**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

**ЗАДАНИЕ 3**

Пояснительная записка

Выполнил студент группы БПИ196 (2)

Шестаков Михаил Сергеевич

**Вариант 28**. И снова пляшущие человечки. Узнав о планах преступников, Шерлок Холмс предложил лондонской полиции специальную машину для дешифровки сообщений злоумышленников. Реализовать многопоточное приложение, дешифрующее кодированный текст. В качестве ключа используется известная кодовая таблица, устанавливающая однозначное соответствие между каждой буквой и каким-нибудь числом. Процессом узнавания кода в решении задачи пренебречь. Каждый поток дешифрует свои кусочки текста. При решении использовать парадигму портфеля задач.

**1. Модель вычислений**

В программе используется парадигма портфеля задач [1], [2]. Массив, который нужно дешифровать разбивается не несколько равных частей, после чего эти части складываются в портфель в виде заданий. Затем программа запускает несколько потоков. Каждый из потоков берёт из портфеля задачу и выполняет её. Если задач в портфеле не осталось, поток успешно завершается. Чтобы два потока не смогли случайно взять одну и ту же задачу, доступ к портфелю задач осуществляется с использованием критической секции.

**2. Текст программы**

Для удобства текст программы разбит на несколько файлов

**2.1 main.c**

#include <stdio.h>

#include <pthread.h>

#include <stdbool.h>

#include <stdlib.h>

#include "Util.h"

#include "Cipher.h"

#include "TasksBag.h"

#define threadsCount 8

#define countPerTask 1000

*// структура для храниения результата выполнения потока*

struct ThreadResult {

bool success;

int wrongValue;

};

struct TasksBag tasksBag; *// портфель задач*

int\* encoded; *// закодированные данные -- числа*

char\* decoded; *// декодированные данные -- символы (c-string)*

*// основная функция для потоков*

void\* runTaskLoop(void\* args) {

while (**true**) {

struct Task task;

if (!getNextTask(&tasksBag, &task)) {

**break**; *// если задачи в портфеле кончились, то выходим*

}

decode(encoded, decoded, task.firstIndex, task.lastIndex);

}

struct ThreadResult\* res = malloc(sizeof(struct ThreadResult));

res->success = **true**;

return res;

}

*// выходит из потока с ошибкой*

void reportError(int value) {

*// очищаем очередь, потому что программа всё равно завершится с ошибкой*

clearTasksBag(&tasksBag);

*// выходим из потока с неудачей*

struct ThreadResult\* res = malloc(sizeof(struct ThreadResult));

res->success = **false**;

res->wrongValue = value;

pthread\_exit((void\*)res);

}

*// создаёт задачи*

void createTasks(struct TasksBag\* \_tasksBag, int count) {

for (int i = 0; i < count; i += countPerTask) {

struct Task task;

task.firstIndex = i;

task.lastIndex = min(i + countPerTask, count);

addTask(\_tasksBag, task);

}

}

*// создаёт потоки*

void createThreads(pthread\_t\* \_threads) {

for (int i = 0; i < threadsCount; i++) {

pthread\_create(\_threads + i, NULL, runTaskLoop, NULL);

}

}

*// дожидается пока потоки отработают. возвращает false, если хотя бы один из потоков*

*// завершился с ошибкой*

bool joinThreads(pthread\_t\* \_threads) {

for (int i = 0; i < threadsCount; i++) {

struct ThreadResult\* result;

pthread\_join(\_threads[i], (void \*\*) &result);

if (!result->success) {

printf("Invalid value: %d**\n**", result->wrongValue);

return **false**;

}

free(result);

}

return **true**;

}

int main(int argc, char\*\* argv) {

*// количество символов*

int count;

*// считываем исходные данные*

if (!readCipherData(&count, &encoded, &decoded)) {

pause();

return 1;

}

*// создаём портфель задач*

initTasksBag(&tasksBag, (count + countPerTask - 1) / countPerTask);

createTasks(&tasksBag, count);

*// создаём потоки*

pthread\_t threads[threadsCount];

createThreads(threads);

*// запускаем потоки*

if (!joinThreads(threads)) {

pause();

return 1;

}

printf("Decoded message: ");

*// выводим результат*

puts(decoded);

*// ждём ввода пользователя*

pause();

*// очищаем память*

destroyTasksBag(&tasksBag);

free(encoded);

free(decoded);

return 0;

}

**2.2 Cipher.h**

#ifndef CIPHER\_H

#define CIPHER\_H

#include <stdio.h>

#include <stdlib.h>

char cipherTable[255];

void reportError(int value);

*// расшифровывает данные с промежутка [firstIndex, lastIndex)*

void decode(int\* encoded, char\* decoded, int firstIndex, int lastIndex) {

for (int i = firstIndex; i < lastIndex; i++) {

int value = encoded[i];

if (value > 256 || value < 0 || cipherTable[value] == 0) {

reportError(value);

}

decoded[i] = cipherTable[value];

}

}

*// читает таблицу шифрования*

bool readCipherTable() {

int cipherCount = 0;

printf("Count of cipher entries: ");

if (scanf("%d", &cipherCount) != 1) {

printf("Unable to read count of cipher entries**\n**");

return **false**;

}

if (cipherCount < 0 || cipherCount > 256) {

printf("Incorrect count of cipher entries**\n**");

return **false**;

}

char tmp[100];

printf("Cipher entries (in format '<character> = <number>**\\**n'):**\n**");

for (int i = 0; i < cipherCount; i++) {

char c;

int number;

scanf("%[**\n** ]", tmp); *// пропускаем пробелы и переносы строк*

if (scanf("%c%[= ]", &c, tmp) != 2 || scanf("%d", &number) != 1) {

printf("Unable to read cipher entry**\n**");

return **false**;

}

if (number < 0 || number > 255) {

printf("Incorrect number: %d. Number should be between 0 and 255**\n**", number);

return **false**;

}

if (cipherTable[number] != 0) {

printf("Repeating entries in cipher table");

return **false**;

}

cipherTable[number] = c;

}

return **true**;

}

*// читает данные, которые нужно расшифровать*

bool readEncoded(int count, int\* encoded) {

printf("Encoded characters: (in format '<number1> <number2> ... <numberN>'**\n**");

for (int i = 0; i < count; i++) {

if (scanf("%d", encoded + i) != 1) {

printf("Unable to read %d encoded**\n**", i + 1);

return **false**;

}

}

return **true**;

}

*// читает данные, которые нужно расшифровать и таблицу шифрования*

bool readCipherData(int\* count, int\*\* encoded, char\*\* decoded) {

printf("Count of encoded characters: ");

if (scanf("%d", count) != 1) {

printf("Unable to read encoded count**\n**");

return **false**;

}

if (\*count < 0 || \*count > 5e8) {

printf("Incorrect encoded count**\n**");

return **false**;

}

\*encoded = malloc(\*count \* sizeof(int));

\*decoded = malloc((\*count + 1) \* sizeof(char));

(\*decoded)[\*count] = 0; *// записываем в конец нуль-турминатор*

if (!readEncoded(\*count, \*encoded)) {

return **false**;

}

return readCipherTable();

}

#endif

**2.3 TasksBag.h**

#ifndef TASKS\_BAG\_H

#define TASKS\_BAG\_H

#include <stdlib.h>

struct Task {

int firstIndex;

int lastIndex;

};

*// класс для синхронного портфеля задач*

struct TasksBag {

struct Task\* tasks;

int curTask;

int tasksCount;

pthread\_mutex\_t tasksMutex;

};

*// инициализирует портфель задач*

void initTasksBag(struct TasksBag\* \_tasksBag, int maxTasksCount) {

\_tasksBag->tasks = malloc(maxTasksCount \* sizeof(struct Task));

\_tasksBag->curTask = 0;

\_tasksBag->tasksCount = 0;

pthread\_mutex\_init(&\_tasksBag->tasksMutex, NULL);

}

*// добавляет задачу в портфель задач*

void addTask(struct TasksBag\* \_tasksBag, struct Task task) {

\_tasksBag->tasks[\_tasksBag->tasksCount++] = task;

}

*// удаляет все задачи из портфеля задач*

void clearTasksBag(struct TasksBag\* \_tasksBag) {

pthread\_mutex\_lock(&\_tasksBag->tasksMutex);

\_tasksBag->curTask = \_tasksBag->tasksCount = 0;

pthread\_mutex\_unlock(&\_tasksBag->tasksMutex);

}

*// деструктор портфеля задач*

void destroyTasksBag(struct TasksBag\* \_tasksBag) {

free(\_tasksBag->tasks);

pthread\_mutex\_destroy(&\_tasksBag->tasksMutex);

}

*// получает следующую задачу из портфеля*

bool getNextTask(struct TasksBag\* \_tasksBag, struct Task\* task) {

pthread\_mutex\_lock(&\_tasksBag->tasksMutex);

if (\_tasksBag->curTask >= \_tasksBag->tasksCount) {

pthread\_mutex\_unlock(&\_tasksBag->tasksMutex);

return **false**;

}

\*task = \_tasksBag->tasks[\_tasksBag->curTask++];

pthread\_mutex\_unlock(&\_tasksBag->tasksMutex);

return **true**;

}

#endif

**2.4 Util.h**

#ifndef UTIL\_H

#define UTIL\_H

*// ждёт от пользователя ввода*

void pause() {

#ifdef \_WIN32

system("pause");

#else

system("read");

#endif

}

*// находит минимальное число из двух*

int min(int x, int y) {

if (x < y) {

return x;

}

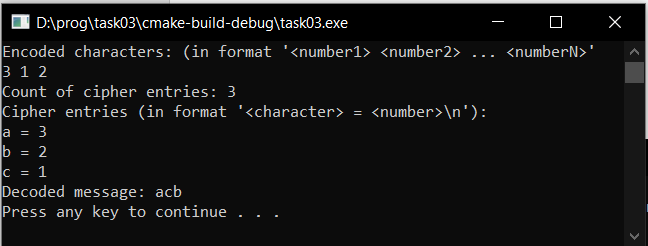
return y;

}

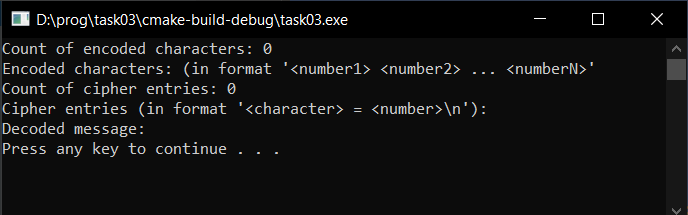
#endif

**3. Тестирование**

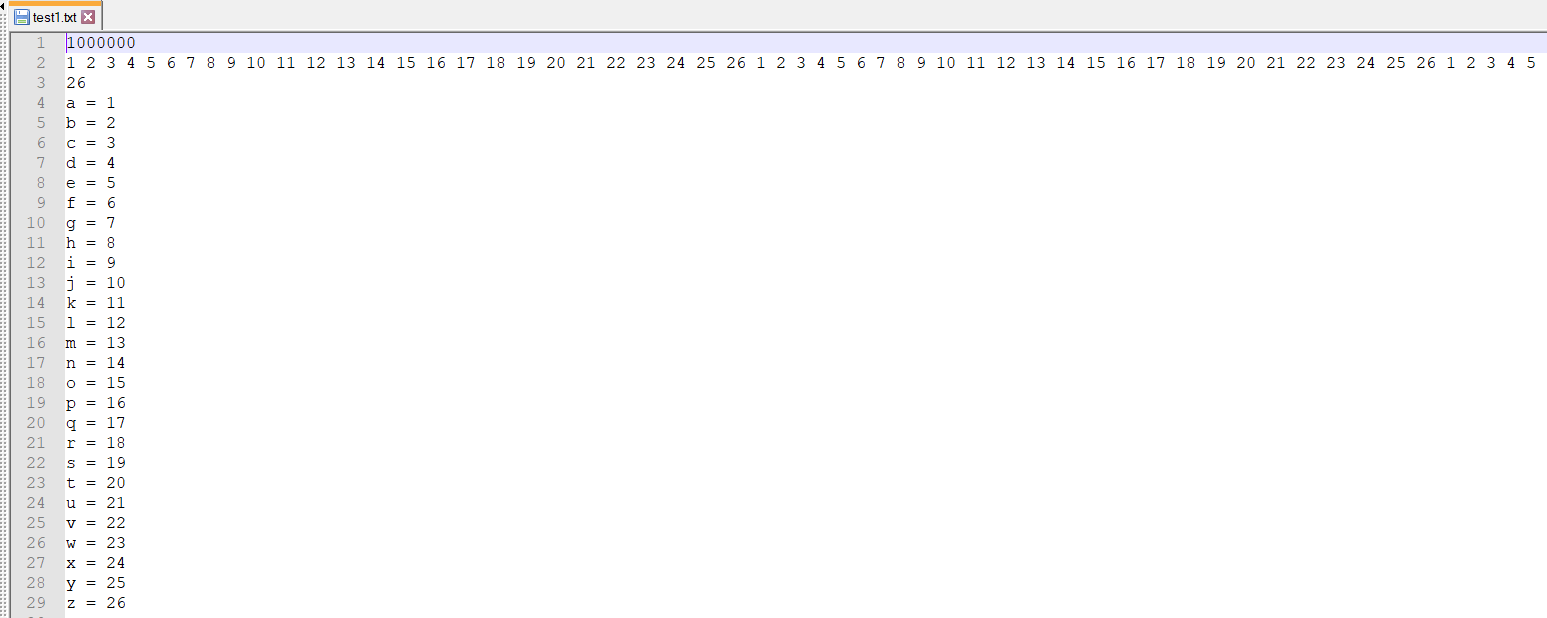
3.1. Простой тест

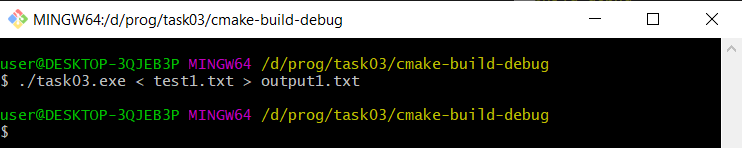


3.2. Пустой тест



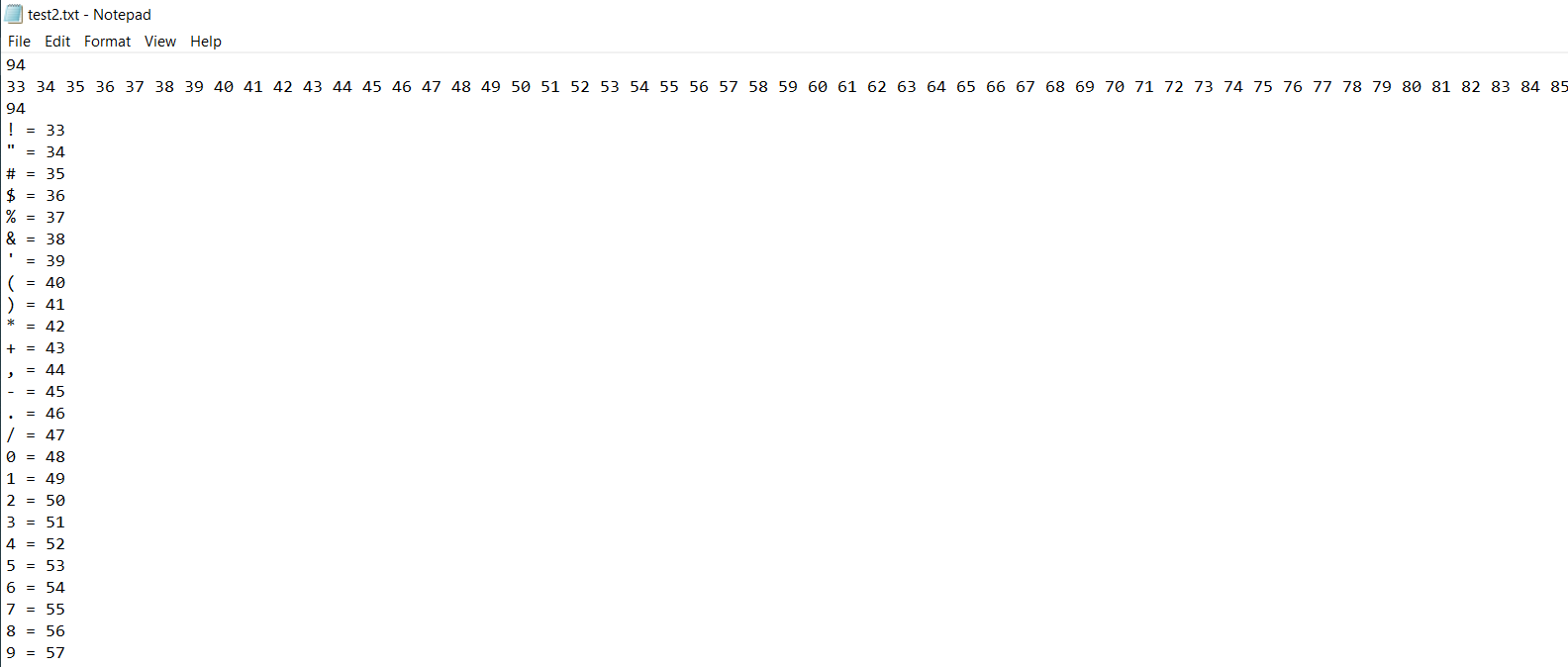
3.3 Большой тест (1 000 000 символов)

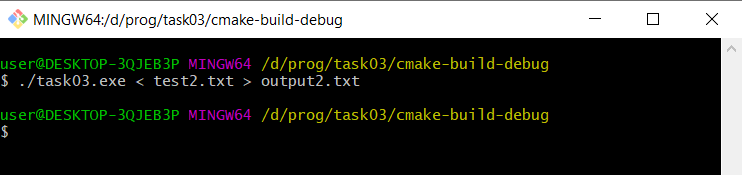


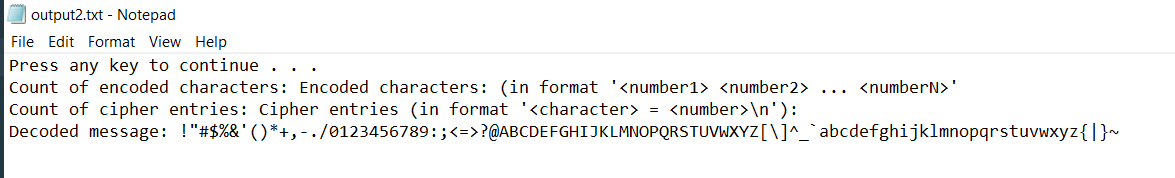




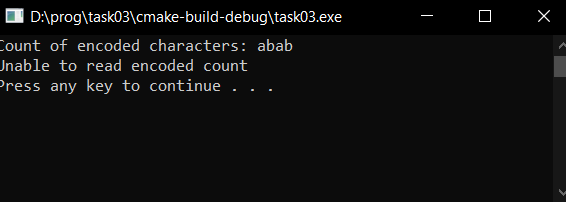
3.4 Все ASCII символы



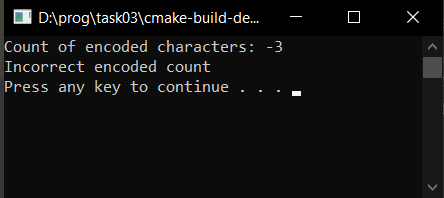




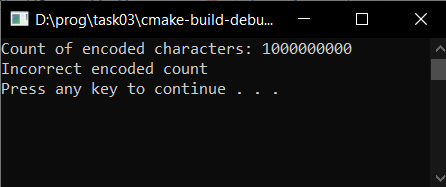
3.5. Некорректные данные – количество символов



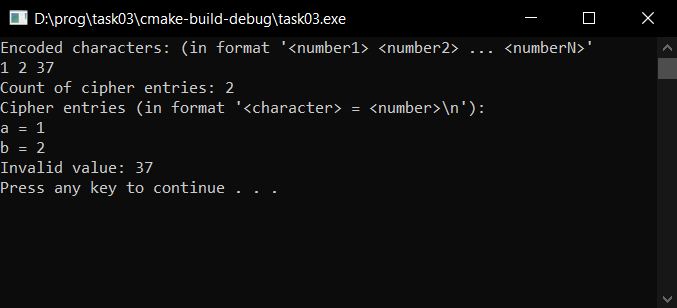
3.6. Некорректные данные – количество символов слишком маленькое



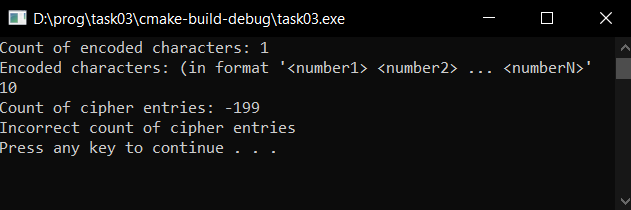
3.7. Некорректные данные – количество символов слишком большое



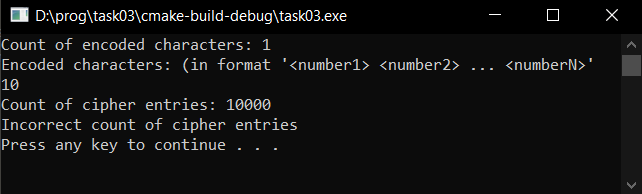
3.8 Некорректные данные – несуществующий закодированный символ



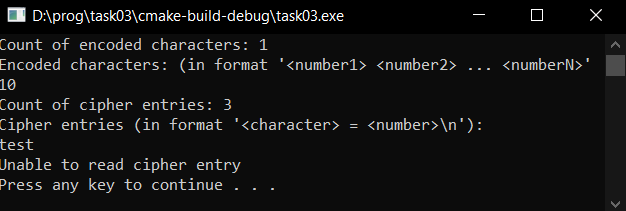
3.9 Некорректные данные – слишком мало символов в таблице шифрования



3.10 Некорректные данные – слишком много символов в таблице шифрования



3.11 Некорректные данные – неправильный формат в таблице шифрования



**Источники**

[1] https://l.wzm.me/\_coder/custom/parallel.programming/003.htm

[2] http://softcraft.ru/edu/comparch/tasks/t03/