Студент: Пантюхин А.Е., группа: ДТ-460а
Лабораторная работа №4
Задание:
Вариант 9
В бинарном файле хранится последовательность целых чисел. Вывести в порядке убывания те числа, которые встречаются в последовательности более одного раза.
Использовать для решения задачи односвязный список.

В рамках решения реализовать функции:
1) Ввода чисел с клавиатуры и записи введённого в файл
2) Чтения неупорядоченных чисел из файла и вывода в поток stdout
3) Создания списка из упорядоченных по возрастанию/убыванию значений, каждый элемент содержит счётчик повторений числа и само число.
4) Вывод содержимого списка на экран.

Тестовые данные:
1) 1 2 2 2 3 -4 -4 5 66 8 8 1
2) 0
3) 3 3 3 3 3 3 3 3 3 3 3 3

Текст программы с комментариями:
List_fill.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//создание и заполнение бинарного файла числами
int main(void)
{
    char user_input[BUFSIZ] = {0}; //буфер для хранения и обработки пользовательского ввода
    printf("Укажите имя файла для вывода списка:\n");
    fgets(user_input, BUFSIZ-1, stdin); //получили всю строку потока stdin
    sscanf(user_input, "%s", user_input); //получили из строки только значимую часть (до первого пустого элемента (" ", "\n", "\0"))
    printf("Вы ввели: %s\n", user_input); //вывод для проверки
    FILE* file = fopen(user_input, "wb"); //открытие бинарного файла для записи (и/или его создание)
    if (!file)
    {
        printf("Ошибка! Файл невозможно открыть для записи!\n");
        return 1;
    }


    printf("Введите элементы списка через пробел, ввод новой строки завершит ввод:\n");
    fgets(user_input, BUFSIZ-1, stdin); //вновь получаем строку из stdin
    printf("В файл записано: "); //информируем пользователя параллельно с записью
    char* token = strtok(user_input, " ,"); //"токенизация" строки - отделяем первый элемент (до пробела или запятой)
    while (token)
    { //пока элемент найден
        int value = atoi(token); //получаем значение типа int из этого элемента
```

```
        printf("%d ", value); //информируем пользователя
        fwrite(&value, sizeof(int), 1, file); //запись элемента в бинарный файл
        token = strtok(NULL, " ,"); //отделяем от исходной строки следующий элемент
    }
    printf("\n");
    fclose(file);//закрываем поток.
    return 0;
}
```

Task.c:

```
//В бинарном файле хранится последовательность целых чисел.
// Вывести в порядке убывания те числа, которые встречаются в последовательности
// более одного раза.
// Использовать для решения задачи односвязный список.


#include <stdio.h>


#include "list.h"


/*
Получаем из указанного бинарного файла данные в виде набора чисел, формируем из них
список1
Полученный список1 конвертируем в список1 с учётом количества вхождений
Выводим значения списка2, если их количество вхождений больше 1
*/
int main(void) {
    list* l1 = 0;//"входной" список1
    stacked_list* slist1 = 0; //итоговый список2
    char user_input[BUFSIZ] = {};//буфер для пользовательского ввода
    //BUFSIZ - размер буфера потока из stdio
    printf("Введите имя файла для ввода списка:\n");
    scanf("%s", user_input); //scanf потому, что fgets заберёт \n тоже, и, в отличие
от заполнения, нам нет дела до того, что останется в потоке
    if (!list_get_from_file(user_input, &l1)) printf("Ошибка! Невозможно прочитать
файл!\n");
    else
    {
        printf("Содержимое файла:\n");
        list_print(l1); //вывод списка, описано в месте определения
        printf("\n");
        /*
        поздно увидел, что нужна структура с количеством повторений
        поэтому конвертируем+сортируем
        */
        list_to_slist_s(l1, &slist1, 0); //конвертация списка, описано в месте
определения
        printf("Список из значений, встречающихся более одного раза, в порядке
убывания: \n");
```

```
        //вывод всех с кол-вом вхождений больше 1.
        slist_print_many(slist1); //описано в месте определения
        printf("\n");
    }
    //очистка памяти
    list_dealloc(&l1); //описано в месте определения
    slist_dealloc(&slist1); //описано в месте определения
    return 0;
}
```

List.h:

```
#ifndef LIST_H
#define LIST_H //защита от переопределения


//список
typedef struct list
{
    int value; //значение
    struct list* next; //служебная часть
}
list;


//список с подсчётом количества вхождений
typedef struct stacked_list
{
    int value; //значение
    int amount; //количество вхождений
    struct stacked_list* next; //служебная часть
}
stacked_list;


/*
функции описаны в файле с определениями.
*/


void list_push_back(int value, list** head);


void list_insert(int value, list** head);


void list_add_sorted(int value, list** head);
```

```
void list_dealloc(list** head);

void list_print(const list* head);

const list* list_contains(int value, const list* head);

int list_contains_amount(int value, const list* head);

int list_get_from_file(const char* path, list** head);

int list_fill_file(const char* path, const list* head);

stacked_list* slist_get(int value, stacked_list* head);

int slist_insert(int value, stacked_list** head);

int slist_add_s(int value, stacked_list** head, int is_raising);

int list_to_slist_s(const list* src, stacked_list** dest, int is_raising);

void slist_print(const stacked_list* head);

void slist_print_many(const stacked_list* head);

void slist_dealloc(stacked_list** head);

#endif
```

List.c:

```
#include <stdlib.h>
#include <stdio.h>


#include "list.h"


//добавление элемента в конец списка
void list_push_back(int value, list** head)
```

```
{
    if (!head) return; //некорректные данные, выход
    list* ptr = *head; //служебные указатели
    list* prev = 0;
    while (ptr)
    { //движение в конец списка
        prev = ptr;
        ptr = ptr->next;
    }
    ptr = malloc(sizeof(list)); //выделение памяти
    *ptr = (list){value, 0}; //инициализация
    if (prev) prev->next = ptr; //если есть прошлый элемент
    else *head = ptr; //иначе - мы в начале списка
    return;
}


//добавление элемента в начало списка
void list_insert(int value, list** head)
{
    if (!head) return; //некорректные данные, выход
    list* temp = malloc(sizeof(list)); //выделение памяти
    *temp = (list) {value, *head};//инициализация
    *head = temp; //добавление в список
}


//освобождение памяти под список
void list_dealloc(list** head)
{
    if (!head) return; //некорректные данные, выход
    if (!*head) return; //конец списка(или его просто нет)
    list_dealloc(&((*head)->next)); //рекурсивный вызов для следующего элемента
    free(*head); //очистка памяти
    *head = 0; //зануление, на случай если *head будет переиспользован
    return;
}


//вывод списка в поток stdout
void list_print(const list* head)
{
    if (!head) return; //некорректные данные, выход
    printf("%d ", head->value); //вывод значения
    list_print(head->next); //рекурсивный вызов для следующего элемента
    return;
}


//чтение списка из бинарного файла
int list_get_from_file(const char* path, list** head)
```

```
{
    int buffer; //новый элемент списка
    FILE* file = fopen(path, "rb"); //открываем бинарный файл на чтение
    if (!file) return 0; //файл не открылся, ошибка
    while(fread(&buffer, sizeof(int), 1, file))
    { //fread вернёт число считанных элементов, если 0 - значит, файл закончился или
встречен некорректный формат(некратный 4)
        //пока что-то прочитано, записываем значение в конец списка
        list_push_back(buffer, head);
    }
    fclose(file); //закрываем поток.
    return 1;//1 == ошибок нет
}


//заполнение бинарного файла набором значений из списка.
int list_fill_file(const char* path, const list* head)
{
    FILE* file = fopen(path, "wb"); //открываем бинарный файл на чтение(содержимое
будет стёрто)
    if (!file) return 0; //файл не открылся, ошибка
    while (head)
    {
        //запись значения в файл
        fwrite(&(head->value),sizeof(int), 1,file);
        //перемещение по списку
        head = head->next;
    }
    fclose(file); //закрываем поток
    return 1; //1 == ошибок нет
}


//получить адрес элемента с данным значением в списке
stacked_list* slist_get(int value, stacked_list* head)
{
    if (!head) return 0; //некорректные данные, выход
    if (head->value == value) return head; //если значение равно искомому -
возвращаем адрес
    return slist_get(value, head->next); //рекурсивный вызов функции для следующего
элемента (возвращаем результат)
}


//добавление элемента в начало списка (или, если он уже есть - увеличить кол-во на
1)
int slist_insert(int value, stacked_list** head)
{
    if (!head) return 0;//некорректные данные, выход
    stacked_list* elem = slist_get(value, *head);//если существует, получит адрес
    if (elem) elem->amount++; //если есть адрес, количество++
```

```
    else
    { //если нет такого элемента
        elem = malloc(sizeof(stacked_list)); //выделяем память
        *elem = (stacked_list){value, 1, *head}; //инициализация
        *head = elem; //добавление в начало списка
    }
    return 1; //1 == ошибок нет
}


int slist_add_s(int value, stacked_list** head, int is_raising)
{
    if (!head) return 0; //некорректные данные, выход
    stacked_list* ptr = *head, *prev = 0; //объявляем и инициализируем служебные
переменные
    //если список пустой или такой элемент есть, сразу добавляем
    if (!ptr || slist_get(value, ptr)) return slist_insert(value, head);
    //иначе, начинаем искать место для элемента
    while (ptr //пока не дошли до конца
        //и (по возрастанию) значение элемента меньше входного
        && (((ptr->value < value) && is_raising)
        //или (по убыванию) значение элемента больше входного
        || ((ptr->value > value) && !is_raising)))
    {
        prev = ptr; //запоминаем прошлый элемент
        ptr = ptr->next; //движемся по списку
    }
    //Либо нашли место, и ставим элемент на него, либо мы находимся в начале списка.
    slist_insert(value, &ptr);
    //Если не в начале - добавляем элемент в список "справа"
    if (prev) prev->next = ptr;
    //если в начале - добавляем "слева".
    else *head = ptr;
    return 1; //1 == ошибок нет
}


//конвертация списка в список с количеством вхождений
int list_to_slist_s(const list* src, stacked_list** dest, int is_raising)
{
    int all_good = 1; //флаговая переменная, 1 == ошибок нет
    while(src && all_good)
    { //пока не в конце исходного списка и нет ошибок...
        all_good = slist_add_s(src->value, dest, is_raising); //добавляем в список с
количеством элемент
        src = src->next; //движемся по исходному списку
    }
    return all_good;
}
```

```
//вывод списка с количеством вхождений в формате <значение>[<количество>]...
void slist_print(const stacked_list* head)
{
    if (!head) return; //списка нет или мы уже в конце, выход
    printf("%d[%d] ", head->value, head->amount); //вывод значения
    slist_print(head->next); //рекурсивный вызов для следующего элемента
    return;
}


//вывод элементов, где количество вхождений больше 1.
void slist_print_many(const stacked_list* head)
{
    if (!head) return; //списка нет или мы уже в конце, выход
    if (head->amount > 1) printf("%d ", head->value); //вывод значения
    slist_print_many(head->next); //рекурсивный вызов для следующего элемента
    return;
}


//освобождение памяти
void slist_dealloc(stacked_list** head)
{
    if (!head) return; //некорректные данные, выход
    if (!*head) return; //конец списка, выход
    slist_dealloc(&((*head)->next)); //рекурсивный вызов для следующего элемента
    free(*head);//освобождение памяти
    *head = 0; //зануление для возможности переиспользования указателя
    return;
}
```

Makefile (использовался для сборки и тестирования проекта):

```
all: filler task


clean:
    @ rm -f liblist.a
    @ rm -f list.o
    @ rm -f task
    @ rm -f filler


liblist.a: list.c list.h
    @ gcc list.c -c -o list.o
    @ ar rcs liblist.a list.o


filler: list_fill.c
```

```
    @ gcc list_fill.c -o filler


task: task.c liblist.a
    @ gcc task.c liblist.a -o task


fill: filler
    @ ./filler


run: task
    @ ./task
```