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.

Москва 17.11.2020

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 хранения 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
```

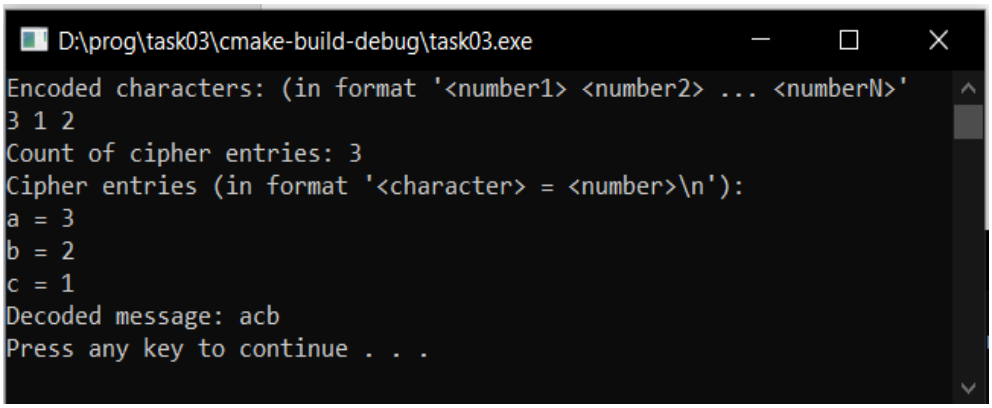
```c
#define TASKS_BAG_H

#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

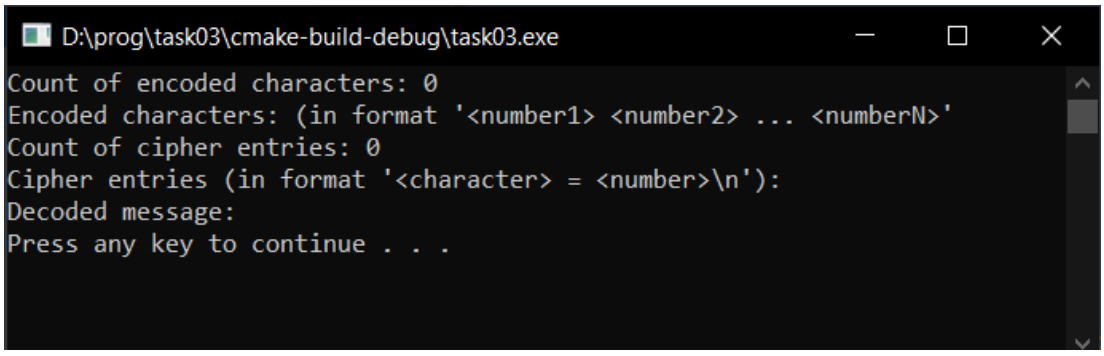
```
#ifndef UTIL_H
#define UTIL_H

// 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



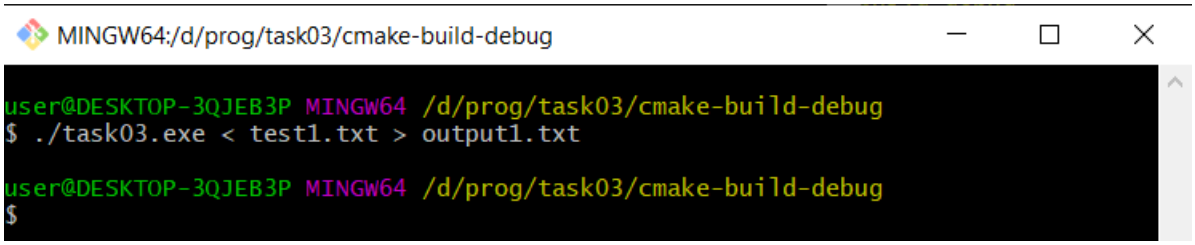
3.2. Пустой тест



3.3 Большой тест (1 000 000 символов)

test1.txt

```
1000000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 1 2 3 4 5
26
a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
h = 8
i = 9
j = 10
k = 11
l = 12
m = 13
n = 14
o = 15
p = 16
q = 17
r = 18
s = 19
t = 20
u = 21
v = 22
w = 23
x = 24
y = 25
z = 26
```



output1.txt - Notepad

File Edit Format View Help

```
Count of encoded characters: Encoded characters: (in format '<number1> <number2> ... <numberN>'
Count of cipher entries: Cipher entries (in format '<character> = <number>\n'):
Decoded message: abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefg
tuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopq
defghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyza
nopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijk
xyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstu
hijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcde
rstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmno
bcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxy
lmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghi
vwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs
fghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabc
pqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklm
zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvw
jklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefg
tuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopq
defghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyza
nopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijk
xyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstu
hijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcde
rstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmno
bcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxy
lmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghi
vwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs
fghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabc
pqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklm
zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvw
```

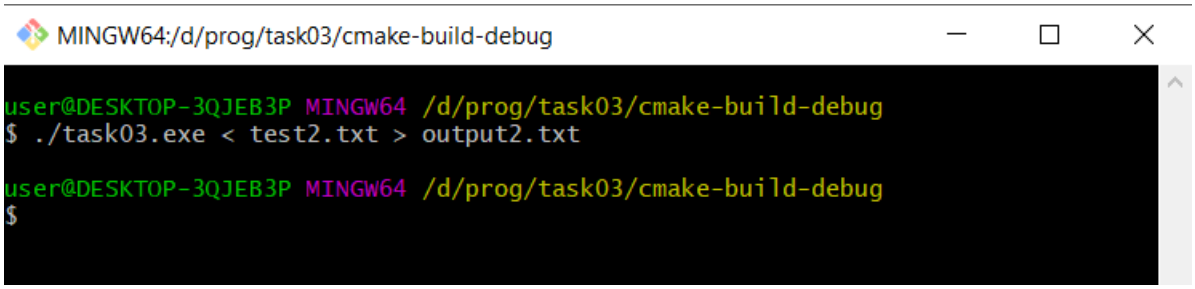
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

3.4 Все ASCII символы

test2.txt - Notepad

File Edit Format View Help

```
94
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
94
! = 33
" = 34
# = 35
$ = 36
% = 37
& = 38
' = 39
( = 40
) = 41
* = 42
+ = 43
, = 44
- = 45
. = 46
/ = 47
0 = 48
1 = 49
2 = 50
3 = 51
4 = 52
5 = 53
6 = 54
7 = 55
8 = 56
9 = 57
```

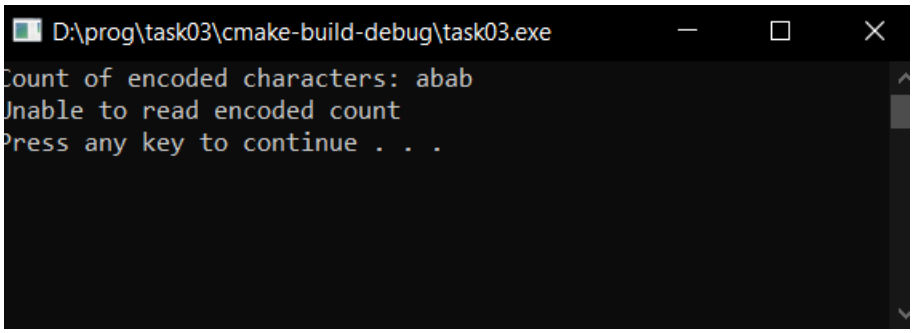


output2.txt - Notepad

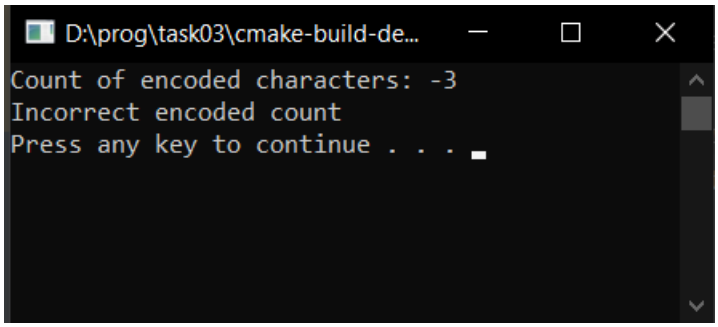
File Edit Format View Help

```
Press any key to continue . . .
Count of encoded characters: Encoded characters: (in format '<number1> <number2> ... <numberN>'
Count of cipher entries: Cipher entries (in format '<character> = <number>\n'):
Decoded message: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

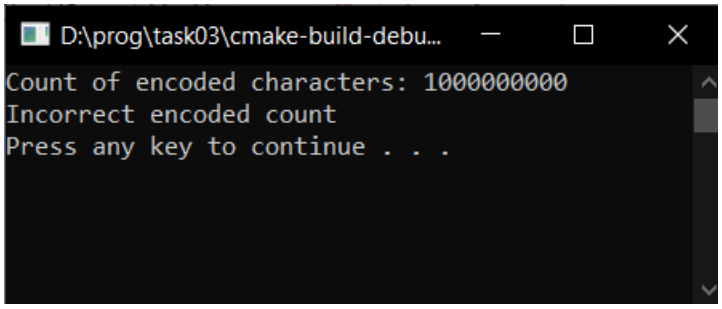
3.5. Некорректные данные – количество символов



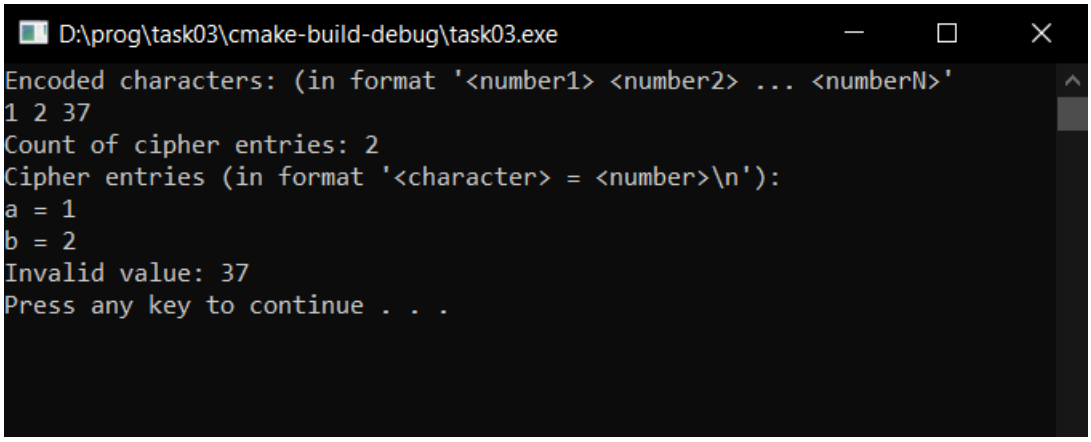
3.6. Некорректные данные – количество символов слишком маленькое



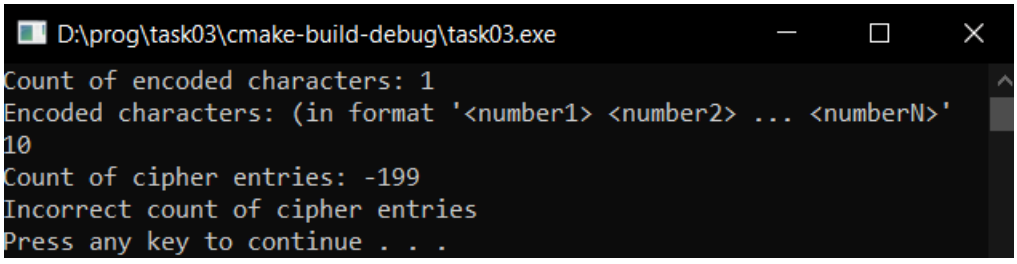
3.7. Некорректные данные – количество символов слишком большое



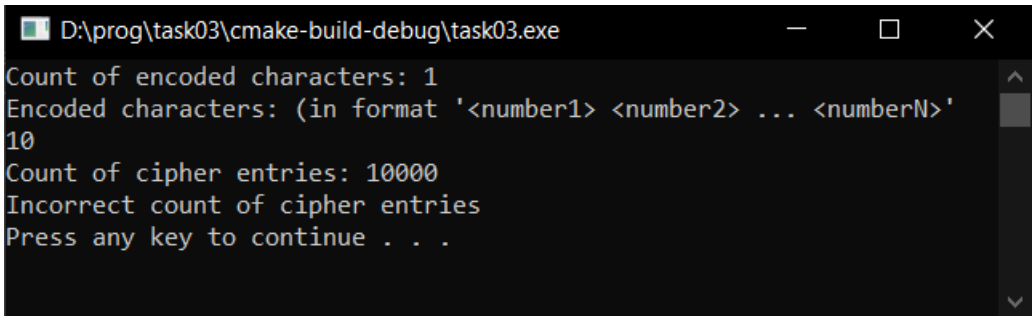
3.8 Некорректные данные – несуществующий закодированный символ



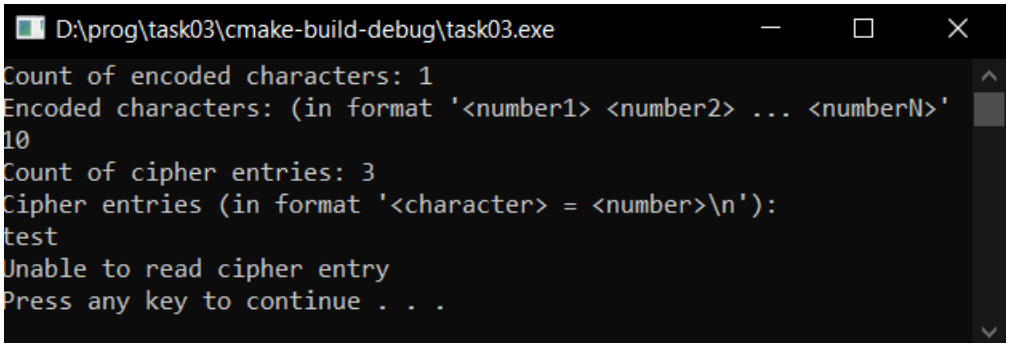
3.9 Некорректные данные – слишком мало символов в таблице шифрования



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/