Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Программная инженерия»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ № 6

Связные списки
Тема

Руководитель _____ А.С. Черниговский
Подпись, дата Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А.К. Никитин
Номер группы, зачетной книжки Подпись, дата Инициалы, Фамилия

Красноярск 2020

1 Цель

Ознакомиться с видами, принципом построения и функционирования списков и научиться применять их для динамического хранения сложных типов данных.

2 Задачи

Для выполнения лабораторной работы необходимо взять за основу задание из лабораторной работы № 5 и выполнить следующие задачи:

1) выполнить работу в соответствии с заданием;

2) заменить массив структур двусвязным списком;

3) добавить функцию определения длины списка;

4) добавить функцию инвертирования связного списка.

3 Описание варианта задания

Реализовать структуру, описывающую дату/время. Создать следующие поля: год, месяц, день, час, минута, секунда, стандарт. Добавить вычисляемые поля: преобразование в стандарт UTC, вычисления промежутка между двумя датами, преобразование к строке.

4 Ход выполнения

Ниже представлен листинг программы по заданию.

Листинг 1 – Операции с датой и временем

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <locale.h>
#include <string.h>
#include <time.h>

typedef struct _list {
    struct tm date;
    struct _list* next;
    struct _list* prev;
} list;

struct filterOutput
{
    list* dateHead;
    list* dateTail;
    int error;
};

// Ввод целого числа
void inputNat(int* number)
{
    while (!scanf("%d", number))
    {
        fflush(stdin);
        printf("Введите корректные данные!\n");
    }
    if (*number < 0)
    {
        printf("Число должно быть натуральным или 0!\n");
        inputNat(number);
```

Продолжение листинга

```
    }
}
```

```
// Ввод строки произвольной
void inputString(char** word)
{
    int count = 0;
    char inputChar = 0;

    fflush(stdin);

    while(1)
    {
        inputChar = getchar();
        if (inputChar == '\n')
            break;
        else
        {
            *word = realloc(*word, count + 1);
            (*word)[count] = inputChar;
            count++;
        }
    }
    (*word)[count] = '\0';
}

// Проверяет правильность ввода стандарта кодирования
char* checkStandard(char* standard)
{
    int timezoneUTC = 0;

    standard = strlwr(standard);
    // Стандарт  UNIX
    if (strcmp(standard, "unix") == 0)
    {
        return "unix";
    }
        // Стандарт UTC. Если начинаеся со строки "utc"
```

Продолжение листинга

```
    else if (strstr(standard, "utc") != NULL && strcmp(strstr(standard, "utc"),
standard) == 0)
    {
```

```
        // Мировое время
        if (strlen(standard) == 3)
            return "utc";

            // Часовой пояс
        else if (strlen(standard) == 6 && (standard[3] == '+' || standard[3] ==
'-'))
        {
            timezoneUTC = atoi(standard + 3);
            if (timezoneUTC == 0 || timezoneUTC > 12 || timezoneUTC < -12)
            {
                printf("Вы ввели неправильный часовой пояс (правильные от -12 до
12)!\n");
                return NULL;
            }
            return standard;
        }
    }
    printf("Неверный формат стандарта!\n");
    return NULL;
}

// Преобразовывает локальное время к мировому
struct tm UTC2UTC(struct tm date, char* standard)
{
    int tempDate = 0;
    int timezoneUTC = 0;

    // Если время не мировое, берется поправка на временной пояс
    if (strlen(standard) > 3)
        timezoneUTC = atoi(standard + 3);

    // Преобразование
    tempDate = mktime(&date);
    tempDate -= timezoneUTC * 3600 + timezone; // Глобальная переменная timezone
из библиотеки time.h хранит в себе
    date = *gmtime(&tempDate);  // разницу локального времени компьютера с мировым
```

Продолжение листинга

```
    return date;
}
```

```
// Преобразовывает время в секундах (стандарт UNIX) в мировое
struct tm UNIX2UTC(int seconds)
{
    struct tm date;

    date = *gmtime(&seconds);

    return date;
}

// Удаляет из массива дату по порядковому номеру
struct tm* deleteDate(struct tm* dateBase, int length, int index)
{
    struct tm* newDateBase;
    newDateBase = (struct tm*) malloc(sizeof(struct tm) * (length-1));

    // Создание нового массива без учета удаляемого элемента
    for (int i = 0; i < length; i++)
    {
        if (i >= index)
            newDateBase[i] = dateBase[i+1];
        else
            newDateBase[i] = dateBase[i];
    }

    return newDateBase;
}

// Функция сортирует массив дат по определенному ключу по возрастанию или убыванию
// В функции будет много однотипного кода для каждого ключа, избавление от которого
усложнило бы код
void sortArray(list** head, char* key, int order)
{
    struct tm temp;

    if (strcmp(key, "seconds") == 0)
    {
```

Продолжение листинга

```
        // Сортировка пузырьком
        for (list* i = (*head); i != NULL; i = i->next)
            for (list* j = (*head); j != NULL; j = j->next)
```

```
            if (order)  // По возрастанию
            {
                if (i->date.tm_sec < j->date.tm_sec)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
            }
            else  // По убыванию
            if (i->date.tm_sec > j->date.tm_sec)
            {
                temp = i->date;
                i->date = j->date;
                j->date = temp;
            }

}

else if (strcmp(key, "minutes") == 0)
{
    for (list* i = (*head); i != NULL; i = i->next)
        for (list* j = (*head); j != NULL; j = j->next)
            if (order)  // По возрастанию
            {
                if (i->date.tm_min < j->date.tm_min)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
            }
            else  // По убыванию
            if (i->date.tm_min > j->date.tm_min)
            {
                temp = i->date;
                i->date = j->date;
```

Продолжение листинга

```
                j->date = temp;
            }
}
```

```
else if (strcmp(key, "hours") == 0)
{
    for (list* i = (*head); i != NULL; i = i->next)
        for (list* j = (*head); j != NULL; j = j->next)
            if (order)  // По возрастанию
            {
                if (i->date.tm_hour < j->date.tm_hour)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
            }
            else  // По убыванию
            if (i->date.tm_hour > j->date.tm_hour)
            {
                temp = i->date;
                i->date = j->date;
                j->date = temp;
            }
}

else if (strcmp(key, "monthday") == 0)
{
    for (list* i = (*head); i != NULL; i = i->next)
        for (list* j = (*head); j != NULL; j = j->next)
            if (order)  // По возрастанию
            {
                if (i->date.tm_mday < j->date.tm_mday)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
            }
            else  // По убыванию
```

Продолжение листинга

```
            if (i->date.tm_mday > j->date.tm_mday)
            {
                temp = i->date;
```

```
                i->date = j->date;
                j->date = temp;
            }
    }

    else if (strcmp(key, "weekday") == 0)
    {
        for (list* i = (*head); i != NULL; i = i->next)
            for (list* j = (*head); j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (i->date.tm_wday < j->date.tm_wday)
                    {
                        temp = i->date;
                        i->date = j->date;
                        j->date = temp;
                    }
                }
                else  // По убыванию
                if (i->date.tm_wday > j->date.tm_wday)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
    }

    else if (strcmp(key, "month") == 0)
    {
        for (list* i = (*head); i != NULL; i = i->next)
            for (list* j = (*head); j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (i->date.tm_mon < j->date.tm_mon)
                    {
                        temp = i->date;
                        i->date = j->date;
```

Продолжение листинга

```
                        j->date = temp;
                    }
                }
```

```
                else  // По убыванию
                if (i->date.tm_mon > j->date.tm_mon)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
    }

    else if (strcmp(key, "year") == 0)
    {
        for (list* i = (*head); i != NULL; i = i->next)
            for (list* j = (*head); j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (i->date.tm_year < j->date.tm_year)
                    {
                        temp = i->date;
                        i->date = j->date;
                        j->date = temp;
                    }
                }
                else  // По убыванию
                if (i->date.tm_year > j->date.tm_year)
                {
                    temp = i->date;
                    i->date = j->date;
                    j->date = temp;
                }
    }

    else
        puts("Введите правильный ключ!");
}

// Добавление нового элемента в двусвязный список
void add(list** head, list** tail, struct tm date)
```

Продолжение листинга

```
{
    list* array = (list*)malloc(sizeof(list));
    array->next = NULL;
```

```
    array->prev = NULL;
    array->date = date;

    if(*head == NULL)
        *head = *tail = array;
    else {
        (*tail)->next = array;
        array->prev = *tail;
        *tail  = array;
    }
}

// Функция возвращает элемент списка по его порядковому номеру
list* extract(list* head, int position)
{
    int count = 0;

    if (position < 0)
        return NULL;

    if (head == NULL)
        return NULL;

    while (1)
    {
        if (count == position)
            break;
        head = head->next;
        count++;
    }
    return head;
}

// Основной алгоритм для фильтрации, который будет использоваться в функции ниже
struct filterOutput filterAlgorithm(list* head, char* key)
{
    int number = 0;
```

Продолжение листинга

```
    int keyLen = 0;
    int filterVar = 0;
    list* newHead = NULL;
```

```
    list* newTail = NULL;

    struct filterOutput output;

    for (list* i = head; i != NULL; i = i->next)
    {
        // В переменной key хранится поле для фильтра. Данный условный оператор адаптирует алгоритм под каждое из полей
        if (strstr(key, "seconds") != NULL)
        {
            filterVar = i->date.tm_sec;
            keyLen = strlen("seconds");
        }
        else if (strstr(key, "minutes") != NULL)
        {
            filterVar = i->date.tm_min;
            keyLen = strlen("minutes");
        }
        else if (strstr(key, "hours") != NULL)
        {
            filterVar = i->date.tm_hour;
            keyLen = strlen("hours");
        }
        else if (strstr(key, "weekday") != NULL)
        {
            filterVar = i->date.tm_wday + 1;
            keyLen = strlen("weekday");
        }
        else if (strstr(key, "monthday") != NULL)
        {
            filterVar = i->date.tm_mday;
            keyLen = strlen("monthday");
        }
        else if (strstr(key, "month") != NULL)
        {
            filterVar = i->date.tm_mon + 1;
            keyLen = strlen("month");
```

Продолжение листинга

```
        }
        else if (strstr(key, "year") != NULL)
        {
```

```
            filterVar = i->date.tm_year + 1900; // Времяисчисление начинается с
1900 года
            keyLen = strlen("year");
        }
        puts("2");

        number = atoi(key + keyLen + 1);

        if (number == 0)
        {
            output.dateHead = NULL;
            output.dateTail = NULL;
            output.error = 1;
            return output;
        }

        if (key[keyLen] == '>')
        {
            if (filterVar > number)
                add(&newHead, &newTail, i->date);
        }

        if (key[keyLen] == '<')
        {
            if (filterVar < number)
                add(&newHead, &newTail, i->date);
        }

        if (key[keyLen] == '=')
        {
            if (filterVar == number)
                add(&newHead, &newTail, i->date);
        }
    }

    output.dateHead = newHead;
    output.dateTail = newTail;
```

Продолжение листинга

```
    output.error = 0;
    return output;
}
```

```
// Главная функция фильтрации
struct filterOutput filter(list* head, char* key)
{
    int count = 0;
    int keysNumber = 0;
    int keyLen = 0;
    int checker = 0;
    char *correctKey = NULL;
    char **keys = NULL;

    struct filterOutput output;

    // Если ключ состоит только из пробелов
    if (strspn(key, " ") == strlen(key))
    {
        output.dateHead = NULL;
        output.dateTail = NULL;
        output.error = 1;
        return output;
    }

    // Удаление пробелов из строчки
    for (int i = 0; i < strlen(key); i++)
        if (key[i] != ' ')
        {
            count++;
            correctKey = realloc(correctKey, sizeof(char) * (count));
            correctKey[count - 1] = key[i];
        }
    correctKey[count] = '\0';

    // Разделение строки по запятым
    keys = (char **) malloc(sizeof(char *));
    keys[0] = strtok(correctKey, ",");

    while (keys[keysNumber] != NULL)
```

Продолжение листинга

```
    {
        keysNumber++;
        keys = realloc(keys, sizeof(char *) * (keysNumber + 1));
```

```
        keys[keysNumber] = strtok(NULL, ",");
    }

    // Цикл для каждого из условий
    for (int i = 0; i < keysNumber; i++)
    {
        keyLen = strlen("seconds");
        // Если начинается с seconds, следующие после него символы - >, < или =,
и после них еще что-то должно быть
        if (strstr(keys[i], "seconds") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            checker = 1;
        }

        keyLen = strlen("minutes");
        if (strstr(keys[i], "minutes") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
                checker = 1;
        }

        keyLen = strlen("hours");
        if (strstr(keys[i], "hours") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
```

Продолжение листинга

```
                checker = 1;
        }
```

```
        keyLen = strlen("weekday");
        if (strstr(keys[i], "weekday") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
                checker = 1;
        }

        keyLen = strlen("monthday");
        if (strstr(keys[i], "monthday") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
                checker = 1;
        }

        keyLen = strlen("month");
        if (strstr(keys[i], "month") != 0 &&
            (keys[i][keyLen] == '=' || keys[i][keyLen] == '>' || keys[i][keyLen]
== '<')
            && strlen(keys[i]) > keyLen)
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
                checker = 1;
        }

        keyLen = strlen("year");
         if  (strstr(keys[i],  "year")  !=  0  &&  (keys[i][keyLen]  ==  '='  ||
keys[i][keyLen] == '>' || keys[i][keyLen] == '<')
            && strlen(keys[i]) > keyLen)
```

Продолжение листинга

```
        {
            output = filterAlgorithm(head, keys[i]);
            if (output.error != 0)
```

```
                checker = 1;
            }
        }
        if (checker == 0)
        {
            output.dateHead = NULL;
            output.dateTail = NULL;
            output.error = 1;
        }
        return output;
}

// Выводит даты, хранящиеся в списке
void listOutput(list* head, int enumerate)
{
    int count = 0;
    if (enumerate)
    for(list* i = head; i != NULL; i = i->next)
    {
        count++;
        printf("%d: %s\n", count, asctime(&i->date));
    }
    else
        for(list* i = head; i != NULL; i = i->next)
            printf("%s\n", asctime(&i->date));
}

// Определение длины списка
int len(list* head)
{
    int count = 0;
    for(list* i = head; i != NULL; i = i->next)
        count++;
    return count;
}
```

Продолжение листинга

```
void listFree(list** head, list** tail) {
    list* tmp, *nextElem = *head;
    while(nextElem != NULL) {
```

```
        tmp = nextElem;
        nextElem = nextElem->next;
        free(tmp);
    }
    *head = *tail = NULL;
}

void deleteElem(list** head, list** tail, int index)
{
    list* elem4Delete;

    elem4Delete = extract(*head, index);

    if (len(*head) == 1)
    {
        *head = NULL;
        *tail = NULL;
    }

    else if (elem4Delete==(*head)) // если элемент для удаления первый
    {
        (*head) = (*head)->next;
        (*head)->prev = NULL;
        free(elem4Delete);
    }
    else if (elem4Delete==(*tail)) // если элемент для удаления последний
    {
        (*tail) = (*tail)->prev;
        (*tail)->next = NULL;
        free(elem4Delete);
    }
    else  // удаление из средины списка
    {
        elem4Delete->next->prev = elem4Delete->prev;
        if(elem4Delete->next)
            elem4Delete->prev->next = elem4Delete->next;
        free(elem4Delete);
```

Продолжение листинга

```
    }
}
```

```
void invert(list** head)
{
    list* temp = NULL;
    list* current = *head;

    while (current !=  NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    if(temp != NULL )
        *head = temp->prev;
}

int main()
{
    int userChoice = 0;
    long int second = 0;

    struct tm currentDate;

    list* head = NULL, *tail = NULL;

    char* standard = NULL;

    setlocale(LC_ALL, "");

    enum Case {dateInput = 1, transform2UTC, calculateInterval, chooseDate,
dataDel, dataOutput,
            sort, filterList, showLength, invertList, loadData, dataSave,
exitProg};

    FILE *file;
```

Продолжение листинга

```
    do
    {
```

```
        printf("\nРАБОТА С ДАТОЙ:                                РАБОТА СО СПИСКОМ ДАТ:
РАБОТА С ФАЙЛОМ:\n"
               "1. Ввести дату.                                  4. Выбрать дату из
списка.              11. Загрузить дату из файла в список.\n"
               "2. Преобразовать в стандарт UTC.              5. Удалить дату из
списка.              12. Сохранить дату в файл из списка.\n"
               "3. Вычислить промежуток.                       6. Вывести список
дат.\nмежду двумя датами.                "
               "       7. Отсортировать даты в массиве.\n"
               "                                            8. Отфильтровать даты в
списке.\n"
               "                                                  НОВОЕ!!!\n"
               "                                             9. Вывести длину
списка.\n"
               "                                                  10 Инвертировать
список.\n\n\n"
               "                                        13. ВЫЙТИ ИЗ ПРОГРАММЫ\n");
        inputNat(&userChoice);

        if (userChoice < 1 || userChoice > 13)
        {
            printf("Введите значение от 1 до 11!\n");
            continue;
        }

        switch (userChoice)
        {
            // Ввод даты с клавиатуры. Дата автоматически добавится в массив с
датами
            case(dateInput):
                {
                    char* correctStandard = NULL;
                    standard = NULL;

                    printf("Введите стандарт времени: UNIX, UTC или UTCxxx (н.п.
UTC+07)\n");
                    inputString(&standard);
```

Продолжение листинга

```
                    correctStandard = checkStandard(standard);
                    if (correctStandard == NULL)
                        break;
```

```
                    // Ввод даты стандарта UNIX, т.е. время представлено в виде
секунд, начиная с 1970 года
                    if (strcmp(correctStandard, "unix") == 0)
                    {
                        printf("Введите количество секунд:\n");
                        inputNat(&second);

                        add(&head, &tail, UNIX2UTC(second));

                        printf("Дата успешно создана.\n");
                        break;
                    }

                    // Ввод мирового времени или локального времени. Условие: если
начинается с "utc"
                    else if (strcmp(strstr(standard, "utc"), standard) == 0)
                    {
                        printf("Введите номер дня:\n");
                        inputNat(&currentDate.tm_mday);
                        printf("Введите порядковый номер дня недели:\n");
                        inputNat(&currentDate.tm_wday);
                        currentDate.tm_wday--;
                        printf("Введите месяц:\n");
                        inputNat(&currentDate.tm_mon);
                        currentDate.tm_mon--;
                        printf("Введите год:\n");
                        inputNat(&currentDate.tm_year);
                        currentDate.tm_year -= 1900;
                        printf("Введите количество часов:\n");
                        inputNat(&currentDate.tm_hour);
                        printf("Введите количество минут:\n");
                        inputNat(&currentDate.tm_min);
                        printf("Введите количество секунд:\n");
                        inputNat(&currentDate.tm_sec);
```

Продолжение листинга

```
                        // Если asctime не удается перевести дату в строку, он
возвращает NULL
                        if (!asctime(&currentDate))
```

```
                {
                    printf("Введенная вами информация неверная!\n");
                    break;
                }

                add(&head, &tail, UTC2UTC(currentDate, standard));
                break;
            }
        }

    // Преобразует текущую дату в формат UTC
    case (transform2UTC):
    {
        char* dateString = NULL;

        if (standard == NULL)
        {
            printf("Сначала введите дату!\n");
            break;
        }

        // UTC
        else if (strcmp(standard, "utc") == 0)
        {
            printf("Введенная дата уже в формате UTC!\n");
            break;
        }

        // UNIX
        else if (strcmp(standard, "unix") == 0)
        {
            currentDate = UNIX2UTC(second);
            strcpy(standard, "utc");

            // Вывод даты в двух форматах
            printf("UNIX: %d\n", second);
            dateString = asctime(&currentDate);
```

Продолжение листинга

```
            printf("UTC: %s\n", dateString);
            break;
        }
```

```
                // UTCxxx
                else if (strlen(standard) == 6 && (standard[3] == '+' ||
standard[3] == '-'))
                {
                    // Строчка до преобразования
                    dateString = asctime(&currentDate);
                    printf("%s: %s\n", strupr(standard), dateString);

                    currentDate = UTC2UTC(currentDate, standard);
                    strcpy(standard, "utc");

                    //Строчка после преобразования
                    dateString = asctime(&currentDate);
                    printf("UTC: %s\n", dateString);

                    break;
                }
                else
                {
                    printf("С введенным стандартом что-то не так...");
                    break;
                }
            }

            // Изначально по заданию требовалось посчитать промежуток между двумя
датами, но я решил, что промежуток
            // между введенной датой и настоящим временем будет интереснее и в
той же степени отражает задание
            case(calculateInterval):
            {
                struct tm date;

                time_t currentTime = time(NULL);  // Текущее время
                time_t interval;

                int enteredTime = 0;
```

Продолжение листинга

```
                if (standard == NULL)
                {
                    printf("Сначала введите дату!\n");
```

```
                    break;
                }

                // Для вычисление время перевожу в секунды
                enteredTime = (strcmp(standard, "unix") == 0) ? second : mktime(&currentDate) - timezone;

                // Difftime отнимает время друг от друга. Модуль, чтобы время не было отрицательным.
                interval = abs(difftime(enteredTime, currentTime));
                date = *gmtime(&interval);

                printf("Промежуток между датами:\nГодов: %d; суток: %d; часов: %d; минут: %d; секунд: %d\n",
                       date.tm_year - 70, date.tm_mday - 1, date.tm_hour, date.tm_min, date.tm_sec);
                break;
            }

            // Выбрать дату для работы из массива
            case (chooseDate):
            {
                int dateChoice = 0;
                int count = 0;

                if (head==NULL)
                {
                    printf("Список с датами пустой!\n");
                    break;
                }

                listOutput(head, 1);

                printf("Введите интересующий вас номер даты:\n");
                inputNat(&dateChoice);

                // Если время в диапазоне
```

Продолжение листинга

```
                if (dateChoice >= 1 && dateChoice <= len(head))
                {
                    currentDate = extract(head, dateChoice - 1)->date;
```

```
            standard = "utc";
        }
        else
            printf("Введите значение в дипапазоне 1-%d\n", len(head));
        break;
    }

    // Удаляет дату из массива. Принцип работы аналогичен с выбором даты
    case (dataDel):
    {
        int dateChoice = 0;

        if (head==NULL)
        {
            printf("Массив с датами пустой!\n");
            break;
        }

        listOutput(head, 1);

        printf("Введите интересующий вас номер даты:\n");
        inputNat(&dateChoice);

        if (dateChoice >= 1 && dateChoice <= len(head))
        {
            deleteElem(&head, &tail, dateChoice - 1);
            printf("Удаление произошло успешно\n");
        }
        else
            printf("Введите значение в дипапазоне 1-%d\n", len(head));
        break;
    }

    // Вывод даты
    case(dataOutput):
    {
        listOutput(head, 0);
```

Продолжение листинга

```
        break;
    }
```

```
            // Отсортировывает массив по введенным пользователем ключу
            case (sort):
            {
                int order = 0;
                char* field = NULL;

                printf("По какому полю вы хотите произвести сортировку?\n"
                       "Доступные поля: seconds, minutes, hours, monthday,
weekday, month, year:\n");
                inputString(&field);
                printf("Вы хотите произвести сортировку по возрастанию или
убыванию?(1/0)\n");
                inputNat(&order);

                if (order != 1 && order != 0)
                {
                    printf("Введите 1 или 0!");
                    break;
                }

                sortArray(&head, field, order);
                break;
            }

            // Фильтрует массив по одному или нескольким критериям
            case (filterList):
            {
                char* keyFilterChoice = NULL;

                struct filterOutput out;

                if (head == NULL)
                {
                    printf("База данных пуста!\n");
                    break;
                }
```

Продолжение листинга

```
                printf("По каким полям вы хотите произвести сортировку?\n"
                       "Доступные поля: seconds, minutes, hours, monthday,
weekday, month, year:\n"
```

```
                    "Поля перечислять через запятую. Пример фильтрации:
year>2000,month=3\n");
                inputString(&keyFilterChoice);

                out = filter(head, keyFilterChoice);
                if (out.error == 1) // Неправильное условие
                {
                    printf("Введите правильное условие фильтрации!\n");
                    break;
                }
                else if (out.dateHead == NULL) // NULL - условию ничего не
удовлетворяет
                {
                    printf("По данному условию ничего не найдено.\n");
                    break;
                }

                listFree(&head, &tail);
                head = out.dateHead;
                tail = out.dateTail;
                listOutput(head, 0);
                break;
            }

            // Выводит длину списка
            case(showLength):
            {
                printf("Длина списка:%d\n", len(head));
                break;
            }

            // Инвертирует список
            case(invertList):
            {
                invert(&head);
                listOutput(head, 0);
                break;
```

Продолжение листинга

```
            }

            // Загрузить информацию из файла в массив
```

```
            case (loadData):
            {
                struct tm date;

                file = fopen("datebase.txt","r");

                if (file==NULL)
                {
                    printf("Файл отсутствует. Он будет создан автоматически.
Пожалуйста, повторите попытку\n");
                    file = fopen("datebase.txt","w");
                    fclose(file);
                    break;
                }

                listFree(&head, &tail); // Очистка массива для создания нового
                while (!feof(file))
                {
                    fread(&date, sizeof(struct tm), 1, file);
                    add(&head, &tail, date);
                }
                tail = tail->prev;
                tail->next = NULL;

                fclose(file);

                printf("Данные успешно загружены.\n");
                break;
            }


                // Загрузить информацию из массива в файл
            case (dataSave):
                {
                    if (head == NULL)
                    {
                        puts("Сначала заполните массив!");
```

Продолжение листинга

```
                        break;
                    }
```

```
                    file = fopen("datebase.txt", "w");
                    for (list* i = head; i != NULL; i = i->next)
                        fwrite(&i->date, sizeof(struct tm), 1, file);
                    fclose(file);

                    printf("Дата была автоматически преобразована в формат UTC и
сохранена в файл.\n");
                    break;
                }

            case (exitProg):
            {
                listFree(&head, &tail);
                exit(1);
                break;
            }
        }

    }while (1);
}
```

5 Результат

Ниже представлены скриншоты с консольным выводом.

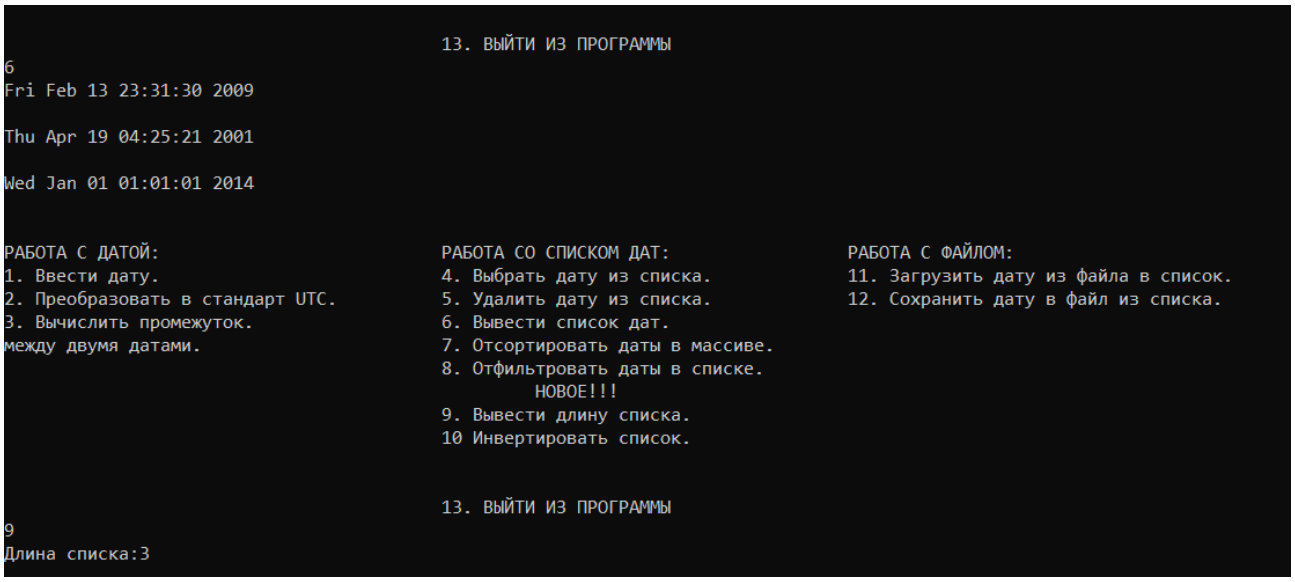

Рисунок 1 – Определение длины списка

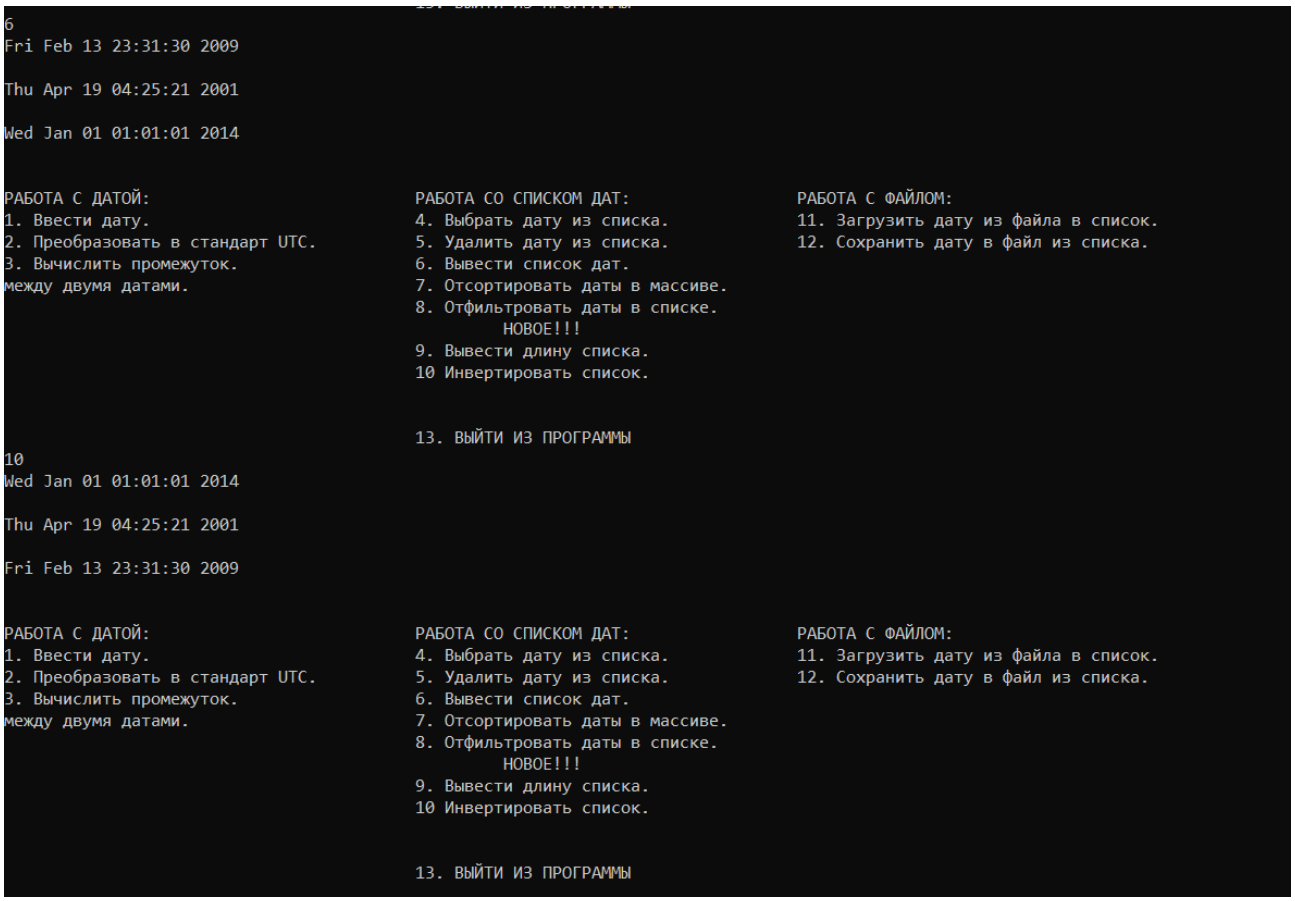

Рисунок 2 – Инвертирование списка

6 Выводы

По окончании работ были выполнены следующие задачи:

1) изучены основной алгоритм работы односвязных и двусвязных списков;

2) проведены операции над списками, такие как добавление и удаление элемента, инвертирование всех элементов;

3) реализована программа по заданию.