limited(const char \*str1, const char \*str2, size\_t n);

// Находит первый совпадающий символ между двумя строками

extern char \*find\_first\_matching\_char(const char \*str1, const char \*str2);

// Находит последнее вхождение символа в строке

extern char \*find\_last\_char(const char \*str, int symbol);

// Разделяет строку на части по указанному разделителю

extern char \*split\_string(char \*str, const char \*delim);

// Преобразует строку в верхний регистр

extern void \*convert\_to\_uppercase(const char \*str);

// Преобразует строку в нижний регистр

extern void \*convert\_to\_lowercase(const char \*str);

// Вставляет одну строку в другую по указанному индексу

extern void \*insert\_string(const char \*src, const char \*str, size\_t start\_index);

// Удаляет символы по краям строки, указанные в trim\_chars

extern void \*trim\_string\_edges(const char \*src, const char \*trim\_chars);

**Реализация Функций**

Функция *find\_byte* предназначена для поиска первого вхождения заданного символа в определенной области памяти. Эта функция полезна, когда необходимо найти конкретный байт в блоке данных, например, в строке или массиве символов.

extern void \*find\_byte(const void \*str, int symbol, size\_t n) {

unsigned char \*ptr = (unsigned char \*)str;

void \*out = NULL;

size\_t i = 0;

while (i < n && out == NULL) {

if (\*ptr == (unsigned char)symbol) {

out = ptr;

} else {

++ptr;

++i;

}

}

return out;

}

Функция *compare\_memory* предназначена для сравнения двух блоков памяти одинакового размера. Она сравнивает каждый байт в этих областях и возвращает разницу между первым несовпадающим байтом, если таковой имеется, или 0, если блоки идентичны.

extern int compare\_memory(const void \*str1, const void \*str2, size\_t n) {

const unsigned char \*s1 = (const unsigned char \*)str1;

const unsigned char \*s2 = (const unsigned char \*)str2;

int difference = 0;

size\_t i = 0;

while (i < n && difference == 0) {

difference = s1[i] - s2[i];

++i;

}

return difference;

}

Функция *fill\_memory* предназначена для заполнения определенного участка памяти заданным символом. Она часто используется для инициализации массива или строки конкретными значениями, такими как нули или пробелы.

extern void \*fill\_memory(void \*str, int symbol, size\_t n) {

unsigned char \*s = (unsigned char \*)str;

for (size\_t i = 0; i < n; ++i) {

s[i] = (unsigned char)symbol;

}

return str;

}

Функция *copy\_memory* предназначена для копирования данных из одной области памяти в другую. Особенностью этой функции является возможность безопасного копирования, даже если области перекрываются.

extern void \*copy\_memory(void \*dest, const void \*src, size\_t n) {

unsigned char \*d = (unsigned char \*)dest;

const unsigned char \*s = (const unsigned char \*)src;

if (d < s || d >= s + n) {

for (size\_t i = 0; i < n; i++) d[i] = s[i];

} else {

for (size\_t i = n; i > 0; i--) d[i - 1] = s[i - 1];

}

return dest;

}

Функция *concat\_strings\_limited* предназначена для объединения двух строк, при этом ограничивается максимальная длина результирующей строки. Она добавляет содержимое строки-источника (*src*) к концу строки-приемника (*dest*), но не превышает указанный предел длины.

extern char \*concat\_strings\_limited(char \*dest, const char \*src, size\_t n) {

size\_t dest\_len = get\_string\_length(dest);

size\_t i = 0;

for (; i < n && src[i]; ++i) {

dest[dest\_len + i] = src[i];

}

dest[dest\_len + i] = '\0';

return dest;

}

Функция *get\_string\_length* предназначена для определения длины строки, заканчивающейся нулевым символом ('\0'). Она проходит по всем символам строки до тех пор, пока не встретит этот завершающий символ, и затем возвращает количество символов без учета самого '\0'.

extern size\_t get\_string\_length(const char \*str) {

const char \*end = str;

while (\*end != '\0') end++;

return end - str;

}

Функция *find\_first\_char* предназначена для поиска первого вхождения заданного символа в строке. Она просматривает строку посимвольно, пока не найдет нужный символ или не дойдет до конца строки.

extern char \*find\_first\_char(const char \*str, int symbol) {

while (\*str != (char)symbol) {

if (\*str == '\0') return NULL;

str++;

}

return (char \*)str;

}

Функция *copy\_string\_limited* предназначена для копирования строки из источника (*src*) в буфер назначения (*dest*), ограничивая количество копируемых символов значением n. Она гарантирует, что строка будет корректно завершена нулевым символом, даже если исходная строка длиннее указанного лимита

extern char \*copy\_string\_limited(char \*dest, const char \*src, size\_t n) {

fill\_memory(dest, '\0', n);

for (size\_t i = 0; i < n && src[i]; i++) {

dest[i] = src[i];

}

return dest;

}

Функция *count\_not\_matching\_chars* предназначена для подсчета количества символов в первой строке, которые отсутствуют во второй строке. Она проходит по первой строке и для каждого её символа проверяет наличие этого символа во второй строке. Если символ не найден, счётчик увеличивается.

extern size\_t count\_not\_matching\_chars(const char \*str1, const char \*str2) {

size\_t counter = 0;

for (const char \*a = str1; \*a; a++) {

for (const char \*b = str2; \*b; b++) {

if (\*a == \*b) return counter;

}

counter++;

}

return counter;

}

Функция *compare\_strings\_limited* предназначена для сравнения двух строк ограниченной длины. Она сравнивает первые n символов двух строк и возвращает разницу между первым несовпадающим символом или 0, если строки совпадают в пределах заданного ограничения.

extern int compare\_strings\_limited(const char \*str1, const char \*str2, size\_t n) {

for (size\_t i = 0; i < n; i++) {

if (str1[i] != str2[i] || str1[i] == '\0' || str2[i] == '\0') {

return (unsigned char)str1[i] - (unsigned char)str2[i];

}

}

return 0;

}

Функция *find\_first\_matching\_char* предназначена для поиска первого символа в строке *str1*, который также присутствует в строке *str2*. Она проходит по всем символам строки *str1* и ищет совпадение с любым символом строки *str2*.

extern char \*find\_first\_matching\_char(const char \*str1, const char \*str2) {

char \*res = NULL;

for (const char \*a = str1; \*a; ++a) {

for (const char \*b = str2; \*b; ++b) {

if (\*a == \*b) {

res = (char \*)a;

return res;

}

}

if (res) {

return res;

}

}

return res;

}

Функция *find\_last\_char* предназначена для поиска последнего вхождения заданного символа в строке. Она начинает поиск с конца строки и движется к началу, останавливаясь, как только находит соответствующий символ.

extern char \*find\_last\_char(const char \*str, int symbol) {

char \*res = NULL;

size\_t len = get\_string\_length(str);

for (int i = len; i >= 0; --i) {

if (str[i] == symbol) {

res = ((char \*)str) + i;

return res;

}

}

return res;

}

Функция *split\_string* разбивает строку на подстроки по указанному разделителю. Она возвращает указатель на начало очередной подстроки, отделённой от остальной части строки путём замены следующего разделителя на нулевой символ. Таким образом, каждая следующая вызов функции продолжает обработку строки с точки остановки предыдущего вызова.

extern char \*split\_string(char \*str, const char \*delim) {

static char \*last;

int c;

if (str == NULL) str = last;

do {

if ((c = \*str++) == '\0') return NULL;

} while (find\_first\_char(delim, c));

--str;

last = str + count\_not\_matching\_chars(str, delim);

if (\*last != '\0') \*last++ = '\0';

return str;

}

Функция *convert\_to\_uppercase* преобразует строку в верхний регистр. Она создаёт новую строку, копирует туда символы исходной строки, преобразуя строчные буквы латинского алфавита в заглавные.

extern void \*convert\_to\_uppercase(const char \*str) {

if (str == NULL) return NULL;

char \*new\_str = (char \*)calloc(get\_string\_length(str) + 1, sizeof(char));

if (new\_str == NULL) return NULL;

for (size\_t i = 0; i < get\_string\_length(str); i++) {

if (str[i] < 'a' || str[i] > 'z') {

new\_str[i] = str[i];

} else {

new\_str[i] = str[i] - ASCII\_SHIFT;

}

}

return (void \*)new\_str;

}

Функция *convert\_to\_lowercase* преобразует строку в нижний регистр. Она создаёт новую строку, копируя туда символы исходной строки и преобразуя прописные буквы латинского алфавита в строчные.

extern void \*convert\_to\_lowercase(const char \*str) {

if (str == NULL) return NULL;

char \*new\_str = (char \*)calloc(get\_string\_length(str) + 1, sizeof(char));

if (new\_str == NULL) return NULL;

for (size\_t i = 0; i < get\_string\_length(str); i++) {

if (str[i] < 'A' || str[i] > 'Z') {

new\_str[i] = str[i];

} else {

new\_str[i] = str[i] + ASCII\_SHIFT;

}

}

return (void \*)new\_str;

}

Функция *insert\_string* вставляет одну строку в другую в указанную позицию. Она создает новую строку, состоящую из исходной строки, в которую вставлена вторая строка, начиная с заданного индекса.

extern void \*insert\_string(const char \*src, const char \*str, size\_t start\_index) {

if (src == NULL || str == NULL) return NULL;

size\_t src\_len = get\_string\_length(src);

size\_t str\_len = get\_string\_length(str);

if (start\_index > src\_len) return NULL;

size\_t new\_len = src\_len + str\_len + 1;

char \*new\_str = calloc(new\_len, sizeof(char));

if (new\_str == NULL) return NULL;

for (size\_t i = 0; i < new\_len; i++) {

if (i < start\_index) {

new\_str[i] = src[i];

} else if (i < str\_len + start\_index) {

new\_str[i] = str[i - start\_index];

} else {

new\_str[i] = src[i - str\_len];

}

}

return (void \*)new\_str;

}

Функция *trim\_string\_edges* удаляет все ведущие и замыкающие символы из строки, указанные в списке символов для обрезки. Это делает строку "обрезанной" по краям, убирая лишние пробелы, табуляции и другие нежелательные символы.

extern void \*trim\_string\_edges(const char \*src, const char \*trim\_chars) {

if (src == NULL || trim\_chars == NULL) return NULL;

char \*trimmed\_str = NULL;

size\_t start = 0;

size\_t end = get\_string\_length(src) - 1;

while (src[start] && find\_first\_char(trim\_chars, src[start]) != NULL) {

start++;

}

while (end > start && find\_first\_char(trim\_chars, src[end])) {

end--;

}

if (start <= end) {

size\_t trimmed\_len = end - start + 2;

trimmed\_str = (char \*)calloc(trimmed\_len, sizeof(char));

if (trimmed\_str == NULL) return NULL;

copy\_string\_limited(trimmed\_str, src + start, end - start + 1);

} else {

trimmed\_str = (char \*)calloc(1, sizeof(char));

if (trimmed\_str == NULL) return NULL;

}

return trimmed\_str;

}

**Тестирование**

Тестирование программного обеспечения является неотъемлемой частью разработки любой программы, особенно когда речь идет о библиотеках, таких как библиотека строковых функций на языке Си. Использование инструментов для автоматизированного тестирования, таких как ***Check***, и анализа покрытия кода, как ***gcov***, позволяют существенно улучшить качество кода и снизить количество ошибок.

Тестирование библиотеки строковых функций

В данном разделе мы рассмотрим, как проводилось тестирование нашей библиотеки строковых функций с использованием инструмента ***Check*** и анализатора покрытия кода ***gcov***.

**Основные этапы тестирования:**

**Создание тестового набора**: Для каждой функции, реализованной в библиотеке, были созданы отдельные тесты. Например, для функции *get\_string\_length*, которая аналогична стандартной функции *strlen*, были разработаны следующие тесты:

START\_TEST(test\_strlen\_1) { // Тест 1: Проверка длины обычной строки

const char \*str1 = "Hello, World!";

ck\_assert\_int\_eq(get\_string\_length(str1), 13);

}

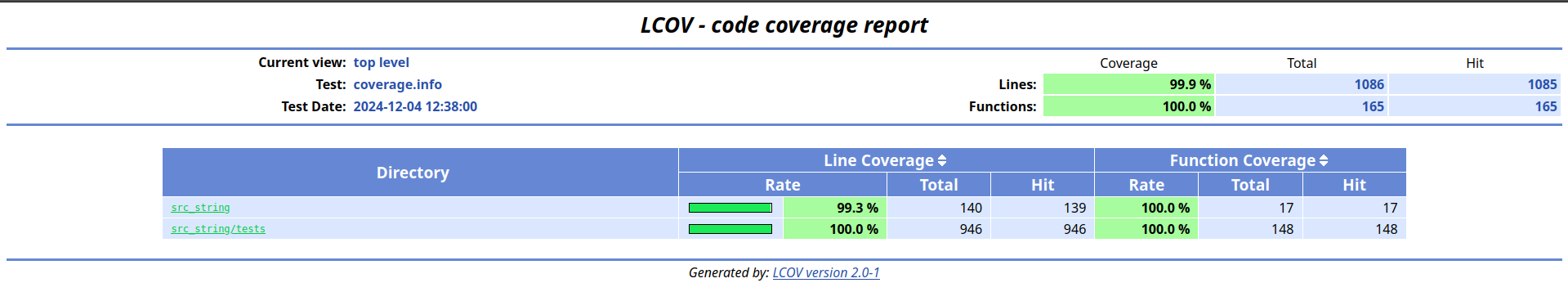
END\_TEST

**Запуск тестов**: После создания всех необходимых тестов их можно запустить с помощью команды *make test*. Это позволит убедиться, что все функции работают корректно и возвращают ожидаемые результаты.

**Анализ покрытия кода**: Для оценки того, насколько полно тестовый набор покрывает исходный код, использовался инструмент ***gcov***. Он генерирует отчеты, показывающие, какие части кода были выполнены во время тестирования, а какие остались без внимания. Команда для запуска анализа покрытия выглядит следующим образом:

make *gcov\_report*

**Просмотр отчета**: Отчет об анализе покрытия создается в виде HTML-файлов, которые можно открыть в браузере. Эти файлы содержат подробную информацию о том, сколько раз каждая строка кода была выполнена, и какие строки остались без выполнения.



**Преимущества использования Check и gcov:**

**Простота использования**: Инструменты *Check* и *gcov* имеют простые и интуитивно понятные интерфейсы, что делает их удобными для начинающих разработчиков.

**Автоматизация**: Возможность автоматической генерации отчетов и логов упрощает процесс тестирования и снижает вероятность человеческих ошибок.

**Модульность**: *Check* позволяет проверять каждую функцию отдельно, что помогает быстрее находить и устранять ошибки.

**Анализ покрытия**: Использование *gcov* дает полное представление о том, насколько хорошо тестируется код, позволяя выявить слабые места и улучшить качество тестирования.

**Поддержка различных платформ**: Оба инструмента поддерживают различные операционные системы и компиляторы, что делает их универсальными для тестирования.

Таким образом, применение инструментов ***Check*** и ***gcov*** для тестирования библиотеки строковых функций на языке Си обеспечивает высокий уровень надежности и качества кода, а также облегчает процесс его дальнейшей разработки и сопровождения.

**Вывод**

Разработка библиотеки строковых функций на языке Си является важным этапом в освоении основ программирования и создании надежных программных решений. В ходе выполнения этой курсовой работы была создана библиотека, включающая ряд ключевых операций над строками, таких как определение длины строки, копирование строк, сравнение строк, конкатенация и поиск подстроки. Каждая функция была разработана с учетом требований к эффективности и безопасности, что позволило получить высококачественный продукт.

Особое внимание было уделено процессу тестирования, который играл ключевую роль в обеспечении надежности и стабильности библиотеки. Применение инструмента ***Check*** позволило автоматизировать процесс тестирования и создать обширный набор тестов, охватывающих различные сценарии использования функций. Это дало уверенность в том, что каждая функция работает корректно даже в сложных и нестандартных ситуациях.

Кроме того, использование инструмента ***gcov*** для анализа покрытия кода помогло выявить участки кода, которые оставались непроверенными в процессе тестирования. Благодаря этому удалось дополнительно расширить тестовые наборы и добиться практически полного покрытия кода, что существенно повысило качество итогового продукта.

Результаты проведенного тестирования и анализа покрытия показали, что созданная библиотека обладает высокой степенью надежности и может быть использована в реальных проектах. Таким образом, данная курсовая работа достигла своей цели — разработать надежную и эффективную библиотеку строковых функций, пригодную для применения в различных приложениях.

Полученные навыки и опыт, приобретенные в ходе выполнения этого проекта, будут полезны в дальнейшем обучении и профессиональной деятельности. Работа с инструментами тестирования и анализа кода, такими как *Check* и *gcov*, способствует формированию правильного подхода к разработке программного обеспечения, ориентированного на качество и надежность.