Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

Лабораторная работа № 2
«Текстовые файлы»

Проверил:
асс. каф. ЭВС
И.Г. Скиба

Выполнил:
ст. гр. 250504
П.А. Щербо

МИНСК 2023

Цель работы – освоить принципы работы с текстовыми файлами в языке С.

Условие:

Написать программу сжатия текстового файла по алгоритму:

1. Подсчет частоты встречи каждого слова в файле.
2. Поиск самого популярного среди длинных слов (А).
3. Поиск самого непопулярного среди коротких слов (В).
4. Замена всех слов А и В друг на друга.
5. Повтор пунктов 2-4 до тех пор, пока это имеет смысл.

- Программа должна сжать и разжать текстовый файл по приведенному выше алгоритму.

- В результате работы должно быть создано два файла: сжатый и разжатый. Последний должен полностью совпадать с исходным. Сжатый файл должен быть меньше исходного.

- При выборе слова А и В понятия «длинное слово» и «короткое слово» можно трактовать гибко. То есть, «длинное слово» не обязательно «самое длинное слово», главное, чтобы его использование имело положительный эффект.

- Сжатие и разжатие файла реализовать двумя разными программами (два отдельных проекта).

- Следовательно, в сжатый файл должны быть упакованы данные, необходимые для точного разжатия.

- Для реализации пункта 1 алгоритма использовать собственный связный список (стек или очередь).

Код программы:

```
void compressor(void) {
    FILE* file = NULL;
    char* word = NULL;
    char* new_word = NULL;
    char* buf = (char*)calloc(4096, 1);
    int index = 0, start = 0;
    file = fopen("/Users/pavelshcherbo/Desktop/compress/Zip.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    fgets(buf, 4096, file);
    while (!feof(file)) {
        while ((start = find_word(buf, &index)) != -1) {
            word = take_word(buf, start);
            new_word = word_from_glossary(word);
            insert_from_glossary(&buf, word, new_word, &index);
            index++;
        }
        puts_file(buf);
        index = 0;
        fgets(buf, 4096, file);
    }
    fclose(file);
```

```
    free(buf);
    free(word);
    free(new_word);
}
```

Блок схемы программы (рис.1, рис.2):

Рисунок 1 - Блок схема для выполнения задания №1

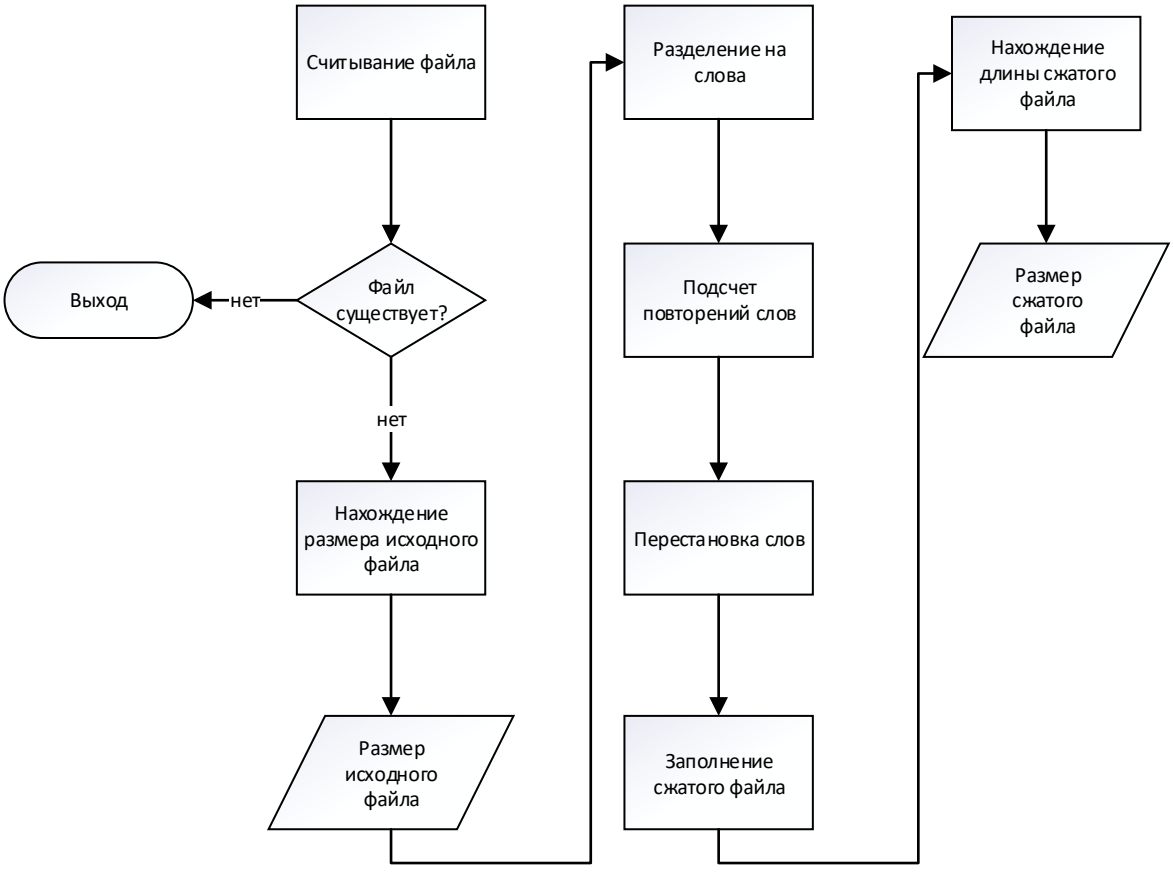

Рисунок 2 - Блок схема сжатия

Код сжатия:

```
int find_word(char* string, int* index) {
    while (string[*index] != '\0') {
        if (*index == 0 && if_letter(string[*index])) {
            return *index;
        }
        else if (string[*index] == '-' && if_letter(string[*index-1]) &&
if_letter(string[*index+1])) {
        }
        else if (if_letter(string[*index]) && !if_letter(string[*index-1])) {
            return *index;
        }
        (*index)++;
    }
    return -1;
}

char* take_word(char* str, int start) {
    int index = 0;
    char* buf = (char*)calloc(256, 1);
    while (str[start] != '\0') {
        if (str[start] == ' ' || str[start] == '\n' || str[start] == ',' ||
str[start] == ':' || str[start] == ';' || str[start] == '.' || str[start] ==
'\"' || str[start] == '!' || str[start] == '?' || str[start] == ')') {
            buf[index] = '\0';
            buf = (char*)realloc(buf, strlen(buf) + 1);
            return buf;
        }
        buf[index] = str[start];
        start++;
        index++;
    }
    buf[index] = '\0';
    buf = (char*)realloc(buf, strlen(buf) + 1);
    return buf;
}

char* check_stack(stack* head, char* word, int size_of_word) {
    stack* p = head;
    while (p) {
        if (!strcmp(word, p->word)) {
            p->size += size_of_word;
            return head;
        }
        p = p->next;
    }
    return NULL;
}

void check_words(stack** head, words** array, int* size, char* word, int
counter) {
    char* check = NULL;
    int size_of_word = strlen(word);

    if (size_of_word < 2) {
        return;
    }

    if (counter > 0) {
        size_of_word *= counter;
    }
```

```
    if (counter == 0) {
        check = check_stack(*head, word, size_of_word);
    } else {
        check = check_stack(*head, word, 0);
    }

    if (check != NULL) {
        *head = check;
        return;
    }

    push_node(head, word, size_of_word + 1);
}


void words_for_change(words** array,     glossary** arr, int* size_of_words,
int* size_of_dictionary) {
        glossary word;
    int start = ((*size_of_words) - 1);
    int end = 0;
    while (strlen((*array)[end].word) <= 4) {
        if (strlen((*array)[start].word) > strlen((*array)[end].word)) {
            if ((*array)[start].size > (*array)[end].size) {
                word.word_for_change = (*array)[start].word;
                word.word_that_change = (*array)[end].word;
                push_dictionary_in_array(word, arr, size_of_dictionary);
                start--;
                end++;
            }
        }
        if (strlen((*array)[start].word) <= strlen((*array)[end].word)) {
            start--;
        }
    }
    glossary_output(arr, size_of_dictionary);
}

void transfer_words(stack** head, words** array, int* size_of_words) {
    words buf;
    while ((*head)) {
        buf.word = (*head)->word;
        buf.size = (*head)->size;
        insert_to_array(buf, array, size_of_words);
        (*head) = (*head)->next;
    }
}

void words_for_glossary(stack** head, words** array, glossary** arr, int*
size_of_words, int* size_of_dictionary) {
    FILE* file = NULL;
    char* word = NULL;
    char* buf = (char*)calloc(4096, 1);
    int index = 0, start = 0, counter = 0;
    file = fopen("/Users/pavelshcherbo/Desktop/compress/Zip.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
    fgets(buf, 4096, file);
    while (!feof(file)) {
        while ((start = find_word(buf, &index)) != -1) {
            word = take_word(buf, start);
            counter = count_word(buf, word, &index);
```

```
            check_words(head, array, size_of_words, word, counter);
            index++;
        }
        index = 0;
        fgets(buf, 4096, file);
    }

    fclose(file);

    free(buf);
    free(word);
    transfer_words(head, array, size_of_words);
    qsort((*array), (*size_of_words), sizeof(words), compare_words);
    output(array, size_of_words);
    words_for_change(array, arr, size_of_words, size_of_dictionary);
}

void insert_glossary(glossary** arr, int* size_of_dictionary) {
    FILE* file = NULL;
    file = fopen("/Users/pavelshcherbo/Desktop/compress/compress.txt", "w");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    for (int i = 0; i < (*size_of_dictionary); i++) {
        fprintf(file, "%s", (*arr)[i].word_for_change);
        fprintf(file, "%c", '/');
        fprintf(file, "%s\n", (*arr)[i].word_that_change);
    }
    fprintf(file, "%c\n", '$');

    fclose(file);
}

char* word_from_glossary(char* word) {
    FILE* file = NULL;
    char* buf = (char*)calloc(1024, 1);
    char* new_word = NULL;

    file = fopen("/Users/pavelshcherbo/Desktop/compress/compress.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    fgets(buf, 1024, file);
    while (buf[0] != '$') {
        if ((new_word = find_word_in_glossary(buf, word)) != NULL) {
            break;
        }
        fgets(buf, 1024, file);
    }

    fclose(file);

    free(buf);
    return new_word;
}

void insert_from_glossary(char** str, char* word, char* new_word, int* index)
{
    if (new_word == NULL) {
```

```
        return;
    }
    int difference= strlen(word) - strlen(new_word), count = 0, start =
(*index), second_start = 0;
    if (difference> 0) {
        while (count != strlen(new_word)) {
            (*str)[start] = new_word[count];
            start++;
            count++;
        }
        second_start = start;
        for (int i = 0; i < difference; i++) {
            while ((*str)[start] != '\0') {
                (*str)[start] = (*str)[start + 1];
                start++;
            }
            start = second_start;
        }
        (*index) += strlen(new_word) - 1;
    }
    else {
        difference= strlen(new_word) - strlen(word);
        start = (strlen((*str)) + 1);
        for (int i = 0; i < difference; i++) {
            while (start != (*index)) {
                (*str)[start] = (*str)[start - 1];
                start--;
            }
            start = strlen((*str)) + 1;
        }
        start = (*index);
        while (count != strlen(new_word)) {
            (*str)[start] = new_word[count];
            start++;
            count++;
        }
        (*index) += strlen(new_word) - 1;
    }
}
```

Результат программы (рис.3):

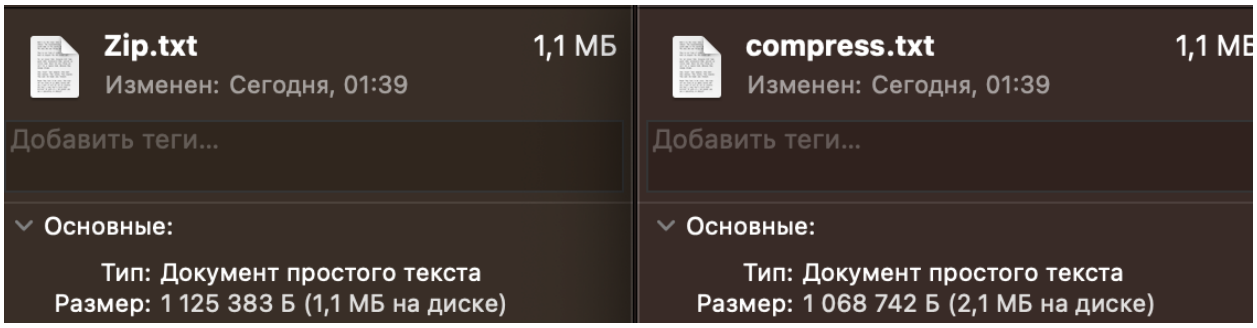

Рисунок 3 - Результат выполнения задания №1

Блок схема (рис.4):

Рисунок 4 - Блок схема для выполнения задания №2

Код программы:

```
void decompress(words** array, int* size) {
    FILE* file = NULL;
    char* buf = (char*)calloc(4096, 1);
    char* word = NULL;
    char* new_word = NULL;
    int index = 0, start = 0;
    file = fopen("/Users/pavelshcherbo/Desktop/compress/compress.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    fgets(buf, 4096, file);
    while (buf[0] != '$') {
        push_glossary(array, buf, size);
        fgets(buf, 4096, file);
    }
    fgets(buf, 4096, file);
    while (!feof(file)) {
        while ((start = find_word(buf, &index)) != -1) {
            word = take_word(buf, start);
            new_word = word_from_glossary(array, word, size);
            push_word_from_glossary(&buf, word, new_word, &index);
            index++;
        }
        puts_file(buf);
        index = 0;
        fgets(buf, 4096, file);
    }
    fclose(file);
    free(buf);
    free(word);
```

```
    free(new_word);
}
```

Блок схема (рис.5):

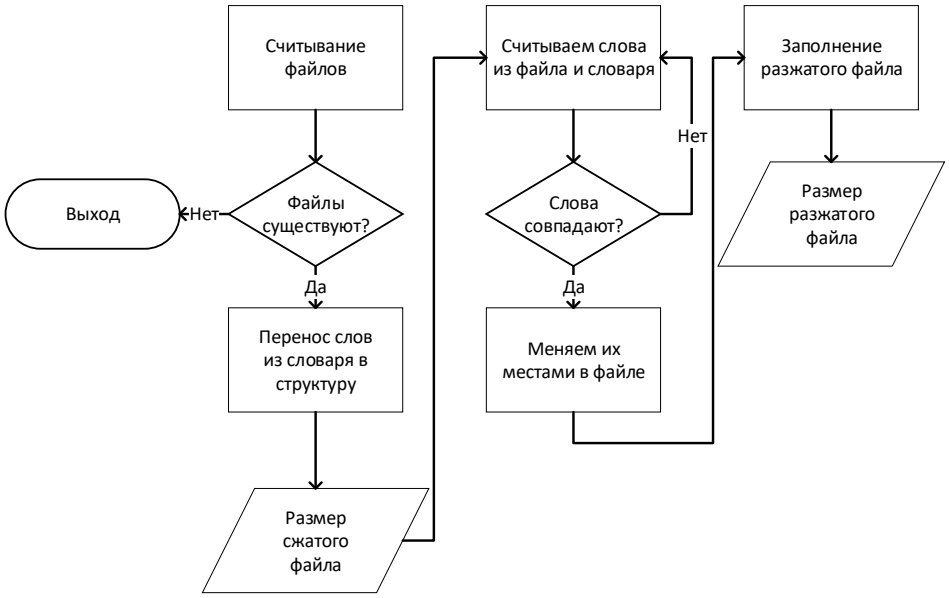

Рисунок 5 - Блок схема расжатия

Код расжатия:

```
void output(Word* wordsArrayOne, Word* wordsArrayTwo, int N)
{
   for (int i = 0; i < N; i++)
      {
   printf("%s %s\n", wordsArrayOne[i].word, wordsArrayTwo[i].word);
    }
}

int ChangeWordsInFile(FILE* file, Word* wordsArr1, Word* wordsArr2, int N, char*
word)
{
   int changed = 0;
   for (int i = 0; i < N; i++)
   {
   if (!strcmp(wordsArr1[i].word, word))
      {
   fprintf(file, "%s", wordsArr2[i].word);
   changed = 1;
   break;
   }
   else if (!strcmp(wordsArr2[i].word, word))
      {
   fprintf(file, "%s", wordsArr1[i].word);
   changed = 1;
   break;
      }
   }
   return changed;
}


void fillStruct(FILE* file, Word** wordsArr1, Word** wordsArr2, int* N)
```

```
{
   for (int i1 = 0; !feof(file); i1++)
   {
   char* word1 = malloc(100);
   char* word2 = malloc(100);

   fscanf_s(file, "%s", word1, 100);


   if (strcmp(word1, "<ENDED>") == 0)
   {
    (*wordsArr1) = realloc(*wordsArr1, (--(*N)) * sizeof(Word));
    (*wordsArr2) = realloc(*wordsArr2, (*N) * sizeof(Word));
   break;
   }
   fscanf_s(file, "%s", word2, 100);

    (*wordsArr1)[(*N) - 1].word = malloc(100);
    (*wordsArr1)[(*N) - 1].word = word1;
    (*wordsArr2)[(*N) - 1].word = malloc(100);
    (*wordsArr2)[(*N) - 1].word = word2;
    (*wordsArr1) = realloc(*wordsArr1, (++(*N)) * sizeof(Word));
    (*wordsArr2) = realloc(*wordsArr2, (*N) * sizeof(Word));
   }
}

void fillFile(FILE* file, FILE* newFile, Word* wordsArr1, Word* wordsArr2, int
N)
{
   while(!feof(file)){
   char c;
   char* word = malloc(100);
   fscanf_s(file, "%s%c", word, 100, &c, 1);
   if    (!ChangeWordsInFile(newFile,    wordsArr1,    wordsArr2,    N,    word)
&& !feof(file))
   {
   fprintf(newFile, "%s", word);
   }
   if(!feof(file)) fprintf(newFile, "%c", c);
   free(word);
   }
}
```

Результат программы (рис.6):

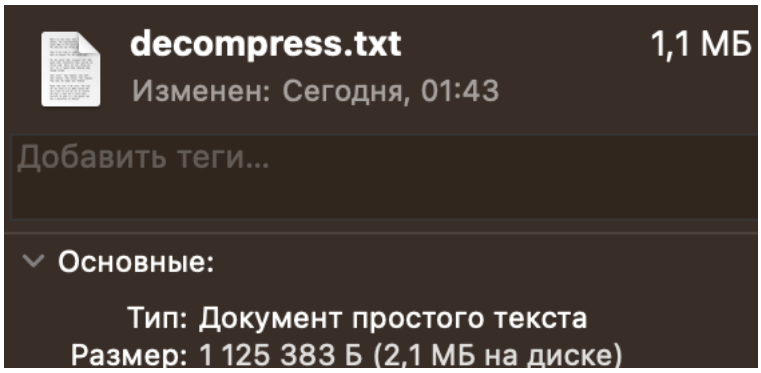

Рисунок 6 - Результат выполнения задания №2