**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

**ЗАДАНИЕ 4**

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]. Программа создаёт несколько потоков с помощью #pragma omp parallel [3]. Затем массив, который нужно дешифровать разбивается на несколько равных частей и с помощью #pragma omp task [4] каждая из частей складывается в портфель задач. Затем потоки начинают выполнение задач, синхронизация окончания выполнения осуществляется с помощью команды #pragma omp taskwait. [4]

После выполнения очередной задачи поток выводит об этом информацию в консоль. Для того, чтобы в один момент времени выводить в консоль мог только один поток используется критическая секция -- #pragma omp critical. [5]

**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 <omp.h>

#include "Util.h"

#include "Cipher.h"

#define countPerTask 10000

// структура для хранения результата выполнения потока

struct ThreadResult {

bool success;

int wrongValue;

};

struct ThreadResult result = {.success=true};

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

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

// выходит из потока с ошибкой

void reportError(int value) {

result.success = false;

result.wrongValue = value;

}

// создаёт задачи

void createTasks(int count) {

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

int index = i;

#pragma omp task

{

int firstIndex = index;

int lastIndex = min(index + countPerTask, count);

decode(encoded, decoded, firstIndex, lastIndex);

int numberOfThread = omp\_get\_thread\_num();

int countOfThreads = omp\_get\_num\_threads();

#pragma omp critical

{

printf("Thread %d / %d finished task %d**\n**", numberOfThread, countOfThreads, index);

}

}

}

}

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

// количество символов

int count;

// считываем исходные данные

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

pause();

return 1;

}

// запускаем секцию с несколькими потоками

#pragma omp parallel

{

// в одном потоке создаём задачи

#pragma omp single nowait

{

createTasks(count);

}

// ждём, пока все задачи завершатся

#pragma omp taskwait

}

// проверяем флаг ошибки

if (!result.success) {

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

pause();

return -1;

}

printf("Decoded message: ");

// выводим результат

puts(decoded);

// ждём ввода пользователя

pause();

// очищаем память

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 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;

}

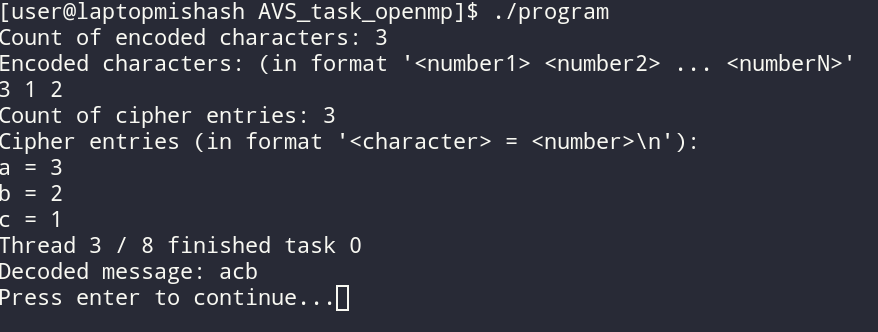
return y;

}

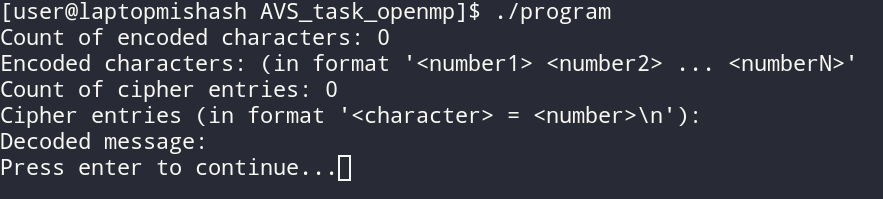
#endif

**3. Тестирование**

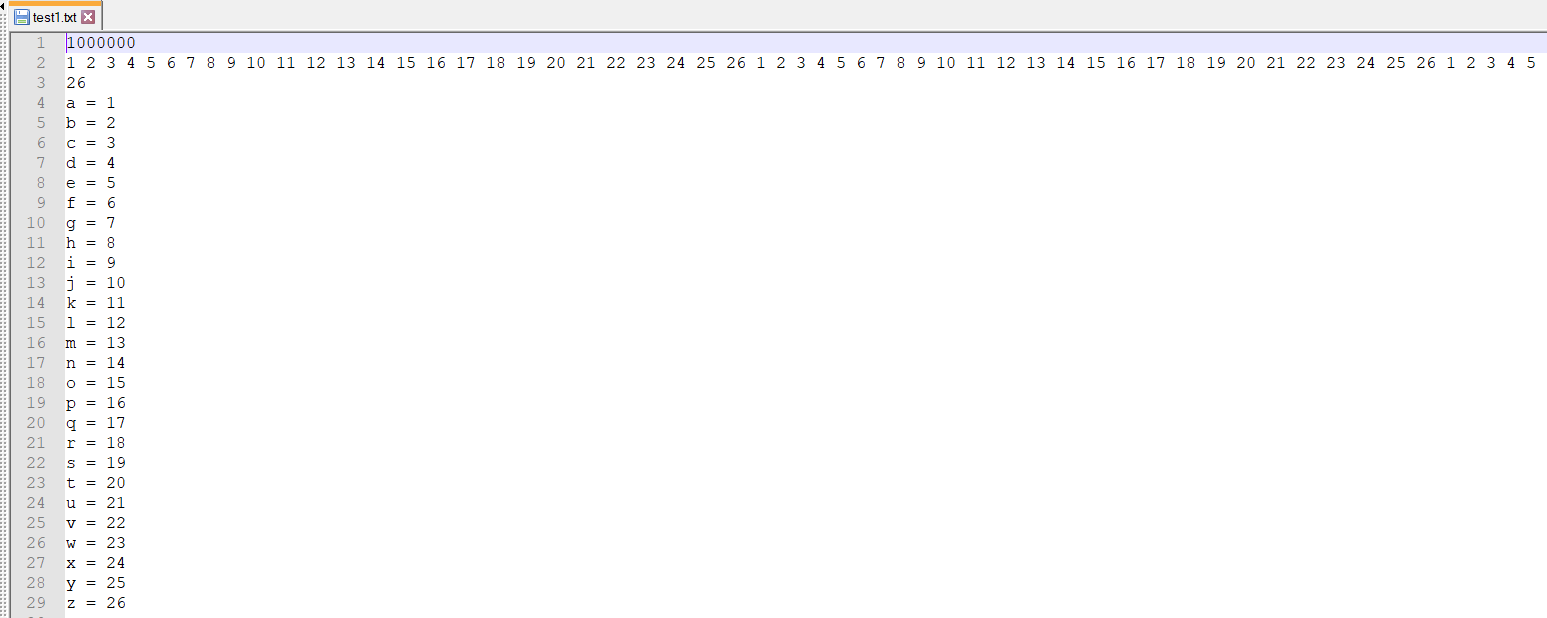
3.1. Простой тест



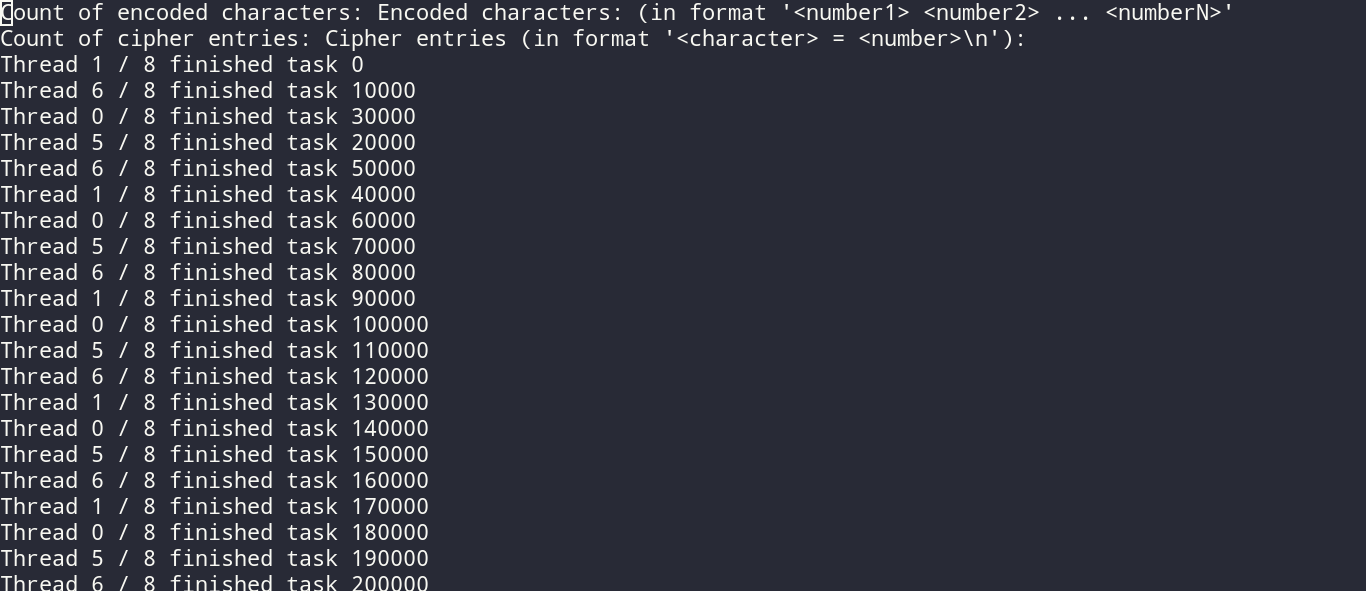
3.2. Пустой тест

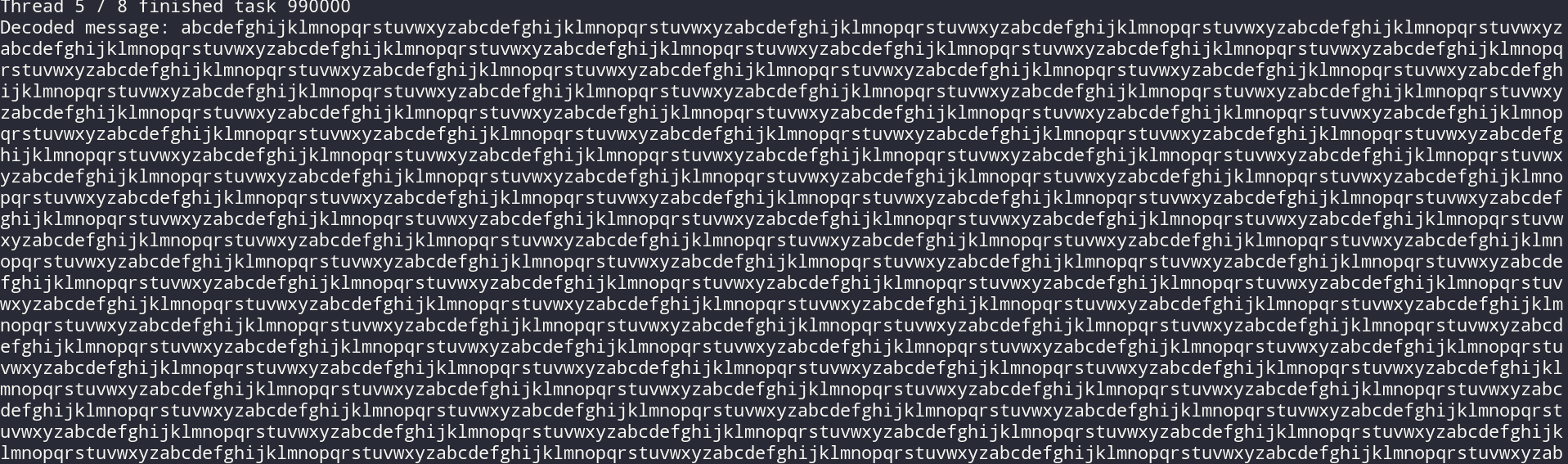


3.3. Большой тест (1 000 000 символов)

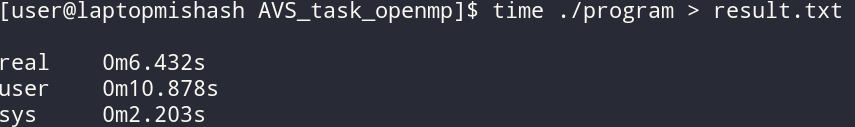


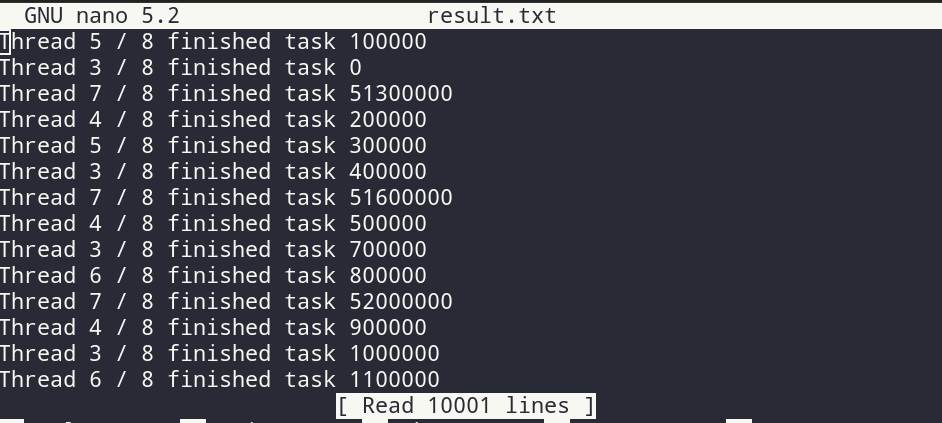




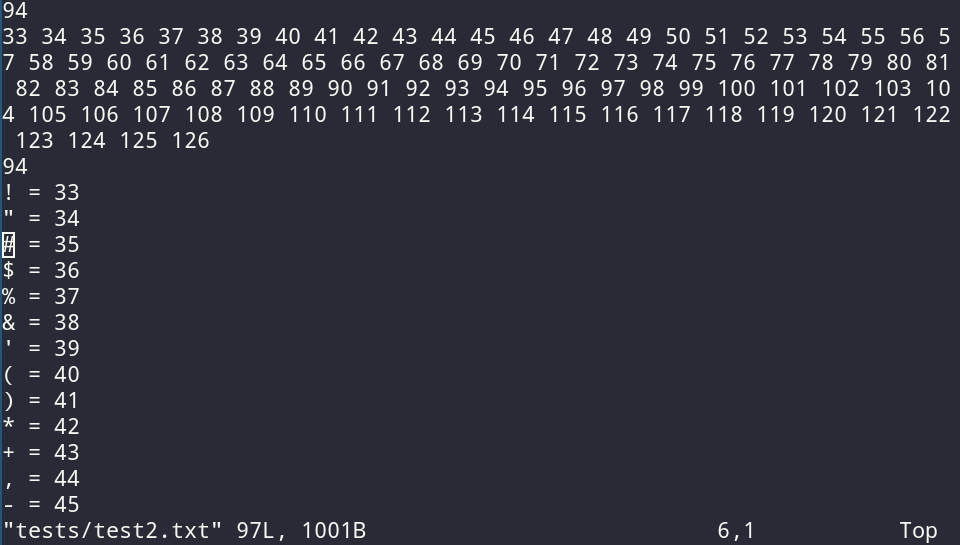


3.4. Очень большой тест (1000 000 000 символов). Вместо входного массива случайные данные, вывод результата отключен.

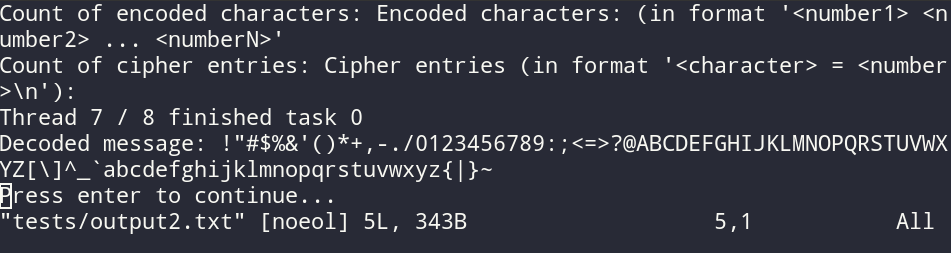




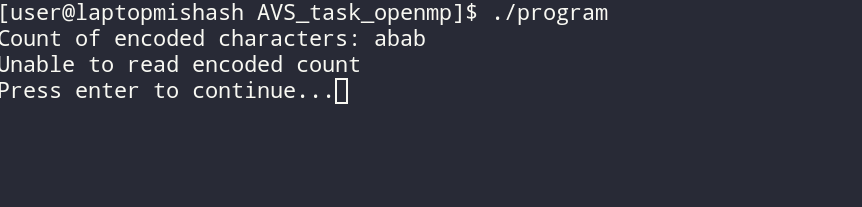
3.5. Все ASCII символы



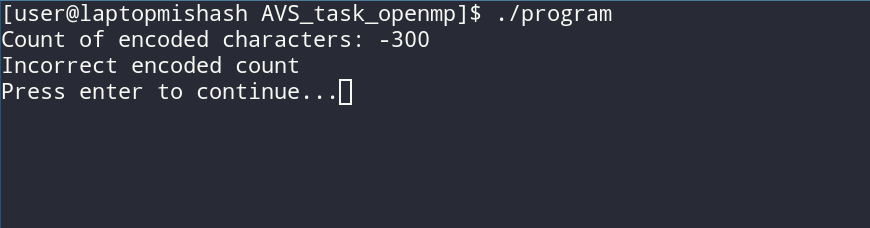




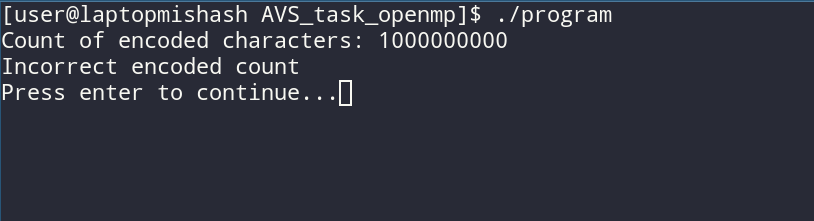
3.6. Некорректные данные – количество символов



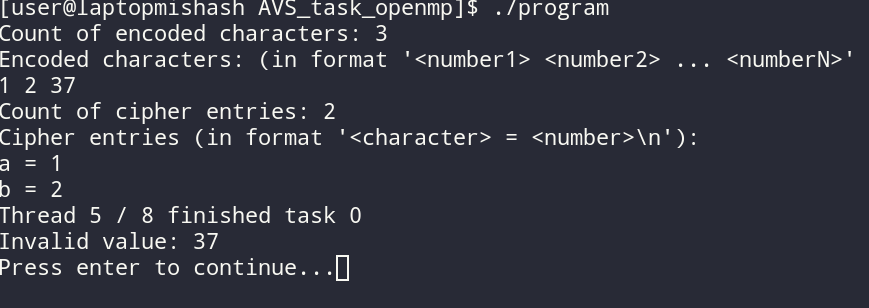
37. Некорректные данные – количество символов слишком маленькое



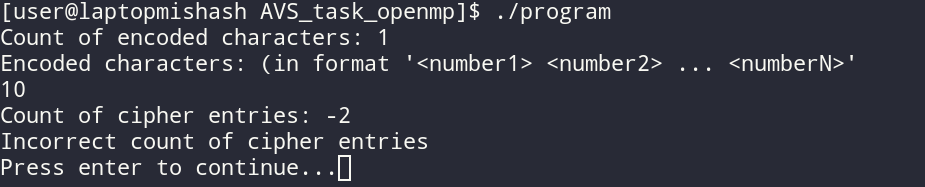
3.8. Некорректные данные – количество символов слишком большое



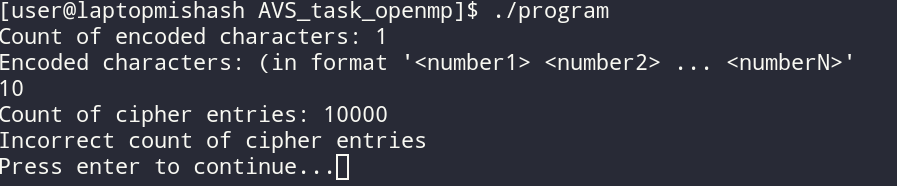
3.9.Некорректные данные – несуществующий закодированный символ



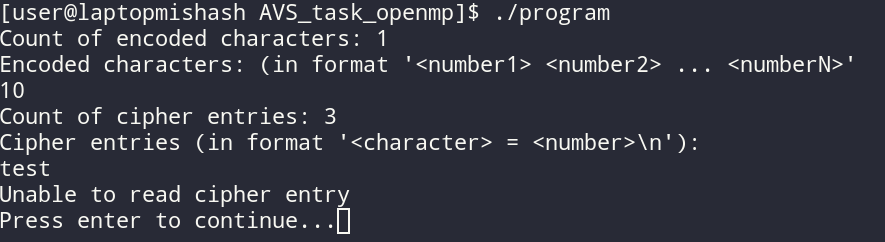
3.10. Некорректные данные – слишком мало символов в таблице шифрования



3.11. Некорректные данные – слишком много символов в таблице шифрования



3.12. 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/t04/>

[3] <https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-160#parallel>

[4] <https://en.wikibooks.org/wiki/OpenMP/Tasks>

[5] <https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-160#critical>