**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Схема структурная

**ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема алгоритма

**ПРИЛОЖЕНИЕ В**

*(обязательное)*

Листинг кода

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string>

#include <iostream>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <conio.h>

#include <clocale>

typedef unsigned char uchar;

typedef unsigned long ulong;

using namespace std;

typedef struct bfile

{

FILE \*file;

uchar mask;

int rack;

int pacifier\_counter;

}

BFILE;

#define PACIFIER\_COUNT 2047

BFILE \*OpenInputBFile(char \*name);

BFILE \*OpenOutputBFile(char \*name);

void WriteBit(BFILE \*bfile, int bit);

void WriteBits(BFILE \*bfile, ulong code, int count);

int ReadBit(BFILE \*bfile);

ulong ReadBits(BFILE \*bfile, int bit\_count);

void CloseInputBFile(BFILE \*bfile);

void CloseOutputBFile(BFILE \*bfile);

void CompressFile(FILE \*input, BFILE \*output);

void ExpandFile(BFILE \*input, FILE \*output);

void usage\_exit(char \*prog\_name);

void print\_ratios(char \*input, char \*output);

long file\_size(char \*name);

void InitTree(int r);

void ContractNode(int old\_node, int new\_node);

void ReplaceNode(int old\_node, int new\_node);

int FindNextNode(int node);

void DeleteString(int p);

int AddString(int new\_node, int \*match\_position);

#define INDEX\_BITS 12

#define LENGTH\_BITS 4

#define WINDOW\_SIZE ( 1 << INDEX\_BITS)

#define RAW\_LOOK\_AHEAD\_SIZE ( 1 << LENGTH\_BITS)

#define BREAK\_EVEN (( 1 + INDEX\_BITS + LENGTH\_BITS) / 9)

#define LOOK\_AHEAD\_SIZE ( RAW\_LOOK\_AHEAD\_SIZE + BREAK\_EVEN)

#define TREE\_ROOT WINDOW\_SIZE

#define END\_OF\_STREAM 0

#define UNUSED 0

#define MODULO(a) ( (a) & (WINDOW\_SIZE - 1) )

char \*CompressionName = "Кодер LZSS";

char \*Usage = "входной\_файл выходной\_файл\n\n";

void fatal\_error(char \*str, ...)

{

printf("Ошибка: %s\n", str);

exit(1);

}

BFILE \*OpenOutputBFILE(char \*name)

{

BFILE \*bfile;

bfile = (BFILE \*)calloc(1, sizeof (BFILE));

bfile->file = fopen(name, "wb");

bfile->rack = 0;

bfile->mask = 0x80;

bfile->pacifier\_counter = 0;

return bfile;

}

BFILE \*OpenInputBFile(char \*name)

{

BFILE \*bfile;

bfile = (BFILE \*)calloc(1, sizeof (BFILE));

bfile->file = fopen(name, "rb");

bfile->rack = 0;

bfile->mask = 0x80;

bfile->pacifier\_counter = 0;

return bfile;

}

void CloseOutputBFile(BFILE \*bfile)

{

if (bfile->mask != 0x80)

putc(bfile->rack, bfile->file);

fclose(bfile->file);

free((char \*)bfile);

}

void CloseInputBFile(BFILE \*bfile)

{

fclose(bfile->file);

free((char \*)bfile);

}

void WriteBit(BFILE \*bfile, int bit)

{

if (bit)

bfile->rack |= bfile->mask;

bfile->mask >>= 1;

if (bfile->mask == 0)

{

putc(bfile->rack, bfile->file);

if ((bfile->pacifier\_counter++ & PACIFIER\_COUNT) == 0)

putc('.', stdout);

bfile->rack = 0;

bfile->mask = 0x80;

}

}

void WriteBits(BFILE \*bfile, ulong code, int count)

{

ulong mask;

mask = 1L << (count - 1);

while (mask != 0)

{

if (mask & code)

bfile->rack |= bfile->mask;

bfile->mask >>= 1;

if (bfile->mask == 0)

{

putc(bfile->rack, bfile->file);

if ((bfile->pacifier\_counter++ & PACIFIER\_COUNT) == 0)

putc('.', stdout);

bfile->rack = 0;

bfile->mask = 0x80;

}

mask >>= 1;

}

}

int ReadBit(BFILE \*bfile)

{

int value;

if (bfile->mask == 0x80)

{

bfile->rack = getc(bfile->file);

if (bfile->rack == EOF)

fatal\_error("Ошибка в процедуре ReadBit!\n");

if ((bfile->pacifier\_counter++ & PACIFIER\_COUNT) == 0)

putc('.', stdout);

}

value = bfile->rack & bfile->mask;

bfile->mask >>= 1;

if (bfile->mask == 0)

bfile->mask = 0x80;

return (value ? 1 : 0);

}

ulong ReadBits(BFILE \*bfile, int bit\_count)

{

ulong mask;

ulong return\_value;

mask = 1L << (bit\_count - 1);

return\_value = 0;

while (mask != 0)

{

if (bfile->mask == 0x80)

{

bfile->rack = getc(bfile->file);

if (bfile->rack == EOF)

fatal\_error("Ошибка в процедуре ReadBits!\n");

if ((bfile->pacifier\_counter++ & PACIFIER\_COUNT) == 0)

putc('.', stdout);

}

if (bfile->rack & bfile->mask)

return\_value |= mask;

mask >>= 1;

bfile->mask >>= 1;

if (bfile->mask == 0)

bfile->mask = 0x80;

}

return return\_value;

}

void MainCompressFunc()

{

char InputPath[1000];

char OutputPath[1000];

int add = 0;

BFILE \*output;

FILE \*input;

printf("Введите адрес файла:\n");

fflush(stdin);

cin >> InputPath;

printf("Введите адрес архива\n");

fflush(stdin);

cin >> OutputPath;

input = fopen(InputPath, "rb");

if (input == NULL)

fatal\_error("Ошибка при открытии файла для ввода\n");

output = OpenOutputBFILE(OutputPath);

if (output == NULL)

fatal\_error("Ошибка при открытии файла для вывода\n");

printf("\nСжимаю %s в %s\n", InputPath, OutputPath);

printf("Используя %s\n", CompressionName);

CompressFile(input, output);

CloseOutputBFile(output);

fclose(input);

print\_ratios(InputPath, OutputPath);

}

void MainExpandFunc()

{

char InputPath[1000];

char OutputPath[1000];

int add = 0;

FILE \*output;

BFILE \*input;

printf("Введите адрес архива\n");

fflush(stdin);

cin >> InputPath;

printf("Введите адрес выходного файла\n");

cin >> OutputPath;

input = OpenInputBFile(InputPath);

if (input == NULL)

fatal\_error("Ошибка при открытии файла для ввода\n");

output = fopen(OutputPath, "wb");

if (output == NULL)

fatal\_error("Ошибка при открытии файла для вывода\n");

printf("\nРаспаковываю %s в %s\n", InputPath, OutputPath);

printf("Используя %s\n", CompressionName);

ExpandFile(input, output);

CloseOutputBFile(input);

fclose(output);

}

void About()

{

system("cls");

printf("Это курсовой проект студента гр.830501\nГлинки Андрея Владимировича\nАрхиватор файлов Windows кодировкой LZSS\n");

\_getch();

system("cls");

}

int main()

{

char choice;

setlocale(LC\_ALL, "Russian");

setbuf(stdout, NULL);

setlocale(LC\_ALL, "Russian");

do

{

printf("\n1-Новый архив \n2-Распаковка \n3-О программе \n4-Выход\n");

fflush(stdin);

choice = getchar();

switch (choice)

{

case '1':

MainCompressFunc();

break;

case '2':

MainExpandFunc();

break;

case '3':

About();

break;

case '4':

exit(0);

default:

break;

}

} while (true);

return 0;

}

void usage\_exit(char \*prog\_name)

{

char \*short\_name;

char \*extension;

short\_name = strrchr(prog\_name, '\\');

if (short\_name == NULL)

short\_name = strrchr(prog\_name, '/');

if (short\_name == NULL)

short\_name = strrchr(prog\_name, ':');

if (short\_name != NULL)

short\_name++;

else

short\_name = prog\_name;

extension = strrchr(short\_name, '.');

if (extension != NULL)

\*extension = '\0';

printf("\nПравильное использование: %s %s\n",

short\_name, Usage);

exit(0);

}

long file\_size(char \*name)

{

long eof\_ftell;

FILE \*file;

file = fopen(name, "r");

if (file == NULL)

return (0L);

fseek(file, 0L, SEEK\_END);

eof\_ftell = ftell(file);

fclose(file);

return eof\_ftell;

}

void print\_ratios(char \*input, char \*output)

{

long input\_size;

long output\_size;

int ratio;

input\_size = file\_size(input);

if (input\_size == 0)

input\_size = 1;

output\_size = file\_size(output);

ratio = (int)(output\_size \* 100L / input\_size);

printf("\nРазмер входного файла (байт): %ld\n",

input\_size);

printf("Размер выходного файла (байт): %ld\n",

output\_size);

if (output\_size == 0)

output\_size = 1;

printf("Степень сжатия: %d%%\n", ratio);

}

uchar window[WINDOW\_SIZE];

struct

{

int parent;

int smaller\_child;

int larger\_child;

}

tree[WINDOW\_SIZE + 1];

void InitTree(int r)

{

tree[TREE\_ROOT].larger\_child = r;

tree[r].parent = TREE\_ROOT;

tree[r].larger\_child = UNUSED;

tree[r].smaller\_child = UNUSED;

}

void ContractNode(int old\_node, int new\_node)

{

tree[new\_node].parent = tree[old\_node].parent;

if (tree[tree[old\_node].parent].larger\_child == old\_node)

tree[tree[old\_node].parent].larger\_child = new\_node;

else

tree[tree[old\_node].parent].smaller\_child = new\_node;

tree[old\_node].parent = UNUSED;

}

void ReplaceNode(int old\_node, int new\_node)

{

int parent;

parent = tree[old\_node].parent;

if (tree[parent].smaller\_child == old\_node)

tree[parent].smaller\_child = new\_node;

else

tree[parent].larger\_child = new\_node;

tree[new\_node] = tree[old\_node];

tree[tree[new\_node].smaller\_child].parent = new\_node;

tree[tree[new\_node].larger\_child].parent = new\_node;

tree[old\_node].parent = UNUSED;

}

int FindNextNode(int node)

{

int next;

next = tree[node].smaller\_child;

while (tree[next].larger\_child != UNUSED)

next = tree[next].larger\_child;

return next;

}

void DeleteString(int p)

{

int replacement;

if (tree[p].parent == UNUSED)

return;

if (tree[p].larger\_child == UNUSED)

ContractNode(p, tree[p].smaller\_child);

else

if (tree[p].smaller\_child == UNUSED)

ContractNode(p, tree[p].larger\_child);

else

{

replacement = FindNextNode(p);

DeleteString(replacement);

ReplaceNode(p, replacement);

}

}

int AddString(int new\_node, int \*match\_pos)

{

int i;

int test\_node;

int delta;

int match\_len;

int \*child;

if (new\_node == END\_OF\_STREAM)

return (0);

test\_node = tree[TREE\_ROOT].larger\_child;

match\_len = 0;

for (;;)

{

for (i = 0; i < LOOK\_AHEAD\_SIZE; i++)

{

delta = window[MODULO(new\_node + i)] -

window[MODULO(test\_node + i)];

if (delta != 0)

break;

}

if (i >= match\_len)

{

match\_len = i;

\*match\_pos = test\_node;

if (match\_len >= LOOK\_AHEAD\_SIZE)

{

ReplaceNode(test\_node, new\_node);

return match\_len;

}

}

if (delta >= 0)

child = &tree[test\_node].larger\_child;

else

child = &tree[test\_node].smaller\_child;

if (\*child == UNUSED)

{

\*child = new\_node;

tree[new\_node].parent = test\_node;

tree[new\_node].larger\_child = UNUSED;

tree[new\_node].smaller\_child = UNUSED;

return match\_len;

}

test\_node = \*child;

}

}

void CompressFile(FILE \*input, BFILE \*output)

{

int i;

int c;

int look\_ahead\_bytes;

int current\_pos;

int replace\_count;

int match\_len;

int match\_pos;

current\_pos = 1;

for (i = 0; i < LOOK\_AHEAD\_SIZE; i++)

{

if ((c = getc(input)) == EOF)

break;

window[current\_pos + i] = (uchar)c;

}

look\_ahead\_bytes = i;

InitTree(current\_pos);

match\_len = 0;

match\_pos = 0;

while (look\_ahead\_bytes > 0)

{

if (match\_len > look\_ahead\_bytes)

match\_len = look\_ahead\_bytes;

if (match\_len <= BREAK\_EVEN)

{

replace\_count = 1;

WriteBit(output, 1);

WriteBits(output, (ulong)window[current\_pos], 8);

}

else

{

WriteBit(output, 0);

WriteBits(output, (ulong)match\_pos, INDEX\_BITS);

WriteBits(output, (ulong)(match\_len - (BREAK\_EVEN + 1)),

LENGTH\_BITS);

replace\_count = match\_len;

}

for (i = 0; i < replace\_count; i++)

{

DeleteString(MODULO(current\_pos + LOOK\_AHEAD\_SIZE));

if ((c = getc(input)) == EOF)

look\_ahead\_bytes--;

else

window[MODULO(current\_pos + LOOK\_AHEAD\_SIZE)] = (uchar)c;

current\_pos = MODULO(current\_pos + 1);

if (look\_ahead\_bytes)

match\_len = AddString(current\_pos, &match\_pos);

}

}

WriteBit(output, 0);

WriteBits(output, (ulong)END\_OF\_STREAM, INDEX\_BITS);

}

void ExpandFile(BFILE \*input, FILE \*output)

{

int i;

int current\_pos;

int c;

int match\_len;

int match\_pos;

current\_pos = 1;

for (;;)

{

if (ReadBit(input))

{

c = (int)ReadBits(input, 8);

putc(c, output);

window[current\_pos] = (uchar)c;

current\_pos = MODULO(current\_pos + 1);

}

else

{

match\_pos = (int)ReadBits(input, INDEX\_BITS);

if (match\_pos == END\_OF\_STREAM)

break;

match\_len = (int)ReadBits(input, LENGTH\_BITS);

match\_len += BREAK\_EVEN;

for (i = 0; i <= match\_len; i++)

{

c = window[MODULO(match\_pos + i)];

putc(c, output);

window[current\_pos] = (uchar)c;

current\_pos = MODULO(current\_pos + 1);

}

}

}

}

**ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Ведомость документов