Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 5**

**Дисциплина:** Низкоуровневое программирование

**Тема:** Программирование на языке C

Выполнил студент гр. 3530901/90003 Андрианов В. С.

(подпись)

Преподаватель Алексюк А.О.

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Содержание**

1. Техническое задание………………………………………………………………3
2. Метод решения……...……………………………………………………...............3
3. Листинг программы…...…………………………………………………...............4

3.1 main.c…...……………………………………………………………………….4

3.2 bloom.c…...…………………………………………………………………….4

3.3 bloom.h …...……………………………………………………………………5

3.4 bloomStruct.h …...………………………………………………………………5

3.5 hash.c …...………………………………………………………………………5

3.6 hash.h …...………………………………………………………………………6

3.7 cmdFile.c …...………………………………………………………………….6

3.8 cmdFile.h …...…………………………………………………………………8

3.9 errors.c …...……………………………………………………………………9

3.10 errors.h …...…………………………………………………………………10

3.11 test.c …...……………………………………………………………………10

3.12 test.h …...……………………………………………………………………11

1. Коды ошибок……………………………………………………………………12
2. Программа и методика испытаний ……………………………………………13

5.1 С помощью тестов……………………………………………………………13

5.2 Консольный режим работы…………………………………………………13

5.3 Выполнение программы без ошибок………………………………………18

6. Ссылка на Github…………………………………………………………………19

# **1. Техническое задание**

Фильтр Блума с заданным размером битового поля и количеством элементов. Начальные параметры: размер битового поля, количество элементов и начальные элементы (которые необходимо добавить) берутся из файла. Работа по добавлению и проверки элементов ведется с помощью командной строки. Ключ -a – добавляет элемент, -c – проверяет элемент, -h – ключ помощи, -e – ключ выхода.

Пример командной строки: -a\_l – добавляет элемент l. -c\_l – проверяет элемент l.

# **2. Метод решения**

Алгоритм решения:

1. Считываем из входного файла данные. В первой строке расположен размер битового поля. Во второй строке – количество элементов. В третьей – начальные элементы. Считываем данные и создаем структуру bloom, задавая размер битового поля, количество элементов, количество хэш функций k = (где n – длина поля, m – кол-во элементов), seed для хэш функции количество добавленных элементов и битовое поле.
2. При добавлении элемента происходит расчет k хэшей. После получения k хэшей, каждое значение хэшей делится на размер битового поля, и по этим значениям расставляются единицы в битовом поле.
3. При проверке элемента происходит расчет k хэшей. После получения k хэшей, каждое значение хэшей делится на размер битового поля, и по этим значениям идет проверка значения из битового поля, если хотя бы одно из k значений равно нулю, тогда этого элемента нет.

# **3. Листинг программы**

3.1 main.c

#include <malloc.h>  
#include "Bloom/bloom.h"  
#include "CommandLineAndFile/cmdFile.h"  
#include "Errors/errors.h"  
  
int main(int argc, char \*args[]) {  
 if (argc != 2) errors(1, args[1]);  
 bloom \*b = readFile(args[1]);  
 char \*cmdLine = (char \*)malloc(4);  
 if (cmdLine == NULL) errors(11, "cmdLine");  
 int cmd = 1;  
 while (cmd == 1) {  
 cmd = commandLine(cmdLine, b, args[1]);  
 cmdLine = (char \*)malloc(4);  
 if (cmdLine == NULL) errors(11, "cmdLine");  
 }  
 free(cmdLine);  
 delete(b);  
 return 0;  
}

3.2 bloom.c

#include <malloc.h>  
#include <stdlib.h>  
#include "bloom.h"  
#include "math.h"  
#include "../Hash/hash.h"  
#include "../Errors/errors.h"  
  
void add(const void \*element, bloom \*b) {  
 uint64\_t \*hashed = hash((const uint8\_t\*)element, b->hashNum, b);  
 for (int i = 0; i < b->hashNum; ++i) b->bitField[hashed[i] % b->size] = 1;  
 b->numOfAddedElem++;  
 free(hashed);  
}  
  
int contains(const void \*element, bloom \*b) {  
 uint64\_t \*hashed = hash((const uint8\_t\*)element, b->hashNum, b);  
 for (int i = 0; i < b->hashNum; ++i) {  
 if (b->bitField[hashed[i] % b->size] != 1) {  
 free(hashed);  
 return 0;  
 }  
 }  
 free(hashed);  
 return 1;  
}  
  
bloom \*create(int length, int elementsNum) {  
 bloom \*b = (bloom \*)calloc(1, sizeof(bloom));  
 if (b == NULL) errors(11, "структуру фильтра Блума: b");  
 b->size = length;  
 b->numOfElements = elementsNum;  
 b->hashNum = (int)(log(2) \* length / elementsNum);  
 b->seed = rand() \* 32 + 32;  
 b->numOfAddedElem = 0;  
 b->bitField = (int \*)calloc(length, sizeof(int));  
 if (b->bitField == NULL) errors(11, "битовое поле структуры фильтра Блума: b (b->bitField)");  
 return b;  
}  
  
void delete(bloom \*b) {  
 if (b->bitField != NULL) free(b->bitField);  
 free(b);  
}

3.3 bloom.h

#ifndef BLOOM\_FILTER\_BLOOM\_H  
#define BLOOM\_FILTER\_BLOOM\_H  
  
#include "bloomStruct.h"  
  
void add(const void \*element, bloom \*b);  
int contains(const void \*element, bloom \* b);  
bloom \*create(int length, int elementsNum);  
void delete(bloom \*b);  
  
#endif

3.4 bloomStruct.h

#ifndef BLOOM\_FILTER\_BLOOMSTRUCT\_H  
#define BLOOM\_FILTER\_BLOOMSTRUCT\_H  
  
#include <stdint.h>  
  
struct BloomStruct {  
 int size;  
 int numOfElements;  
 int hashNum;  
 uint64\_t seed;  
 int numOfAddedElem;  
 int \*bitField;  
};  
  
typedef struct BloomStruct bloom;  
  
#endif

3.5 hash.c

#include <malloc.h>  
#include "hash.h"  
#include "../Errors/errors.h"  
  
uint64\_t \*hash(const uint8\_t \*element, int k, bloom \*b) {  
 // k - количество хэш функций (из формулы: k = ln2 \* n/m (где n – длина поля, m – кол-во элементов))  
 uint64\_t \*hash = (uint64\_t\*)calloc(k, sizeof(uint64\_t));  
 if (hash == NULL) errors(11, "hash");  
 uint64\_t result = 1;  
 int index = 0;  
 const int halfElem = \*element / 2;  
 int halfK = k / 2;  
 int start = 1;  
 for (int i = 0; i < \*element; ++i) {  
 // Создание начального значения хэша и приведение его к 64битному числу  
 result = ((result \* b->seed + i) \* k) & 0xFFFFFFFF;  
 // для первой половины element последующий расчет различных k / 2 хэшей идет следующим образом:  
 if (i == halfElem) {  
 for (int j = start; j < halfK; ++j) {  
 hash[index] = result \* j + j;  
 index++;  
 }  
 start = halfK;  
 result = 1;  
 }  
 // для второй половины последующий расчет различных [k / 2; k] хэшей идет следующим образом:  
 // такой подход сделан для возможности создания разнообразных k хэшей для каждого элемента  
 if (i == \*element - 1) {  
 for (int j = start; j < k + 1; ++j) {  
 hash[index] = (result + j) \* j;  
 index++;  
 }  
 }  
 }  
 return hash;  
}

3.6 hash.h

#ifndef BLOOM\_FILTER\_HASH\_H  
#define BLOOM\_FILTER\_HASH\_H  
  
#include "../Bloom/bloom.h"  
  
uint64\_t \*hash(const uint8\_t \*element, int k, bloom \*b);  
  
#endif

3.7 cmdFile.c

#include <stdio.h>  
#include <malloc.h>  
#include <stdlib.h>  
#include <string.h>  
#include "cmdFile.h"  
#include "../Errors/errors.h"  
#include <windows.h>  
  
void controlSection(int line, int count, char \*currentSection) {  
 char \*section;  
 if (line == 0) {  
 section = "Filter length: ";  
 for (int i = 0; i < count; ++i) {  
 if (currentSection[i] != section[i]) errors(4, currentSection);  
 }  
 }  
 if (line == 1) {  
 section = "Number of elements: ";  
 for (int i = 0; i < count; ++i) {  
 if (currentSection[i] != section[i]) errors(4, currentSection);  
 }  
 }  
 if (line == 2) {  
 section = "Elements: ";  
 for (int i = 0; i < count; ++i) {  
 if (currentSection[i] != section[i]) errors(4, currentSection);  
 }  
 }  
}  
  
bloom \*readFile(char \*fileName) {  
 FILE \*file = fopen(fileName, "r");  
 if (file == NULL) errors(2, fileName);  
 char ch;  
 bloom \*b;  
 char \*string = (char \*)calloc(1, sizeof(char));  
 if (string == NULL) errors(11, "string");  
 char \*controlOfSections = (char \*)calloc(1, sizeof(char));  
 if (controlOfSections == NULL) errors(11, "controlOfSections");  
 // Текущая строка line  
 int line = 0;  
 // Переменная count используется для проверки текущего символа (текущей строки line)  
 int count = 0;  
 int index = 0;  
 int filterLength = 0;  
 int numOfElem = 0;  
 while ((ch = fgetc(file)) != EOF) {  
 if (ch == '\n') {  
 line++;  
 index = 0;  
 count = 0;  
 controlOfSections = (char \*)calloc(1, sizeof(char));  
 if (controlOfSections == NULL) errors(11, "controlOfSections");  
 }  
 // Проверка, является ли текущий символ строки 1 - 16м  
 if (count == 15 && line == 0) controlSection(line, count, controlOfSections);  
 if (count > 14 && line == 0) {  
 string[index] = ch;  
 index++;  
 string = (char \*)realloc(string, index + 1);  
 if (string == NULL) errors(12, "string");  
 }  
 // Проверка, является ли текущий символ строки 2 - 20м  
 if (count == 20 && line == 1) controlSection(line, count, controlOfSections);  
 if (count > 19 && line == 1) {  
 string[index] = ch;  
 index++;  
 string = (char \*)realloc(string, index + 1);  
 if (string == NULL) errors(12, "string");  
 }  
 // Проверка, является ли текущий символ строки 3 - 10м  
 if (count == 10 && line == 2) controlSection(line, count, controlOfSections);  
 if (count > 9 && line == 2 && ch != ' ') {  
 string[index] = ch;  
 index++;  
 if (index > 1) errors(7, fileName);  
 string = (char \*)realloc(string, index + 1);  
 if (string == NULL) errors(12, "string");  
 }  
 if (line == 1 && filterLength == 0) {  
 filterLength = strtol(string, 0, 10);  
 string = (char \*)calloc(1, sizeof(char));  
 if (string == NULL) errors(11, "string");  
 }  
 if (line == 2 && numOfElem == 0) {  
 numOfElem = strtol(string, 0, 10);  
 string = (char \*)calloc(1, sizeof(char));  
 if (string == NULL) errors(11, "string");  
 if (filterLength < 1 || numOfElem < 1) errors(5, fileName);  
 b = create(filterLength, numOfElem);  
 if (b->hashNum < 1) errors(6, fileName);  
 }  
 if (ch == ' ' && line == 2 && count > 10) {  
 index = 0;  
 add(string, b);  
 string = (char \*)calloc(1, sizeof(char));  
 if (string == NULL) errors(11, "string");  
 }  
 if (ch != '\n') {  
 controlOfSections[count] = ch;  
 count++;  
 controlOfSections = (char \*) realloc(controlOfSections, count + 1);  
 if (controlOfSections == NULL) errors(12, "controlOfSections");  
 }  
 }  
 if (line > 2) errors(3, fileName);  
 add(string, b);  
 free(string);  
 free(controlOfSections);  
 fclose(file);  
 printf("Конфигурационный файл успешно прочитан\n");  
 return b;  
}  
  
int commandLine(char \*line, bloom \*b, char \*fileName) {  
 SetConsoleOutputCP(CP\_UTF8);  
 scanf("%s", line);  
 if (line[0] != '-' || (line[1] != 'a' && line[1] != 'c' && line[1] != 'e' && line[1] != 'h')) errors(8, fileName);  
 if (line[0] == '-' && line[1] == 'e') return 0;  
 if (line[0] == '-' && line[1] == 'h') {  
 printf("Командная строка имеет вид -key\_n\n");  
 printf("-key - режим работы\n-a - добавление элемента\n-c - проверка содержания\n-h - помощь\n-e - выход");  
 printf("\n\_n - элемент, который необходимо добавить или проверить");  
 }  
 if (line[0] == '-' && line[1] == 'a') {  
 if (strlen(line) == 4) {  
 if (b->numOfAddedElem + 1 <= b->numOfElements) {  
 add(&line[3], b);  
 printf("Элемент: %c успешно добавлен\n", line[3]);  
 } else errors(9, fileName);  
 } else errors(10, fileName);  
 }  
 if (line[0] == '-' && line[1] == 'c') {  
 if (strlen(line) == 4) {  
 if (contains(&line[3], b) == 1) printf("Элемент: %c возможно есть\n", line[3]);  
 else printf("Элемента: %c точно нет\n", line[3]);  
 } else errors(10, fileName);  
 }  
 return 1;  
}

3.8 cmdFile.h

#ifndef BLOOM\_FILTER\_CMDFILE\_H  
#define BLOOM\_FILTER\_CMDFILE\_H  
  
#include "../Bloom/bloom.h"  
  
bloom \*readFile(char \*fileName);  
int commandLine(char \*line, bloom \*b, char \*fileName);  
  
#endif

3.9 errors.c

#include <stdio.h>  
#include "errors.h"  
#include <windows.h>  
  
void errors(int err, char \*string) {  
 SetConsoleOutputCP(CP\_UTF8);  
 if (err == 1) {  
 printf("\nОшибка № %d: Неккоректный вид командной строки. Ожидалось: ...\\Bloom\_filter.exe \*Путь файла конфигурации\*\n", err);  
 exit(err);  
 }  
 if (err == 2) {  
 printf("\nОшибка № %d: Не удалось прочитать файл: %s. Возможно его не существует\n", err, string);  
 exit(err);  
 }  
 if (err == 3) {  
 printf("\nОшибка № %d: Файл: %s неправильно задан. Ожидалось:\nFilter length:"  
 "\nNumber of elements:\nElements:\nКоличество строк больше трех", err, string);  
 exit(err);  
 }  
 if (err == 4) {  
 printf("\nОшибка № %d: Раздел: %s неправильно задан. Ожидалось:\nFilter length:\nNumber of elements:"  
 "\nElements:", err, string);  
 exit(err);  
 }  
 if (err == 5) {  
 printf("\nОшибка № %d: Неправильно задана длина битового поля или количество элементов. Эти значения должны быть больше нуля\n", err);  
 exit(err);  
 }  
 if (err == 6) {  
 printf("\nОшибка № %d: Битовое поле слишком мало для такого количества элементов\n", err);  
 exit(err);  
 }  
 if (err == 7) {  
 printf("\nОшибка № %d: Некорректный входной элемент, длина больше одного\n", err);  
 exit(err);  
 }  
 if (err == 8) {  
 printf("\nОшибка № %d: Некорректный ключ командной строки. Для просмотра режимов работы, введите ключ: -h\n", err);  
 exit(err);  
 }  
 if (err == 9) {  
 printf("\nОшибка № %d: Больше нельзя добавить элемент\n", err);  
 exit(err);  
 }  
 if (err == 10) {  
 printf("\nОшибка № %d: Не задан элемент для добавления или проверки\n", err);  
 exit(err);  
 }  
 if (err == 11) {  
 printf("\nОшибка № %d: Ошибка выделения памяти под %s\n", err, string);  
 exit(err);  
 }  
 if (err == 12) {  
 printf("\nОшибка № %d: Ошибка перераспределения памяти под %s\n", err, string);  
 exit(err);  
 }  
}

3.10 errors.h

#ifndef BLOOM\_FILTER\_ERRORS\_H  
#define BLOOM\_FILTER\_ERRORS\_H  
  
void errors(int err, char \*string);  
  
#endif

3.11 test.c

#include <stdio.h>  
#include "test.h"  
#include "../Bloom/bloom.h"  
  
void test1() {  
 printf("Test: 1\n");  
 int length = 6;  
 int elementsNum = 2;  
 bloom \*b = create(length, elementsNum);  
  
 int elem1 = 6;  
 add(&elem1, b);  
 int cont = contains(&elem1, b);  
 printf("expected: true | result: ");  
 if (cont == 1) printf("true");  
 else printf("false");  
 printf("\n");  
  
 int elem2 = 'c';  
 add(&elem2, b);  
 cont = contains(&elem2, b);  
 printf("expected: true | result: ");  
 if (cont == 1) printf("true");  
 else printf("false");  
  
 delete(b);  
}  
  
void test2() {  
 printf("\n\nTest: 2\n");  
 int length = 12;  
 int elementsNum = 4;  
 bloom \*b = create(length, elementsNum);  
  
 int elem = 1;  
 add(&elem, b);  
 elem++;  
 add(&elem, b);  
 int elem1 = 'c';  
 add(&elem1, b);  
 int cont = contains(&elem1, b);  
 printf("expected: true | result: ");  
 if (cont == 1) printf("true\n");  
 else printf("false\n");  
  
 double elem2 = 1.01;  
 cont = contains(&elem2, b);  
 printf("expected: false | result: ");  
 if (cont == 1) printf("true\n");  
 else printf("false\n");  
  
 add(&elem2, b);  
 cont = contains(&elem1, b);  
 printf("expected: true | result: ");  
 if (cont == 1) printf("true\n");  
 else printf("false\n");  
  
 delete(b);  
}  
  
void test3() {  
 printf("\n\nTest: 3\n");  
 int length = 3;  
 int elementsNum = 1;  
 bloom \*b = create(length, elementsNum);  
  
 char \*elem = "asd";  
 int cont = contains(&elem, b);  
 printf("expected: false | result: ");  
 if (cont == 1) printf("true\n");  
 else printf("false\n");  
  
 add(&elem, b);  
 cont = contains(&elem, b);  
 printf("expected: true | result: ");  
 if (cont == 1) printf("true\n");  
 else printf("false\n");  
  
 delete(b);  
}  
  
//int main() {  
// test1();  
// test2();  
// test3();  
// return 0;  
//}

3.12 test.h

#ifndef BLOOM\_FILTER\_TEST\_H  
#define BLOOM\_FILTER\_TEST\_H  
  
int main();  
  
#endif

**4. Коды ошибок**

**1.** Ошибка № 1: Неккоректный вид командной строки. Ожидалось: ...\\Bloom\_filter.exe \*Путь файла конфигурации\*

**2.** Ошибка № 2: Не удалось прочитать файл: \*Путь файла конфигурации\*. Возможно его не существует

**3.** Ошибка № 3: Файл: \*Путь файла конфигурации\* неправильно задан. Ожидалось:

Filter length:

Number of elements:

Elements:

Количество строк больше трех

**4.** Ошибка № 4: Раздел: \*Некорректное имя раздела\* неправильно задан. Ожидалось:

Filter length:

Number of elements:

Elements

**5.** Ошибка № 5: Неправильно задана длина битового поля или количество элементов. Эти значения должны быть больше нуля

**6.** Ошибка № 6: Битовое поле слишком мало для такого количества элементов

**7.** Ошибка № 7: Некорректный входной элемент, длина больше одного

**8.** Ошибка № 8: Некорректный ключ командной строки. Для просмотра режимов работы, введите ключ: -h

**9.** Ошибка № 9: Больше нельзя добавить элемент

**10.** Ошибка № 10: Не задан элемент для добавления или проверки\

**11.** Ошибка № 11: Ошибка выделения памяти под \*Название переменной, для которой выделялась память\*

**12.** Ошибка № 12: Ошибка перераспределения памяти под \*Название переменной, для которой выделялась память\*

**5. Программа и методика испытаний**

**5.1 С помощью тестов**

Разработанная библиотека была протестирована с помощью тестов из файла test.c. На рисунке 1 представлен результат тестирования.

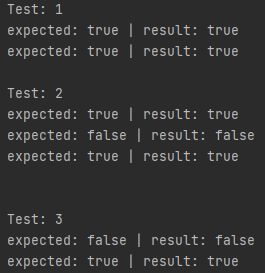


Рис. 1 Результат тестирования

Из рисунка 1 видно, что библиотека работает корректно.

**5.2 Консольный режим**

**Ошибка № 1:**

При запуске программы с некорректной командной строкой (количество аргументов командной строки не равно двум), будет выведена ошибка:



Рис. 2 Ошибка № 1

**Ошибка № 2:**

При запуске программы с несуществующим файлом данных, будет выведена ошибка:

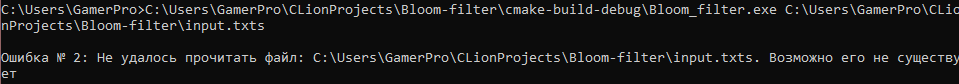
****

Рис. 3 Ошибка № 2

**Ошибка № 3:**

Если файл не имеет вид:



Рис. 4 Вид входного файла

Например будет:

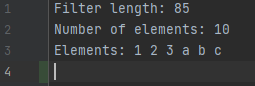


Рис. 5 Некорректный входной файл

Тогда будет выведена ошибка:

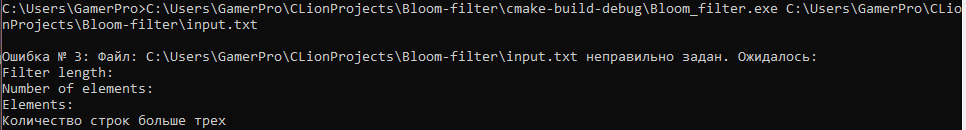


Рис. 6 Ошибка № 3

**Ошибка № 4:**

Если во входном файле разделы будут иметь некорректный вид, например:

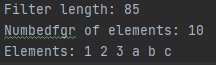
****

Рис. 7 Некорректно заданный раздел

Тогда в консоль будет выведена ошибка:

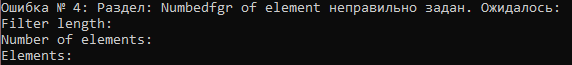


Рис. 8 Ошибка № 4

**Ошибка № 5:**

Если во входном файле длина битового поля или количества элементов будет меньше 1, а также иметь некорректный вид: Filter length: asd85, то в консоль будет выведена ошибка:

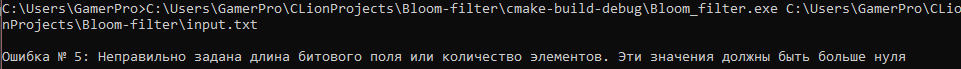


Рис. 9 Ошибка № 5

**Ошибка № 6:**

Если при заданном размере битового поля, количество элементов будет слишком велико, то есть результат формулы: (где n – длина поля, m – кол-во элементов) будет меньше 1, тогда будет выведена ошибка:

Пусть n = 10, m = 7

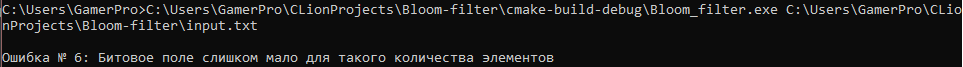


Рис. 10 Ошибка № 6

**Ошибка № 7:**

Если входные элементы имеют некорректный вид, например: корректный вид: “a”, некорректный вид: “asd”, тогда будет выведена ошибка:

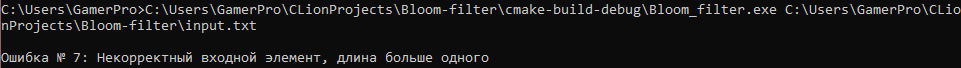
****

Рис. 11 Ошибка № 7

**Ошибка № 8:**

Если будет введен несуществующий ключ командной строки, то будет выведена ошибка:

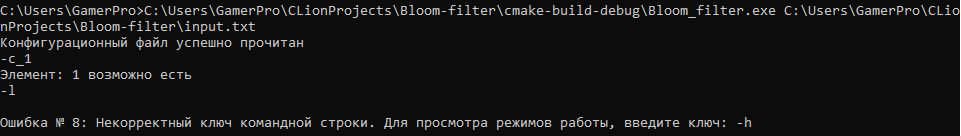


Рис. 12 Ошибка № 8

**Ошибка № 9:**

Если количество добавляемых элементов будет превышать заданное количество элементов, то будет выведена ошибка:

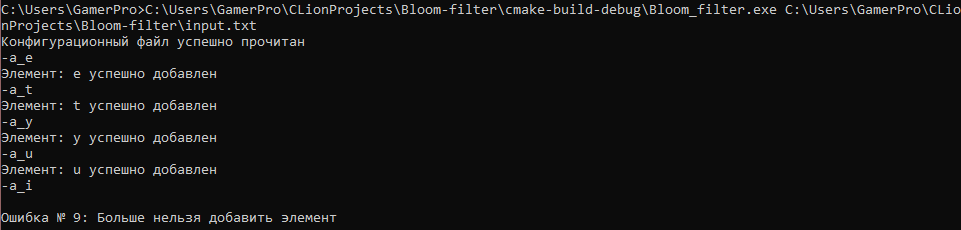
****

Рис. 13 Ошибка № 9

Изначально из файла было добавлено 6 элементов.

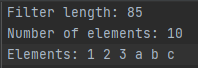


Рис. 14 Входные параметры

Далее было успешно добавлено 4 элемента, а 5 элемент добавить уже нельзя.

**Ошибка № 10:**

Если вместе с ключом добавления/проверки не будет задан элемент, то будет выведена ошибка:

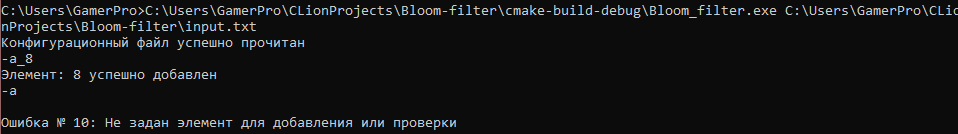


Рис. 15 Ошибка № 10

**Ошибка № 11:**

Если возникла ошибка перераспределения памяти с помощью realloc, то будет выведена ошибка:

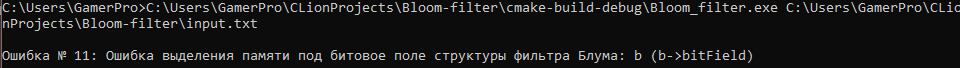


Рис. 16 Ошибка № 11

**Ошибка № 12:**

Если возникла ошибка выделения памяти с помощью malloc, то будет выведена ошибка:



Рис. 17 Ошибка № 12

**5.3 Выполнение программы без ошибок**

Входной файл имеет вид:

****

Рис. 18 Входной файл

На рисунке 19 представлена проверка элемента 7, добавление его и снова проверка.

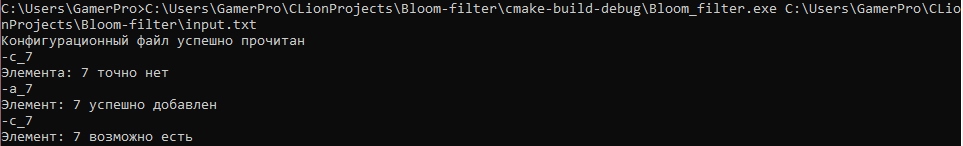


Рис. 19 Командная строка

На рисунке 20 представлена информация о режимах работы командной строки и выход из программы.

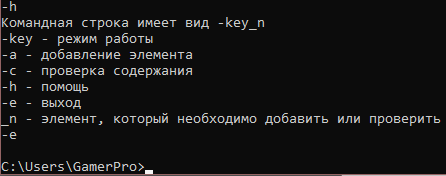


Рис. 20 Командная строка

**6. Ссылка на Github**

<https://github.com/SlimeLoveAllTheTime/Bloom_filter>