Федеральное агентство связи   
Федеральное бюджетное образовательное учреждение высшего образования   
«Сибирский государственный университет телекоммуникаций и информатики»   
(ФГОУ ВПО «СибГУТИ»)

Факультет ИВТ

Кафедра вычислительных систем

**Отчет по курсовой работе**

«Алгоритм Лемпела – Зива (Lempel – Ziv) LZ78**»**

Выполнил:

студент гр. ИВ - 621

Дьяченко Д. В.

Проверила:

Старший преподаватель Кафедры ВС

Перышкова Е.Н.

Новосибирск

2017

### Задание:

Реализовать программу lz78compress сжатия текстовых файлов на англий- ском языке алгоритмом Зива-Лемпела. Сжатие осуществляется с аргументом командной строки -c (compress), а распаковка – с аргументом -d (decompress). Опция -o указывает имя выходного файла. Вызов программы с аргументами командрой строки может выглядить следующим образом:

$ lz78compress -c -o file.lz78 file.txt # сжатие file.txt в file.lz78

$ lz78compress -d -o file1.txt file.lz78 # распаковка file.lz78 в file1.txt

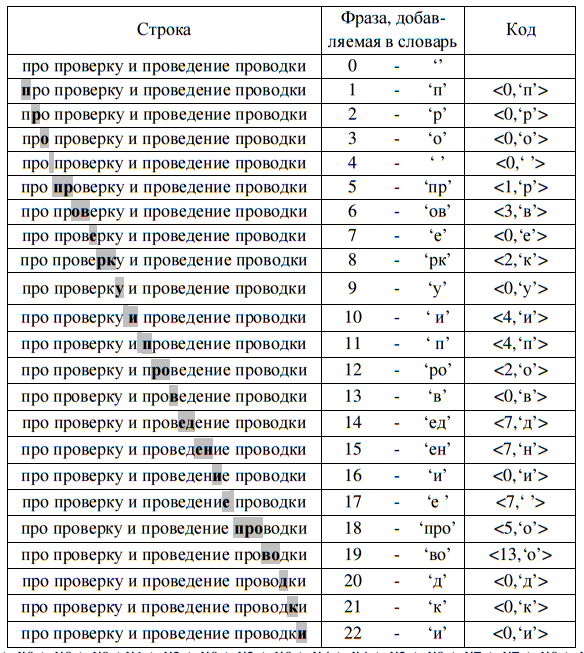
### **Критерии оценки** **:**

**Оценка «хорошо»**: реализован алгоритм сжатия, размер словаря 65536 элементов, при переполнении словаря он полностью сбрасывается.

**Оценка «отлично»**: реализован алгоритм сжатия, размер словаря может быть задан пользователем, при переполнении словаря удаляется наименее часто используемая фраза. Дополнительным плюсом будет использование кодов пе- ременной длины (размер ссылки на словарь сначала 1 байт, когда 1 байт пере- полняется – 2 байта и т.д.).

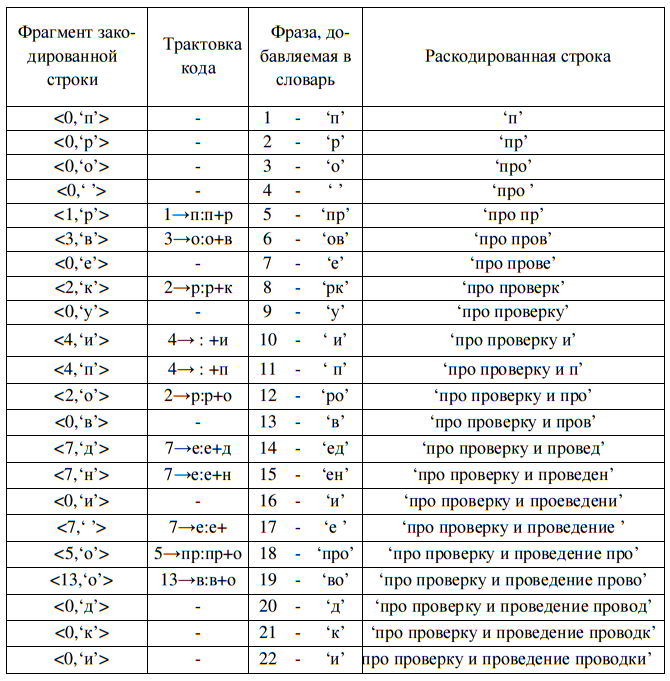
**Указания к выполнению задания**

В алгоритме LZ78 используется словарь из уже просмотренных фраз, ко- торый в процессе сжатия постоянно пополняется. При старте алгоритма этот словарь содержит только одну пустую строку (строку длины ноль). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока вхо- дит целиком в одну из фраз словаря. Как только эта строка перестанет соответ- ствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержа- ла входную строку, и символа, нарушившего совпадение. Затем в словарь до- бавляется полученная подстрока: X = <номер строки из словаря><символ, нарушивший совпадение>. Например, пусть мы кодируем следующий текст:”ппр”. Заносим в сло- варь первый символ ‘п’ и начинаем поиск новой фразы, для этого считываем следующий символ (‘п’). Поскольку он совпадает с фразой из словаря, считы- ваем очередной символ. Получившейся фразы ‘пр’ нет в словаре – в соответст- вии с алгоритмом генерируем код для нее и записываем ее в словарь. 49 Если словарь уже заполнен, то из него предварительно удаляют наиболее редко используемую фразу либо очищают полностью. Ключевым для размера получаемых кодов является размер словаря во фразах, потому что каждый код при кодировании по методу LZ78 содержит номер фразы в словаре. Из последнего следует, что эти коды имеют постоян- ную длину, равную округленному в большую сторону двоичному логарифму размера словаря плюс количество бит в байт-коде ASCII. Рассмотрим алгоритм на примере. Пусть нам необходимо закодировать стро- ку: «про проверку и проведение проводки». Пошаговый алгоритм кодирования представлен на рисунке 26.



0,‘п’|0,‘р’|0,‘о’|0,‘ ’|1,‘р’|3,‘в’|0,‘е’|2,‘к’|0,‘у’|4,‘и’|4,‘п’|2,‘о’|0,‘в’|7,‘д’|7,‘н’|0,‘и’|7,‘ ’| 5,‘о’|13,‘о’|0,‘д’|0,‘и’

При декодировании данной строки словарь формируется динамически. Из входных данных считываются пары <номер фразы-начала из словаря, завер- шающий символ>, в выходные данные дописывается конкатенация этих двух элементов. Так например, пусть в качестве входных данных получен набор ко- дов: <0,‘п’><1,‘р’>. Считываем первый код - <0,‘п’>, поскольку номер фразы- начала из словаря 0, что соответствует пустой строке, после конкатенации в словарь под номером 1 добавляется фраза ‘п’. Считываем очередной код-<1,‘р’>. В нем номер фразы-начала из словаря равен 1, что соответствует фразе ‘п’, в словарь под номером 2 мы добавляем конкатенацию этой фразы и символа ‘р’ – фразу ‘пр’



**Анализ задачи**

Была реализована программа на оценку «хорошо». В качестве хранения кодов использовался массив структур Code, а для словаря структура Dictionary, которая хранила размер (size) и объем (capacity) словаря и массив строк. Также были реализованы алгоритмы кодирования/декодирования и записи/чтения массива кодов для большего сжатия данных. То есть, коды получились переменной длины. Также для работы со строками были реализованы функции scat (конкатенация строк), slen (размер строки), sspn (вхождение подстроки в строку), scpy (копирование строки в строку. Размер словаря (65536) зашит в программу, а при переполнении очищается полностью.

Структура репозитория:

.

├── bin

│   └── lz78

├── build

│   ├── coder.o

│   ├── lz78.o

│   ├── main.o

│   └── strings.o

├── dec.txt

├── Makefile

├── out.lz78

├── README.md

├── src

│   ├── coder.c

│   ├── coder.h

│   ├── lz78.c

│   ├── lz78.h

│   ├── main.c

│   ├── strings.c

│   └── strings.h

└── txt

├── eng.txt

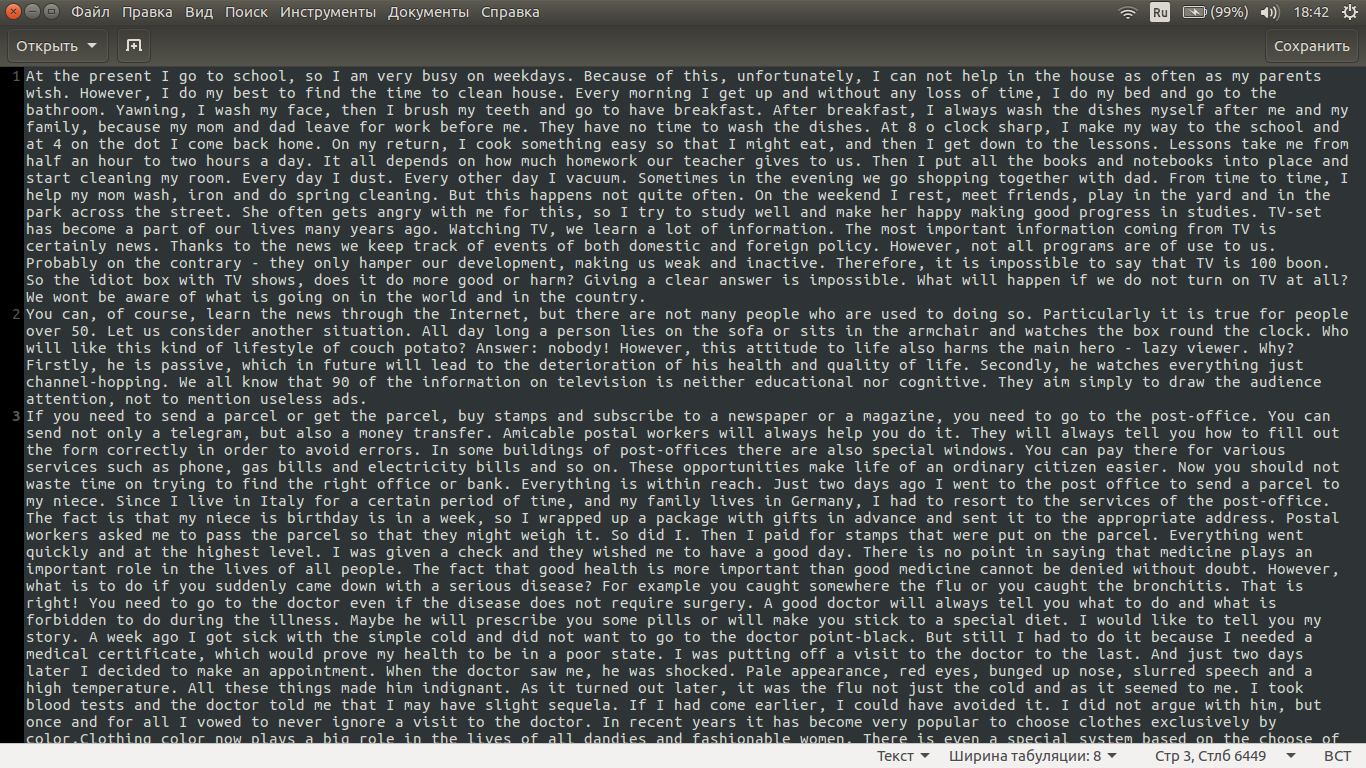
└── tmp.txt

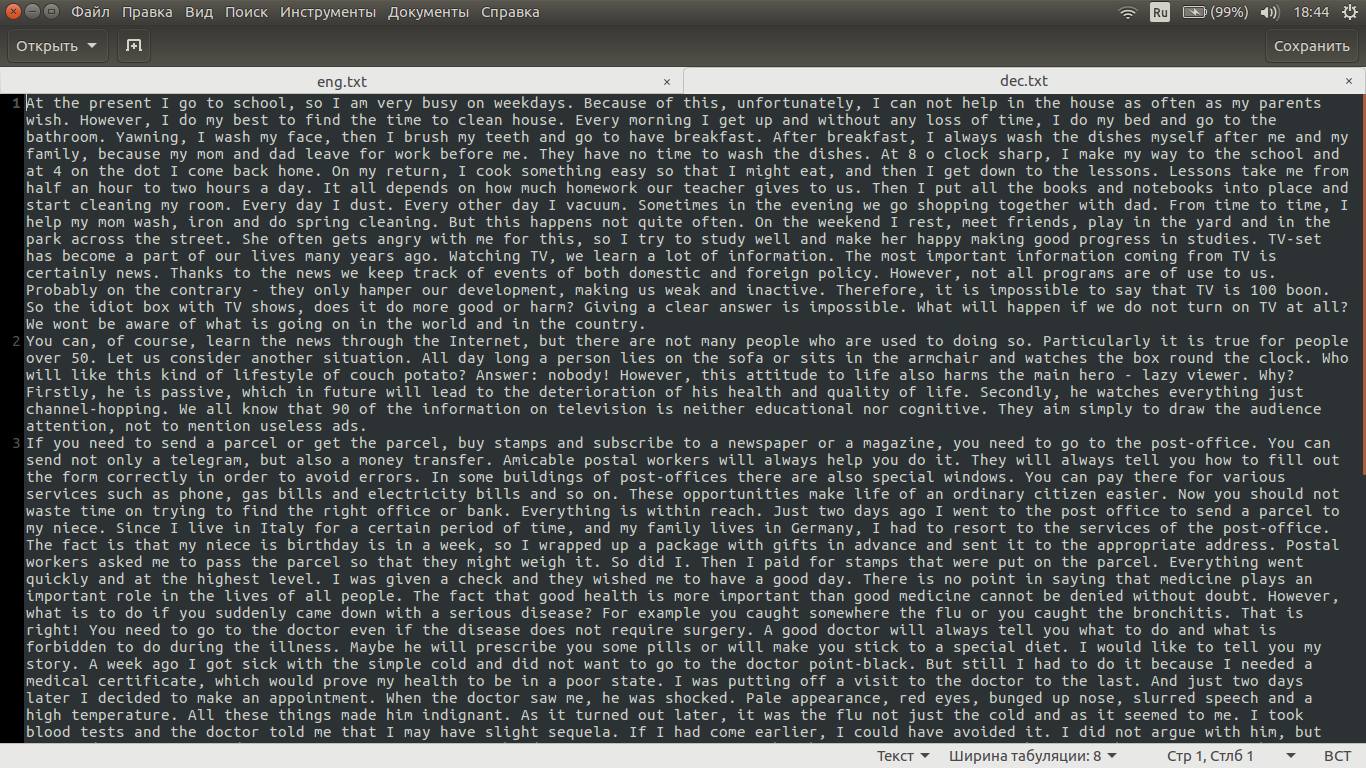
4 directories, 18 files

**Тестовые данные.**

В качестве тестовых данных был взят текст с латинскими буквами и различными служебными символами («!», «?», «,», «.» и тд) размером в общей сложности 6449 символов. Файл, содержащий данный текст, лежит в репозитории в директории «txt» вместе с исходным кодом программы.

*Исходные данные*:

****

*Данные после работы программы:*

*Проверка на корректность входных данных:*

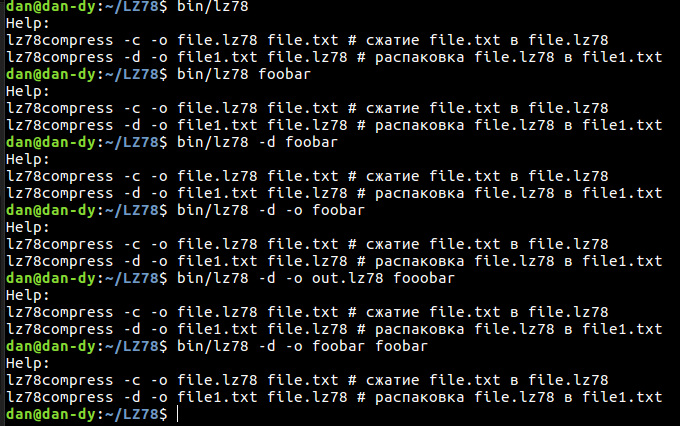
*-компрессия:*

**

*-декомпрессия:*

**

*-неправильные входные данные:*

**

**Листинг программы**

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include "lz78.h"

#include "coder.h"

#include "strings.h"

void help()

{

printf("Help:\n");

printf("lz78compress -c -o file.lz78 file.txt # сжатие file.txt в file.lz78 \n");

printf("lz78compress -d -o file1.txt file.lz78 # распаковка file.lz78 в file1.txt \n");

}

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

{

int dic\_capacity = 65536;

if (argv[1] == NULL || argv[2] == NULL || argv[3] == NULL || argv[4] == NULL) {

help();

return 1;

}

/\*--------------------------COMPRES------------------------------\*/

if (!scmp(argv[1], "-c") && !scmp(argv[2], "-o")) {

FILE \*in = fopen(argv[4], "r");

if (!in) {

help();

return 1;

}

Dictionary dic;

dic\_init(&dic, dic\_capacity);

Code \*code = code\_init(dic.capacity);

compres(&dic, code, in);

fclose(in);

FILE \*out = fopen(argv[3], "w");

encode\_file(out, code, dic);

fclose(out);

printf("Complete!\n");

return 0;

}

/\*--------------------------DECOMPRES---------------------------\*/

if (!scmp(argv[1], "-d") && !scmp(argv[2], "-o")) {

FILE \*in = fopen(argv[4], "r");

if (!in) {

help();

return 1;

}

Dictionary dic;

dic\_init(&dic, dic\_capacity);

Code \*codes = code\_init(dic\_capacity);

decode\_file(in, codes);

fill\_dic(&dic, codes);

fclose(in);

FILE \*out = fopen(argv[3], "w");

write\_to\_file\_decode(out, codes, dic);

fclose(out);

printf("Complete!\n");

return 0;

}

/\*---------------------------------------------------------------\*/

return 0;

}

**coder.h**

#ifndef CODER\_H

#define CODER\_H

#include <inttypes.h>

#include "lz78.h"

enum {

MaxCodeLength = 4

};

typedef struct {

uint8\_t code[MaxCodeLength];

size\_t length;

} CodeUnit;

int encode(uint32\_t code\_point, CodeUnit \*code\_units);

void encode\_file(FILE \*out, Code \*code, Dictionary dic);

uint32\_t decode(const CodeUnit \*code\_unit);

int decode\_file(FILE \*in, Code \*code);

int read\_next\_code\_unit(FILE \*in, CodeUnit \*code\_units);

int write\_code\_unit(FILE \*out, const CodeUnit \*code\_unit);

void write\_to\_file\_decode(FILE \*out, Code \*code, Dictionary dic);

#endif

**coder.c**

#include <stdio.h>

#include <inttypes.h>

#include <math.h>

#include "coder.h"

#include "lz78.h"

int encode(uint32\_t code\_point, CodeUnit \*code\_units)

{

uint8\_t count = 0;

for (uint32\_t i = code\_point; i > 0; i >>= 1) {

count++;

}

if (count <= 7) {

code\_units->code[0] = code\_point;

code\_units->length = 1;

return 0;

} else if (count <= 11) {

code\_units->code[0] = 0xC0 | (code\_point >> 6);

code\_units->code[1] = 0x80 | (code\_point & 0x3F);

code\_units->length = 2;

return 0;

} else if (count <= 16) {

code\_units->code[0] = 0xE0 | (code\_point >> 12);

code\_units->code[1] = 0x80 | ((code\_point & 0xFC0) >> 6);

code\_units->code[2] = 0x80 | (code\_point & 0x3F);

code\_units->length = 3;

return 0;

} else if (count <= 21) {

code\_units->code[0] = 0xF0 | (code\_point >> 18);

code\_units->code[1] = 0x80 | ((code\_point & 0x3F000) >> 12);

code\_units->code[2] = 0x80 | ((code\_point & 0xFC0) >> 6);

code\_units->code[3] = 0x80 | (code\_point & 0x3F);

code\_units->length = 4;

return 0;

}

return -1;

}

void encode\_file(FILE \*out, Code \*code, Dictionary dic)

{

for (int i = 1; i < dic.size; i++) {

CodeUnit code\_unit;

uint32\_t code\_point;

code\_point = code[i].num;

encode(code\_point, &code\_unit);

write\_code\_unit(out, &code\_unit);

code\_point = code[i].str;

encode(code\_point, &code\_unit);

write\_code\_unit(out, &code\_unit);

}

}

uint32\_t decode(const CodeUnit \*code\_unit)

{

uint32\_t code\_point;

if ((code\_unit->code[0] >> 7) == 0) {

return (code\_point = code\_unit->code[0]);

} else if (code\_unit->code[0] <= 0xDF) {

return (code\_point = (((code\_unit->code[0] & 0x1F) << 6) | (code\_unit->code[1] & 0x3F)));

} else if (code\_unit->code[0] <= 0xEF) {

return (((code\_unit->code[0] & 0xF) << 12) | ((code\_unit->code[1] & 0x3F) << 6) | (code\_unit->code[2] & 0x3F));

} else if (code\_unit->code[0] <= 0xF7) {

return (((code\_unit->code[0] & 0x7) << 18) | ((code\_unit->code[1] & 0x3F) << 12) | ((code\_unit->code[2] & 0x3F) << 6) | (code\_unit->code[3] & 0x3F));

}

return 0;

}

int decode\_file(FILE \*in, Code \*code)

{

CodeUnit code\_units;

uint8\_t buf;

fread(&buf, 1, 1, in);

int i;

for (i = 1; !feof(in); i++) {

uint8\_t enum\_bit = 0;

while(buf & (1 << (7 - enum\_bit))) {

enum\_bit++;

}

if (enum\_bit == 1) {

fread(&buf, 1, 1, in);

continue;

}

if (enum\_bit == 0) {

enum\_bit = 1;

}

if (enum\_bit <= MaxCodeLength) {

code\_units.length = 0;

for (int i = 1; i <= enum\_bit; i++) {

code\_units.code[i - 1] = buf;

code\_units.length++;

if (i == enum\_bit) {

break;

}

fread(&buf, 1, 1, in);

if (feof(in)) {

break;

}

if ((buf & 0xC0) != 0x80) {

break;

}

}

}

uint32\_t tmp = decode(&code\_units);

code[i].num = tmp;

fread(&buf, 1, 1, in);

code[i].str = buf;

fread(&buf, 1, 1, in);

//printf("%d\n", i);

}

return 0;

}

void write\_to\_file\_decode(FILE \*out, Code \*code, Dictionary dic)

{

for (int i = 1; i < dic.size; i++) {

if (code[i].num == 0) {

//printf("1 :=: %c\n", code[i].str);

fprintf(out, "%c", code[i].str);

} else {

//printf("2 :=: %s.%c.%d\n", dic.dic\_i[code[i].num].str, code[i].str, code[i].num);

fprintf(out, "%s", dic.dic\_i[code[i].num].str);

fprintf(out, "%c", code[i].str);

}

}

}

int write\_code\_unit(FILE \*out, const CodeUnit \*code\_unit)

{

if (fwrite(code\_unit->code, 1, code\_unit->length, out) == code\_unit->length) {

return 0;

}

return -1;

}

**lz78.h**

#ifndef LZ78\_H

#define LZ78\_H

#include <stdio.h>

typedef struct {

int num;

char str;

} Code;

typedef struct {

char \*str;

//int count;

} Dictionary\_item;

typedef struct {

Dictionary\_item \*dic\_i;

int size;

int capacity;

} Dictionary;

void dic\_init(Dictionary \*dic, int capacity);

char \*swap\_str(char \*str);

void fill\_dic(Dictionary \*dic, Code \*code);

Code \*code\_init(int size);

int find\_i(Dictionary dic, char \*tmp);

void free\_all(char \*tmp\_dic, Dictionary dic);

void add\_code(Code \*code, Dictionary dic);

void compres(Dictionary \*dic, Code \*code, FILE \*in);

void clear\_dic(Dictionary \*dic);

#endif

**lz78.c**

#include "strings.h"

#include "lz78.h"

#include <stdlib.h>

void dic\_init(Dictionary \*dic, int capacity)

{

dic->capacity = capacity;

dic->size = 1;

dic->dic\_i = calloc(dic->capacity, sizeof(Dictionary\_item));

dic->dic\_i[0].str = calloc(1, sizeof(char));

}

char \*swap\_str(char \*str)

{

char \*tmp = malloc(sizeof(char) \* slen(str));

scpy(tmp, str);

for (int i = 0; i < slen(str) / 2; i++) {

char buf = tmp[i];

tmp[i] = tmp[slen(str) - 1 - i];

tmp[slen(str) - 1 - i] = buf;

}

return tmp;

}

void fill\_dic(Dictionary \*dic, Code \*code)

{

for (int i = 1; code[i].str != 0; i++) {

if (i >= dic->capacity) {

clear\_dic(dic);

}

if (code[i].num == 0) {

dic->dic\_i[i].str = calloc(2, sizeof(char));

char ai[2];

ai[0] = code[i].str;

ai[1] = 0;

scat(dic->dic\_i[i].str, ai);

}

if (code[i].num != 0) {

char \*buf\_str = calloc(256, sizeof(char));

char tmp\_code[2];

tmp\_code[0] = code[i].str;

tmp\_code[1] = 0;

int k = code[i].num;

int count = 0;

while (k != 0) {

count++;

char s[2];

s[0] = code[k].str;

s[1] = 0;

scat(buf\_str, s);

k = code[k].num;

}

scat(buf\_str, tmp\_code);

if (count >= 2) {

char \*mediator = calloc(slen(buf\_str) + 1, sizeof(char));

scat(mediator, buf\_str);

char t[2];

t[0] = mediator[slen(buf\_str) - 1];

t[1] = '\0';

mediator[slen(buf\_str) - 1] = '\0';

buf\_str = swap\_str(mediator);

scat(buf\_str, t);

free(mediator);

}

dic->dic\_i[i].str = calloc(slen(buf\_str) + 1, sizeof(char));

scat(dic->dic\_i[i].str, buf\_str);

free(buf\_str);

}

dic->size++;

}

}

Code \*code\_init(int size)

{

Code \*code = calloc(size, sizeof(Code));

code[0].num = 0;

code[0].str = '\0';

return code;

}

int find\_i(Dictionary dic, char \*tmp)

{

if (!scmp("\n", tmp)) {

return 1;

}

for (int i = 0; i < dic.size; i++) {

int len = sspn(dic.dic\_i[i].str, tmp);

if (len == slen(tmp)) {

return 0;

}

}

return -1;

}

void free\_all(char \*tmp\_dic, Dictionary dic)

{

for (int i = 1; i < dic.capacity; i++) {

free(dic.dic\_i[i].str);

}

free(dic.dic\_i);

free(tmp\_dic);

}

void add\_code(Code \*code, Dictionary dic)

{

if (slen(dic.dic\_i[dic.size].str) == 1) {

code[dic.size].num = 0;

code[dic.size].str = dic.dic\_i[dic.size].str[0];

return;

}

code[dic.size].str = dic.dic\_i[dic.size].str[slen(dic.dic\_i[dic.size].str) - 1];

char \*first\_substr = calloc(slen(dic.dic\_i[dic.size].str) + 1, sizeof(char));

scat(first\_substr, dic.dic\_i[dic.size].str);

first\_substr[slen(dic.dic\_i[dic.size].str) - 1] = '\0';

for (int i = 0; i < dic.size; i++) {

int len = sspn(dic.dic\_i[i].str, first\_substr);

if (len == slen(first\_substr)) {

code[dic.size].num = i;

break;

}

}

free(first\_substr);

}

void compres(Dictionary \*dic, Code \*code, FILE \*in)

{

char tmp[2];

char \*tmp\_dic = calloc(256, sizeof(char));

while (fread(&tmp, 1, 1, in)) {

tmp[1] = 0;

scat(tmp\_dic, tmp);

int tmp\_i = find\_i(\*dic, tmp\_dic);

if (tmp\_i == 1) {

break;

} else if (!tmp\_i) {

continue;

} else {

if (dic->size == dic->capacity) {

clear\_dic(dic);

//clear\_dic\_with\_min\_usige\_elem(dic);

//remove(dic);

}

dic->dic\_i[dic->size].str = calloc(slen(tmp\_dic) + 1, sizeof(char));

scat(dic->dic\_i[dic->size].str, tmp\_dic);

add\_code(code, \*dic);

dic->size++;

\*tmp\_dic = '\0';

}

}

free(tmp\_dic);

}

void clear\_dic(Dictionary \*dic)

{

for (int i = 1; i < dic->capacity; i++) {

free(dic->dic\_i[i].str);

dic->dic\_i[i].str = NULL;

}

dic->size = 1;

}

**strings.h**

#ifndef STRINGS\_H

#define STRINGS\_H

int slen(char \*str);

int sspn(char \*str, const char \*substr);

int scmp(char \*mstr, const char \*cpstr);

char \*scat(char \*des, char \*src);

char \*scpy(char \*des, const char \*src);

#endif

**strings.c**

#include <stdio.h>

#include <stdlib.h>

#include "strings.h"

#include <stdint.h>

#include <string.h>

int slen(char \*str)

{

int count = 0;

for (int i = 0; str[i] != '\0'; i++) {

count++;

}

return count;

}

int sspn(char \*str, const char \*substr)

{

int count = 0;

for (int i = 0; str[i]; i++) {

if (str[i] == substr[i]) {

count++;

} else {

return count;

}

}

return count;

}

int scmp(char \*str, const char \*strc)

{

int ncount = 0, pcount = 0;

for (int i = 0; str[i] != 0; i++) {

if (str[i] < strc[i]) {

ncount++;

} else if (str[i] > strc[i]) {

pcount++;

}

}

if (ncount < pcount) {

return pcount;

} else if (ncount > pcount) {

return -ncount;

}

return 0;

}

char \*scat(char \*des, char \*src)

{

int j = slen(des);

for (int i = 0; src[i]; i++) {

des[j] = src[i];

j++;

}

des[j] = 0;

return des;

}

char \*scpy(char \*des, const char \*src)

{

int i;

for (i = 0; src[i] != 0; i++) {

des[i] = src[i];

}

des[i] = 0;

return des;

}