Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 3

на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА С ПОД UNIX. ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Выполнил             К. А. Тимофеев

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 5](#_Toc157722975)

[Выводы 6](#_Toc157722976)

[Список использованных источников 7](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является изучение среды программирования и основных инструментов, таких как компилятор/сборщик *gcc*, управление обработкой проекта *make* и языка *makefile*. Кроме того, на практике необходимо написать программу, на языке программирования *С*, реализующую шифрование и дешифрование символов по азбуке Морзе. Также необходимо создать *makefile* для управления обработкой проекта, собрать и протестировать исполняемый файл.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Язык программирования *С* является языком общего назначения, который широко используется для разработки системного программного обеспечения, прикладных программ и встраиваемых систем.[1]

*GCC* – это коллекция компиляторов для различных языков программирования, включая *С*. Также в данную коллекцию входят компиляторы для языков *С*++, *Go*. В зависимости от расширения имени файла, передаваемых в качестве параметра, и дополнительных опций, *GCC* запускает необходимые препроцессоры, компиляторы, сборщики.[2]

Компиляция – это процесс преобразования исходного кода программы из языка высокого уровня в машинный код. Компиляция происходит с помощью компилятора, который анализирует исходный код. Процесс компиляции преобразует исходный код программы в объектные файлы. Сборщик же создает исполняемый файл из одного или нескольких объектных файлов, полученных в результате компиляции.

*Make* – это утилита для автоматизации процесса сборки программы из исходных файлов. Make использует файл *makefile*, который содержит правила для компиляции и сборки проекта.[3]

Для больших проектов использование *makefile* и утилиты *make* позволяет достаточно сократить время компиляции и сборки.

Для выполнения данной лабораторной работы были использованы следующие сведения и концепции:

1 Язык программирования *С*: код программы был полностью написан на языке программирования *С*.

2 Стандартные библиотеки языка *С*: для работы с функциями ввода и вывода, строками и символами были использованы стандартные библиотеки языка *С*.

3 Компилятор *GCC*: для компиляции и сборки программы был использован компилятор *GCC*, который включает в себя компилятор языка *С*.

4 Система сборки make: для автоматизации процесса компиляции и сборки программы был создан *makefile*, в котором описаны все правила для компиляции и сборки различных модулей программы.

5 Модульное программирование: программа разделена на несколько модулей, каждый из которых отвечает за определенный функционал.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе выполнения лабораторной была разработана программа, которая позволяет шифровать и дешифровать текст по азбуке Морзе. Пользователю необходимо ввести данные в виде строки, состоящей из символов латинского алфавита и цифр.

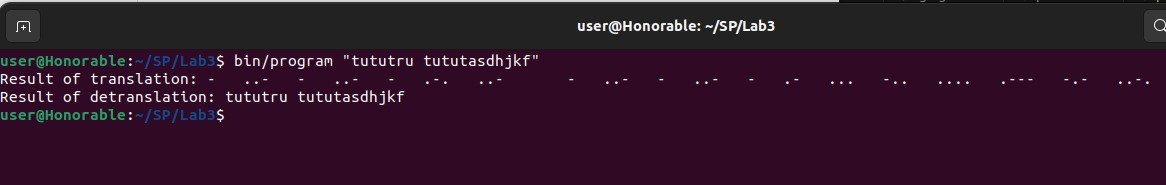


Рисунок 3.1 – Результат работы программы

Таким образом, в ходе данной лабораторной работы была реализована программа, шифрующая и дешифрующая символы по азбуке Морзе.

# **ВЫВОДЫ**

В ходе лабораторной работы были изучена среда программирования и основные инструменты, такие как компилятор/сборщик *gcc*, управление обработкой проекта make и языка *makefile*. Кроме того, была разработана программа на языке программирования *С*, реализующая шифрование и дешифрование символов по азбуке Морзе. Также был создан *makefile* для управления обработкой проекта, а также сборки и тестирования исполняемого файла.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Язык программирования С [Электронный ресурс]. – Режим доступа: https://metanit.com/c/tutorial/1.2.php. – Дата доступа: 08.02.2024.

[2] Компилятор GCC [Электронный ресурс]. – Режим доступа: https://metanit.com/c/tutorial/1.3.php. – Дата доступа: 08.02.2024.

[3] Makefile [Электронный ресурс]. – Режим доступа:   
https://habr.com/ru/articles/155201/. – Дата доступа: 08.02.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код главной функции

#include "stdio.h"

#include "morseTable.h"

#include "stdlib.h"

#include "parsers.h"

#include "translation.h"

#include "ctype.h"

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

if(argc == 1){

printf("Enter the string in arg!!!\n");

return 0;

}

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

printf("arg number %i -- %s\n", i, argv[i]);

}

struct MorseCode\* table = getTable();

int length;

for(int i = 0; i < strlen(argv[1]); ++i){

argv[1][i] = tolower(argv[1][i]);

}

char\* res = toMorse(table, argv[1]);

printf("res = %s\n", res);

char\* res2 = fromMorse(table, res);

printf("Result of detranslation: %s\n", res2);

free(res);

free(res2);

free(table);

}

Листинг 2 – Программный код morseTable.h

#ifndef MORSE\_TABLE

#define MORSE\_TABLE

#include "string.h"

#include "stdlib.h"

struct MorseCode{

char letter;

char code[7];

};

struct MorseCode\* getTable();

#endif

Листинг 3 – Программный код morseTable.c

#include "morseTable.h"

void setVals(struct MorseCode\* m, char letter, const char\* code){

(\*m).letter = letter; strcpy((\*m).code, code);

}

struct MorseCode\* getTable(){

struct MorseCode\* arr = (struct MorseCode\*) malloc(sizeof(struct MorseCode) \* 36);

//struct MorseCode\* arr = new MorseCode[36];

setVals(&arr[0], 'a', ".-");

setVals(&arr[1], 'b', "-...");

setVals(&arr[2], 'c', "-.-.");

setVals(&arr[3], 'd', "-..");

setVals(&arr[4], 'e', ".");

setVals(&arr[5], 'f', "..-.");

setVals(&arr[6], 'g', "--.");

setVals(&arr[7], 'h', "....");

setVals(&arr[8], 'i', "..");

setVals(&arr[9], 'j', ".---");

setVals(&arr[10], 'k', "-.-");

setVals(&arr[11], 'l', ".-..");

setVals(&arr[12], 'm', "--");

setVals(&arr[13], 'n', "-.");

setVals(&arr[14], 'o', "---");

setVals(&arr[15], 'p', ".--.");

setVals(&arr[16], 'q', "--.-");

setVals(&arr[17], 'r', ".-.");

setVals(&arr[18], 's', "...");

setVals(&arr[19], 't', "-");

setVals(&arr[20], 'u', "..-");

setVals(&arr[21], 'v', "...-");

setVals(&arr[22], 'w', ".--");

setVals(&arr[23], 'x', "-..-");

setVals(&arr[24], 'y', "-.--");

setVals(&arr[25], 'z', "--..");

setVals(&arr[26], '1', ".----");

setVals(&arr[27], '2', "..---");

setVals(&arr[28], '3', "...--");

setVals(&arr[29], '4', "....-");

setVals(&arr[30], '5', ".....");

setVals(&arr[31], '6', "-....");

setVals(&arr[32], '7', "--...");

setVals(&arr[33], '8', "---..");

setVals(&arr[34], '9', "----.");

setVals(&arr[35], '0', "-----");

return arr;

}

Листинг 4 – Программный код parsers.h

#ifndef PARSERS

#define PARSERS

#include <string.h>

#include "stdlib.h"

#include "stdio.h"

char\*\* parseToWords(int\* length\_out, const char\* text);

char\*\*\* parseToMorseWords(int\* length\_out, int\*\* lengthOfWords\_out, const char\* text);

char\*\* parseToMorseLetters(int\* length\_out, const char\* text);

void substr(const char\* src, char\* dst, int beg, int end);

void addStr(char\* dst, const char\* newstr, int\* cursor);

#endif

Листинг 5 – Программный код parsers.c

#include "parsers.h"

void substr(const char\* src, char\* dst, int beg, int end) {

for (int i = beg; i < end; ++i) {

dst[i - beg] = src[i];

}

dst[end - beg] = '\0';

}

char\*\* parseToWords(int\* length\_out, const char\* text) {

char\*\* res = NULL;

int length = 0;

int i = 0;

int begin = -1;

while (text[i] != '\0') {

if ((text[i] >= 'a' && text[i] <= 'z') ||

(text[i] >= 'A' && text[i] <= 'Z') ||

(text[i] >= '0' && text[i] <= '9')) {

if (begin == -1) {

begin = i;

}

}

else if (begin != -1) {

char\* word = (char\*)malloc(sizeof(char) \* (i - begin + 1));

substr(text, word, begin, i);

++length;

res = (char\*\*)realloc(res, length \* sizeof(char\*));

res[length - 1] = word;

begin = -1;

}

++i;

}

if (begin != -1) {

char\* word = (char\*)malloc(sizeof(char) \* (i - begin + 1));

substr(text, word, begin, i);

++length;

res = (char\*\*)realloc(res, length \* sizeof(char\*));

res[length - 1] = word;

begin = -1;

}

\*length\_out = length;

return res;

}

char\*\*\* parseToMorseWords(int\* length\_out, int\*\* lengthOfWords\_out, const char\* text) {

char\*\*\* res = NULL;

int length = 0;

int i = 0;

int beginOfSpaces = -1;

int beginOfWord = -1;

char\*\* morseWord = NULL;

int\* lengthOfWords = NULL;

int isTrackingSpaces = 0;

while (text[i] != '\0')

{

if (text[i] != ' ')

{

if (isTrackingSpaces)

{

if (i - beginOfSpaces == 7)

{

int wordLength = 0;

char\* word = (char\*)malloc(sizeof(char) \* (beginOfSpaces - beginOfWord + 1));

substr(text, word, beginOfWord, beginOfSpaces);

morseWord = parseToMorseLetters(&wordLength, word);

lengthOfWords = (int\*)realloc(lengthOfWords, sizeof(int) \* (length + 1));

lengthOfWords[length] = wordLength;

res = (char\*\*\*)realloc(res, sizeof(char\*\*) \* (length + 1));

res[length] = morseWord;

++length;

beginOfWord = -1;

}

beginOfSpaces = -1;

isTrackingSpaces = 0;

}

if (beginOfWord == -1)

beginOfWord = i;

}

else {

if (beginOfSpaces == -1)

beginOfSpaces = i;

isTrackingSpaces = 1;

}

++i;

}

if (beginOfWord != -1) {

int wordLength = 0;

char\* word = (char\*)malloc(sizeof(char) \* (i - beginOfSpaces));

substr(text, word, beginOfWord, i);

morseWord = parseToMorseLetters(&wordLength, word);

lengthOfWords = (int\*)realloc(lengthOfWords, sizeof(int) \* (length + 1));

lengthOfWords[length] = wordLength;

res = (char\*\*\*)realloc(res, sizeof(char\*\*) \* (length + 1));

res[length] = morseWord;

++length;

beginOfWord = -1;

}

\*length\_out = length;

\*lengthOfWords\_out = lengthOfWords;

return res;

}

char\*\* parseToMorseLetters(int\* length\_out, const char\* text) {

char\*\* res = NULL;

int length = 0;

int i = 0;

int begin = -1;

printf("text in ptml = %s\n", text);

while (text[i] != '\0') {

if (text[i] == '.' || text[i] == '-') {

if (begin == -1) {

begin = i;

}

}

else if (begin != -1) {

int memsize = (i - begin + 1) \* (sizeof(char));

char\* letter = (char\*)malloc(memsize);

substr(text, letter, begin, i);

++length;

res = (char\*\*)realloc(res, length \* sizeof(char\*));

res[length - 1] = letter;

begin = -1;

}

++i;

}

if (begin != -1) {

int memsize = (i - begin + 1) \* (sizeof(char));

char\* letter = (char\*)malloc(memsize);

substr(text, letter, begin, i);

++length;

res = (char\*\*)realloc(res, length \* sizeof(char\*));

res[length - 1] = letter;

begin = -1;

}

\*length\_out = length;

return res;

}

void addStr(char\* dst, const char\* newstr, int\* cursor) {

for (int c = 0; c < strlen(newstr); ++c, ++(\*cursor)) {

dst[\*cursor] = newstr[c];

}

}

Листинг 5 – Программный код translation.h

#ifndef TRANSLATION

#define TRANSLATION

#include "morseTable.h"

#include "parsers.h"

#include "stdio.h"

char\* toMorse(struct MorseCode\* table, char\* text);

char\* fromMorse(struct MorseCode\* table, char\* text);

#endif

Листинг 6 – Программный код translation.c

#include "translation.h"

char\* toMorse(struct MorseCode\* table, char\* text){

int length;

char\*\* words = parseToWords(&length, text);

int symbols = 0;

for(int i = 0; i < length; ++i)

symbols += strlen(words[i]);

int lengthOfArray = symbols \* 8 + (length - 1) \* 7 + 1;

char\* res = (char\*) malloc(sizeof(char) \* lengthOfArray);

int cursor = 0;

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

for(int j = 0; j < strlen(words[i]); ++j){

int s = 0;

while (table[s].letter != words[i][j]) ++s;

addStr(res, table[s].code, &cursor);

if(j == strlen(words[i]) - 1) continue;

const char\* gap = " ";

addStr(res, gap, &cursor);

}

if(i == length - 1) continue;

const char\* word\_gap = " ";

addStr(res, word\_gap, &cursor);

}

for(int i = 0; i < length; ++i)

free(words[i]);

free(words);

res[cursor] = '\0';

(int)strlen(res), cursor, lengthOfArray);

return res;

}

char\* fromMorse(struct MorseCode\* table, char\* text){

int length;

int\* lengthOfWords;

char\*\*\* words = parseToMorseWords(&length, &lengthOfWords, text);

printf("wordcount : %i\n", length);

char\* res = NULL;

int resLength = 0;

int i = 0;

while(i < length)

{

int j = 0;

while(j < lengthOfWords[i])

{

int choice = 0;

while(strcmp(table[choice].code, words[i][j]) != 0) ++choice;

res = (char\*)realloc(res, sizeof(char) \* (resLength + 2));

printf("After realloc\n");

res[resLength] = table[choice].letter;

res[resLength+1] = '\0';

++resLength;

++j;

}

res = (char\*)realloc(res, sizeof(char) \* (resLength + 2));

res[resLength] = ' ';

res[resLength+1] = '\0';

++resLength;

++i;

}

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

for(int j = 0; j < lengthOfWords[i]; ++j){

free(words[i][j]);

}

}

printf("After first free");

free(lengthOfWords);

return res;

}

Листинг 7 – Программный код Makefile

bin/program: bin/obj/main.o bin/obj/morseTable.o bin/obj/parsers.o bin/obj/translation.o

gcc -o bin/program bin/obj/main.o bin/obj/morseTable.o bin/obj/parsers.o bin/obj/translation.o

bin/obj/main.o: main.c

gcc -c -o bin/obj/main.o main.c

bin/obj/morseTable.o: morseTable.c

gcc -c -o bin/obj/morseTable.o morseTable.c

bin/obj/parsers.o: parsers.c

gcc -c -o bin/obj/parsers.o parsers.c

bin/obj/translation.o: translation.c

gcc -c -o bin/obj/translation.o translation.c