Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра №12 «Компьютерные системы и технологии»

ОТЧЁТ

О выполнении лабораторной работы № 6

"Работа со структурами данных на основе списков."

Студент: Гаврилов Д. А.

Группа: Б23-516

Преподаватель: Бабалова И. Ф

Москва – 2023

1. Формулировка задания

Вариант №17

Введение

Необходимо спроектировать и разработать на языке С программу, осуществляющую обработку строковых данных, на физическом уровне представленных в виде списков слов. Из входного потока вводится произвольное количество строк произвольной длины. Каждая строка в общем случае содержит одно или более слов, разделенных пробелами и/или знаками табуляции. Завершение ввода определяется концом файла. Каждая выходная строка формируется путем модификации исходной строки в соответствии с требованиями, предъявляемыми индивидуальным заданием. В полученной строке слова разделяются только одним пробелом. Исходная и полученная строки выводятся в кавычках на экран.
После обработки исходной строки и вывода результата пользователь должен иметь возможность, по желанию, выполнить произвольное количество раз какое-то из следующих действий или перейти ко вводу следующей строки:
• Удалить все вхождения некоторого слова.
• Вставить новое слово перед первым вхождением указанного слова.

Примечания:

1. Каждая строка представлена списком. Элементы списка имеют по два поля, первое из которых содержит символ, а второе — указатель на следующий элемент списка или NULL. При желании возможно использование двусвязного списка.

2. Выходная строка должна формироваться путем модификации исходной строки (т.е. путем модификации исходного списка, без создания нового).

3. Ввод строк должен быть организован с помощью функции getchar(), каждый считываемый из входного потока символ должен сразу добавляться в формируемый список.

4. Пользователь должен иметь возможность в диалоговом режиме ввести любые данные для дополнительной обработки строк.

5. Логически законченные части алгоритма решения задачи должны быть оформлены в виде отдельных функций с параметрами. Использование глобальных переменных не допускается.

6. Программа должна корректным образом работать с памятью, для проверки

необходимо использовать соответствующие программные средства, например: valgrind (при тестировании и отладке программы её необходимо запускать командой вида valgrind ./lab6).

Отчёт о выполнении лабораторной работы должен включать:

1. Блок-схемы основных алгоритмов работы программы.

2. Исходные коды программы.

3. Тестовые наборы для программы.

4. Выводы.

Индивидуальное задание

К строке, представляющей собой запись математического выражения (операнды — целые числа в десятичной системе счисления, операции — сложение и вычитание), добавить результат его вычисления. Например, строка «2 + 3 - 10» преобразуется в строку «2 + 3 - 10 = -5».

2. Описание использованных типов данных

Пользовательские типы данных для списка (List) и узла списка (Node). Целочисленный тип данных – int (спецификатор формата %d) для хранения целых чисел. Указатель на символьный тип данных – char* – для работы с словами (массивом символов) и вставке их в качестве узлов в список.

3. Описание использованных алгоритмов

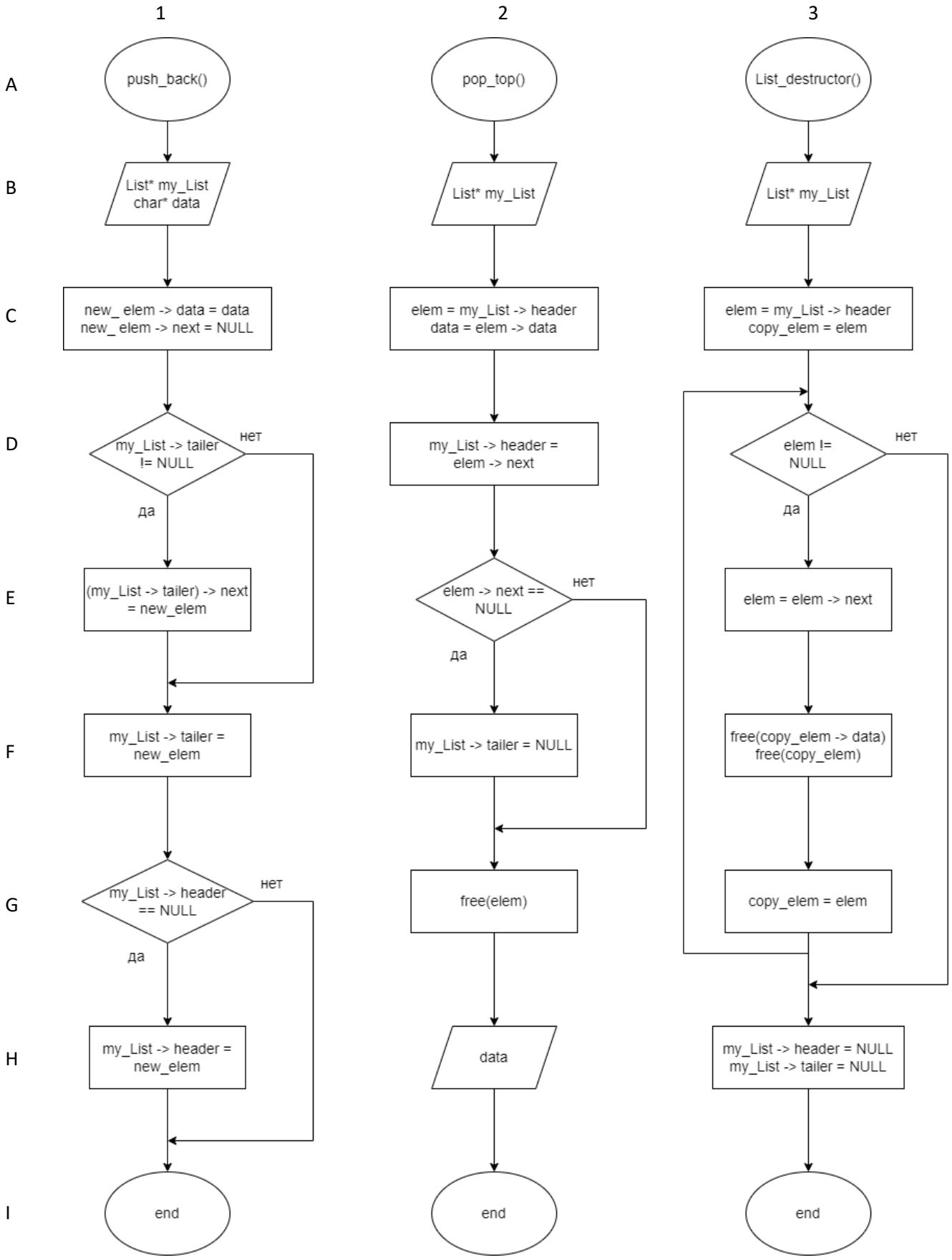

Рис. 1: Блок-схема алгоритмов работы с односвязным списком.

Вставка в конец, удаление из начала и очистка.

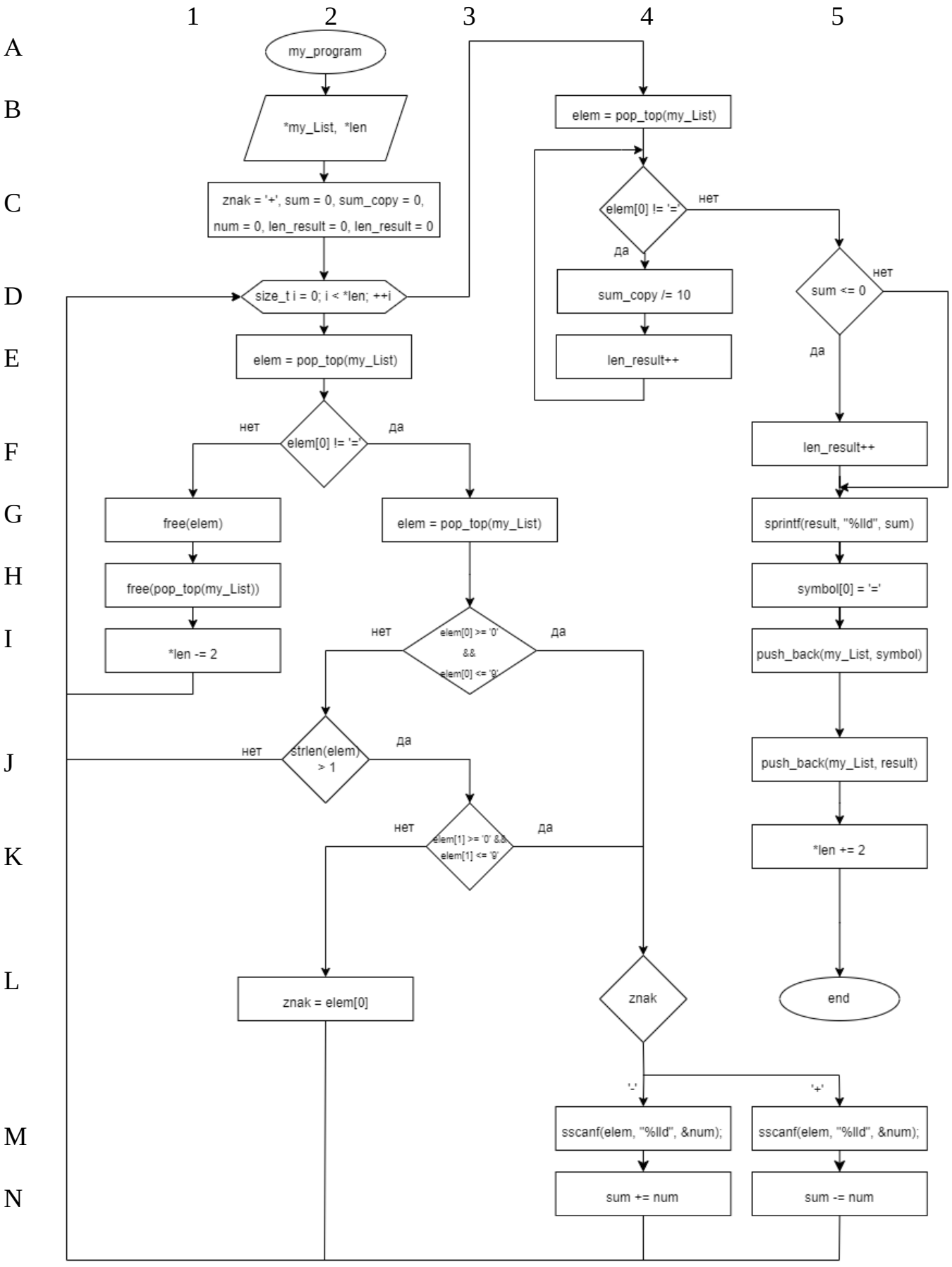

Рис. 2: Блок-схема алгоритма индивидуального задания.

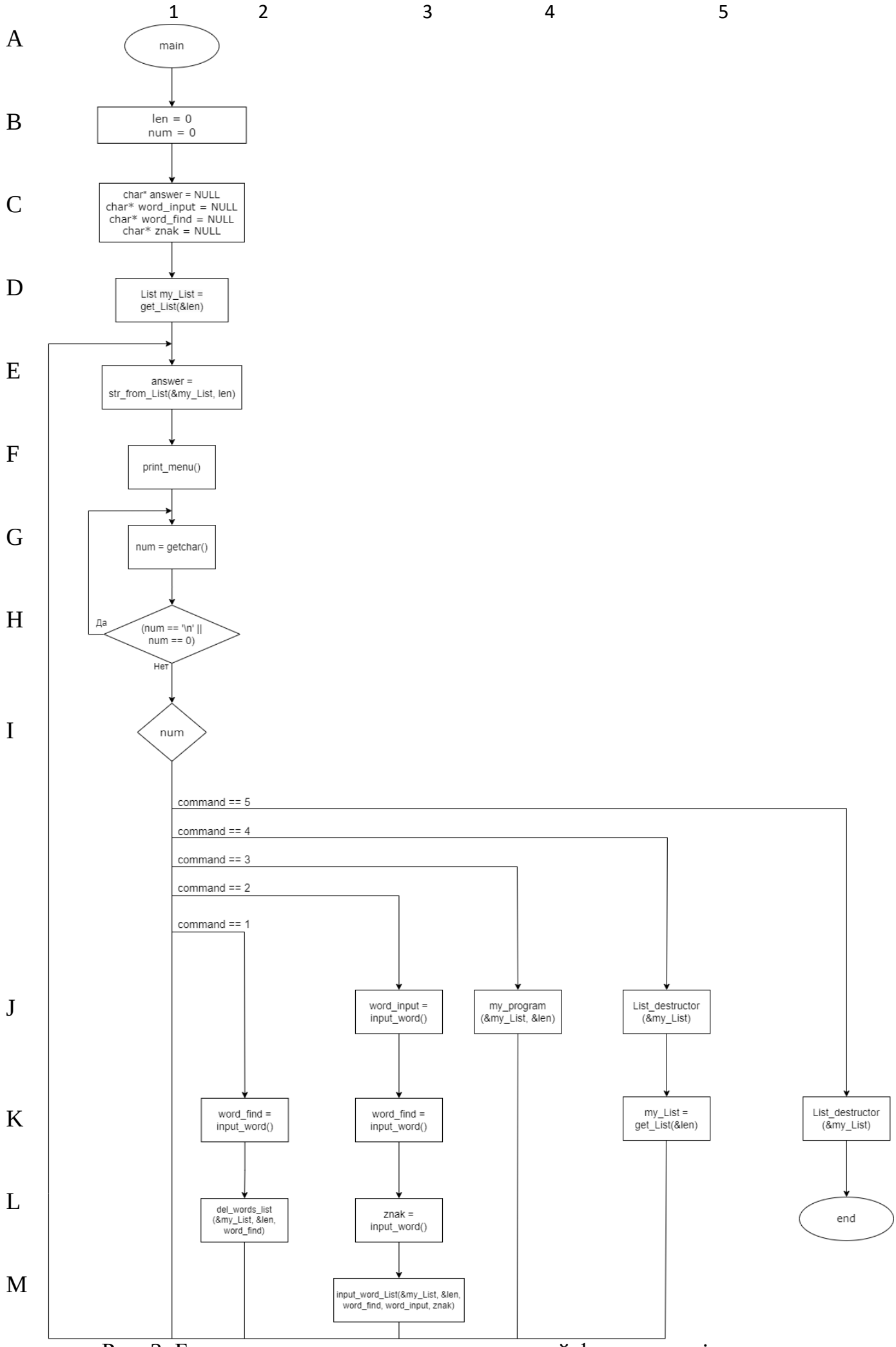

Рис. 3: Блок-схема алгоритма меню в главной функции main.

4. Исходные коды разработанных программ

Структура проекта состоит из 5 файлов: двух заголовочных - LIST.h и main_functions.h и 3 с исходным кодом – main.c, Lists.c, functions.c

Листинг 1: Исходные код программы main (файл: main.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "LIST.h"
#include "main_func.h"

int main(){
    size_t len = 0;
    char num = 0;
    char *answer = NULL;
    char *word_input = NULL, *word_find = NULL, *znak = NULL;
    printf("Введите пример: ");
    List my_List = get_List(&len);
    do {
        answer = str_from_List(&my_List, len);
        printf("Строка: \"%s\"\n", answer);
        free(answer);
        print_menu();
        do {
            num = getchar();
        } while (num == '\n' || num == 0);
        switch (num)
        {
        case '1':
            free(input_word());
            printf("Введите число которое хотите удалить: ");
            word_find = input_word();
            del_words_list(&my_List, &len, word_find);
            free(word_find);
            break;
        case '2':
            free(input_word());
            printf("Введите число перед которым хотите вставить новое: ");
            word_input = input_word();
            printf("Введите число которое хотите вставить: ");
            word_find = input_word();
            printf("Введите знак: ");
            znak = input_word();
            input_word_List(&my_List, &len, word_find, word_input, znak);
            free(word_input);
            break;
         case '3':
             my_program(&my_List, &len);
             break;
         case '4':
             scanf("%*c");
             List_destructor(&my_List);
             printf("Введите пример: ");
             my_List = get_List(&len);
             break;
        }
    } while (num != '5');
    List_destructor(&my_List);
    return 0;
}
```

Листинг 2: Исходные код программы Lists (файл: Lists.c)

```c
#include "LIST.h"
#include <stdlib.h>
#include <stdio.h>

void push_back(List *my_List, char *data) // Функция вставки в конец списка
{
    Node *new_elem = (Node *) malloc(sizeof(Node)); // Выделяем память под элемент списка
    new_elem -> data = data;    // Записываем данные в элемент
    new_elem -> next = NULL;
    if (my_List -> tailer) {    // Вставляем элемент в список (ст 10-17)
        (my_List -> tailer) -> next = new_elem;
    }
    my_List -> tailer = new_elem;
    if (!(my_List -> header)) {
        my_List -> header = new_elem;
    }
}

char *pop_top(List *my_List) // Функция удаления из начала списка
{
    Node *elem = my_List -> header; // Берём начальный элемент
    char *data = elem -> data;  // Записываем данные из элемента списка
    my_List -> header = elem -> next;   // Сдвигаем элементы (ст 24-27)
    if (!(elem -> next)) {
        my_List -> tailer = NULL;
    }
    free(elem); // Очищаем элемент списка
    return data; // Возвращаем данные
}

void List_destructor(List *my_List) // Функция очистки списка
{
    Node *elem = my_List -> header;
    Node *copy_elem = elem;
    while (elem) { // Пробегаемся по списку
        elem = elem -> next; // Переходим на след. элемент
        free(copy_elem -> data); // Очищаем данные текущего элемента
        free(copy_elem);    // Очищаем сам элемент
        copy_elem = elem;
    }
    my_List -> header = NULL;
    my_List -> tailer = NULL;
}
```

Листинг 3: Исходные код программы functions (файл: functions.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "LIST.h"
#include "main_func.h"

void print_menu(){
    printf("Что вы хотите сделать?\n");
    printf("(1) Удалить все вхождения некоторого слова\n");
    printf("(2) Вставить новое слово перед первым вхождением указанного слова\n");
    printf("(3) Выполнить обработку\n");
    printf("(4) Записать новый список\n");
    printf("(5) Выйти\n");
}

List get_List(size_t *len){
    List my_List = {NULL, NULL};
    char *word = (char *) calloc(1, sizeof(char));
    char new_symbol = (char) getchar();
    *len = 0;
    while (new_symbol != '\n') {    // Пробегаемся по всем введённым символам
        if (new_symbol >= '0' && new_symbol <= '9') {   // Если это число
            word = (char *) realloc(word, (strlen(word) + 2) * sizeof(char));
            word[strlen(word) + 1] = 0;
            word[strlen(word)] = new_symbol;
        }
        else if (strlen(word) != 0 && strchr("+-", new_symbol)){ // Если встретили знак
            push_back(&my_List, word);
            word = (char *) calloc(2, sizeof(char));
            word[0] = new_symbol;
            push_back(&my_List, word);
            word = (char *) calloc(1, sizeof(char));
            *len += 2;
        }
        else if (strlen(word) == 0 && strchr("+-", new_symbol) && *len != 0) {
            (my_List.tailer -> data)[0] = new_symbol;
        }
        new_symbol = (char) getchar();
    }
    if (strlen(word) != 0){
        push_back(&my_List, word);
        (*len)++;
```

```
    }
    return my_List;
}

char *str_from_List(List *my_List, const size_t len){ // Перевод списка в строку
    char *str = (char *) calloc(1, sizeof(char));   // Объявляем строку, в которую мы будем добавлять элементы списка
    char *elem = NULL;  // Объявляем переменную в которой мы будем хранить данные элемента списка
    for (size_t i = 0; i < len; ++i) {  // Пробегаемся по всем элементам списка
        elem = pop_top(my_List); // Достаём данные списка
        push_back(my_List, elem);  // И кладём их обратно
        if (i != 0) {   // Если это не первый элемент
            str = realloc(str, (strlen(str) + strlen(elem) + 2) * sizeof(char));
            str[strlen(str) + 1] = 0;
            str[strlen(str)] = ' '; // Добавляем пробел
        }
        else {
            str = realloc(str, (strlen(str) + strlen(elem) + 1) * sizeof(char));
        }
        strcat(str, elem);  // Прибавляем к строке данные списка
    }
    return str;
}

void my_program(List *my_List, size_t *len){ // Индивидуальное задание
    char *elem = NULL, *result = NULL, *symbol = NULL;
    char znak = '+';
    long long int sum = 0, sum_copy = 0, num = 0;
    size_t len_result = 0;
    for (size_t i = 0; i < *len; ++i) { // Пробегаемся по всем элементам списка
        elem = pop_top(my_List);
        if (elem[0] != '=') {   // Достаём элемент из спика
            push_back(my_List, elem);    // Кладём его обратно
        }
        else {    // Если встретили знак '='
            free(elem); // Очищаем знак '='
            free(pop_top(my_List));    // Очищаем ответ
            *len -= 2;
            break;
        }
        if (elem[0] >= '0' && elem[0] <= '9') {
            switch (znak)
            {
            case '+':
                sscanf(elem, "%lld", &num);
                sum += num;
                break;
            case '-':
                sscanf(elem, "%lld", &num);
                sum -= num;
                break;
            }
        }
        else if (strlen(elem) > 1) {
            if (elem[1] >= '0' && elem[1] <= '9') {
                switch (znak)
                {
                case '+':
                    sscanf(elem, "%lld", &num);
                    sum += num;
                    break;
                case '-':
                    sscanf(elem, "%lld", &num);
                    sum -= num;
                    break;
                }
            }
        }
        else {
            znak = elem[0];
        }
    }
    sum_copy = sum;
    while (sum_copy) {  // Определяем сколько знаков занимает ответ (ст 179-187)
        sum_copy /= 10;
        len_result++;
    }
```

```
    if (sum <= 0) {
        len_result++;
    }
    result = (char *) calloc(len_result + 1, sizeof(char));
    sprintf(result, "%lld", sum);
    symbol = (char *) calloc(2, sizeof(char));
    symbol[0] = '=';
    push_back(my_List, symbol);
    push_back(my_List, result);
    *len += 2;
}

char *input_word(){
    char new_symbol = (char) getchar();
    char *str = (char *) calloc(1, sizeof(char));
    while (new_symbol != '\n') {
        str = (char *) realloc(str, (strlen(str) + 2) * sizeof(char));
        str[strlen(str) + 1] = 0;
        str[strlen(str)] = new_symbol;
        new_symbol = (char) getchar();
    }
    return str;
}

void input_word_List(List *my_List, size_t *len, char *word_input, char *word_find, char *znak){
    Node *elem = my_List -> header;
    Node *new_elem = NULL, *new_znak = NULL, *prev = NULL;
    while (elem) {
        if (!strcmp(elem -> data, word_find)) {
            new_elem = (Node *) malloc(sizeof(Node));
            new_znak = (Node *) malloc(sizeof(Node));
            if (prev) {
                prev -> next = new_elem;
            }
            else {
                my_List -> header = new_elem;
            }
            new_elem -> next = new_znak;
            new_znak -> next = elem;
            new_elem -> data = word_input;
            new_znak -> data = znak;
            *len += 2;
            break;
        }
        prev = elem;
        elem = elem -> next;
    }
}

void del_words_list(List *my_List, size_t *len, char *word){
    Node *elem = my_List -> header;
    Node *next = NULL;
    Node *prev = NULL;
    while (elem) {
        next = elem -> next;
        if (!strcmp(elem -> data, word)) {
            if (!next) {
                my_List -> tailer = prev;
            }
            if (prev) {
                prev -> next = next;
            }
            else {
                my_List -> header = next;
            }
            free(elem -> data);
            free(elem);
            (*len)--;
        }
        else {
            prev = elem;
        }
        elem = next;
    }
    elem = my_List -> header;
    prev = NULL;
```

```
    while (elem) {
        next = elem -> next;
        if (next) {
            if ((!strcmp(elem -> data, "+") || !strcmp(elem -> data, "-")) && (!strcmp(next -> data, "+") || !strcmp(next -> data, "-"))) {
                if (prev) {
                    prev -> next = next;
                }
                else {
                    my_List -> header = next;
                }
                free(elem -> data);
                free(elem);
                (*len)--;
            }
            else if (!prev) {
                if (!strcmp(elem -> data, "+") || !strcmp(elem -> data, "-")) {
                    if (prev) {
                        prev -> next = next;
                    }
                    else {
                        my_List -> header = next;
                    }
                    free(elem -> data);
                    free(elem);
                    (*len)--;
                }
                else {
                    prev = elem;
                }
            }
            else {
                prev = elem;
            }
        }
         else if (!next) {
             if (!strcmp(elem -> data, "+") || !strcmp(elem -> data, "-")) {
                 my_List -> tailer = prev;
                 if (prev) {
                     prev -> next = next;
                 }
                 else {
                     my_List -> header = next;
                 }
                 free(elem -> data);
                 free(elem);
                 (*len)--;
             }
             else {
                 prev = elem;
             }
         }
         else {
             if (prev) {
                 if ((!strcmp(elem -> data, "+") || !strcmp(elem -> data, "-")) && prev -> data[0] == '=') {
                     if (!next) {
                         my_List -> tailer = prev;
                     }
                     prev -> next = next;
                     free(elem -> data);
                     free(elem);
                     (*len)--;
                 }
                 else {
                     prev = elem;
                 }
             }
             else if (!strcmp(elem -> data, "+") || !strcmp(elem -> data, "-")) {
                 if (!next) {
                     my_List -> tailer = prev;
                 }
                 my_List -> header = next;
                 free(elem -> data);
                 free(elem);
                 (*len)--;
             }
              else {
                  prev = elem;
              }
         }
         elem = next;
    }
}
```

Листинг 4: Исходные код программы Lists (файл: LIST.h)

```
#ifndef LIST_H
#define LIST_H

typedef struct Node{
    char *data;
    struct Node *next;
} Node;

typedef struct List{
    Node *header;
    Node *tailer;
} List;

void push_back(List *, char *);
char *pop_top(List *);
void List_destructor(List *);
void print_List(const List);

#endif

```

Листинг 5: Исходные код программы functions (файл: main_func.h)

```
#ifndef MAIN_FUNC
#define MAIN_FUNC
void print_menu();
List get_List(size_t *);
char *str_from_List(List *, const size_t);
void my_program(List *, size_t *);
char *input_word();
void input_word_List(List *, size_t *, char *, char *, char *);
void del_words_list(List *, size_t *, char *);
#endif

```

5. Текстовые примеры работы алгоритма вычисления арифметического выражения.

Входной список	Вывод результирующего списка
“2 – 15 + -34 - 34 + 13”	“2 – 15 -34 - 34 + 13 = -68”
“0 + 23 – 121 + 500 - 200 - 1”	“0 + 23 - 121 + 500 - 200 - 1 = 201”
“2314313 + 2332”	“2314313 + 2332 = 2316645“
“”	“”
“0”	“0 = 0”
“121 + + + + + - 121 + 7”	“121 - 121 + 7 = 7”
“232 - 232 + 13 - 7 + 32 - 32 - 100”	“232 - 232 + 13 - 7 + 32 - 32 - 100 = -94”

6. **Текстовые примеры работы алгоритма удаления чисел из выражения.**

Входной список	Вывод результирующего списка
“2314313 + 2332 + 12313”	“2314313 + 2332”
“0 + 23 – 121 + 500 – 200 – 1 + 13 + 13”	“0 + 23 – 121 + 500 – 200 - 1”
“0 + 1 + 1”	“0”
“232 - 232 + 1000 + 13 - 7 + 32 - 32 – 100 + 1000”	“232 - 232 + 13 - 7 + 32 - 32 -100”

7. **Текстовые примеры работы алгоритма вставки числа в выражение.**

Входной список	Вывод результирующего списка
“2 + 2”	“2 + 2 + 2”
“123 – 132 + 234323 – 32”	“345 + 123 – 132 + 234323 – 32”
“23432 – 334 + 2378327 – 272718”	“23432 – 12345 + 334 + 2378327 – 272718”
“0 + 1 + 1”	“0 + 123 + 1 + 1”

8. Скриншоты работы программы

Команда для сборки программы:

cc –o reslab6 main.c functions.c Lists.c

Запуск с valgrind: valgrind ./reslab6

```
[gavrilov.da@unix 6]$ cc -o reslab6 main.c functions.c Lists.c -g
[gavrilov.da@unix 6]$ valgrind ./reslab6
==8320== Memcheck, a memory error detector
==8320== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8320== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==8320== Command: ./reslab6
==8320==
Введите пример: 2314313 + 2332 + 12313
Строка: "2314313 + 2332 + 12313"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
1
Введите число которое хотите удалить: 12313
Строка: "2314313 + 2332"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
3
Строка: "2314313 + 2332 = 2316645"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
4
Введите пример: 0 + 23 - 121 + 500 - 200 - 1 + 13 + 13
Строка: "0 + 23 - 121 + 500 - 200 - 1 + 13 + 13"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
1
Введите число которое хотите удалить: 13
Строка: "0 + 23 - 121 + 500 - 200 - 1"
Что вы хотите сделать?
```

```
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
3
Строка: "0 + 23 - 121 + 500 - 200 - 1 = 201"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
4
Введите пример: 0 + 1 + 1
Строка: "0 + 1 + 1"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
1
Введите число которое хотите удалить: 1
Строка: "0"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
3
Строка: "0 = 0"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
4
Введите пример: 232 - 232 + 1000 + 13 - 7 + 32 - 32 - 100 + 1000
Строка: "232 - 232 + 1000 + 13 - 7 + 32 - 32 - 100 + 1000"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
3
Строка: "232 - 232 + 13 - 7 + 32 - 32 - 100 = -94"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
5
==8320==
==8320== HEAP SUMMARY:
==8320==     in use at exit: 0 bytes in 0 blocks
==8320==   total heap usage: 434 allocs, 434 frees, 7,294 bytes allocated
==8320==
==8320== All heap blocks were freed -- no leaks are possible
==8320==
==8320== For lists of detected and suppressed errors, rerun with: -s
==8320== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[gavrilov.da@unix 6]$
```

```
[gavrilov.da@unix 6]$ valgrind ./reslab6
==7055== Memcheck, a memory error detector
==7055== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==7055== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==7055== Command: ./reslab6
==7055==
Введите пример: 2 + 2
Строка: "2 + 2"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
2
Введите число перед которым хотите вставить новое: 2
Введите число которое хотите вставить: 2
Введите знак: +
Строка: "2 + 2 + 2"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
4
Введите пример: 123 - 132 + 234323 - 32
Строка: "123 - 132 + 234323 - 32"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
2
Введите число перед которым хотите вставить новое: 123
Введите число которое хотите вставить: 345
Введите знак: +
Строка: "345 + 123 - 132 + 234323 - 32"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
```

```
4
Введите пример: 23432 - 334 + 2378327 - 272718
Строка: "23432 - 334 + 2378327 - 272718"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
2
Введите число перед которым хотите вставить новое: 334
Введите число которое хотите вставить: 12345
Введите знак: +
Строка: "23432 - 12345 + 334 + 2378327 - 272718"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
4
Введите пример: 0 + 1 + 1
Строка: "0 + 1 + 1"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
2
Введите число перед которым хотите вставить новое: 1
Введите число которое хотите вставить: 123
Введите знак: +
Строка: "0 + 123 + 1 + 1"
Что вы хотите сделать?
(1) Удалить все вхождения некоторого слова
(2) Вставить новое слово перед первым вхождением указанного слова
(3) Выполнить обработку
(4) Записать новый список
(5) Выйти
5
==7055==
==7055== HEAP SUMMARY:
==7055==     in use at exit: 0 bytes in 0 blocks
==7055==   total heap usage: 246 allocs, 246 frees, 4,322 bytes allocated
```

```
==7055==
==7055== HEAP SUMMARY:
==7055==     in use at exit: 0 bytes in 0 blocks
==7055==   total heap usage: 246 allocs, 246 frees, 4,322 bytes allocated
==7055==
==7055== All heap blocks were freed -- no leaks are possible
==7055==
==7055== For lists of detected and suppressed errors, rerun with: -s
==7055== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[gavrilov.da@unix 6]$
```

9. Трудности при выполнении работы

При работе с односвязными списками возникла проблема, состоящая в том, что для реализации эффективного алгоритма вставки элемента в конец списка необходимо хранить указатель на последний элемент. Проблема была решена добавлением поля tailer в структуру List, которое указывает на последний ненулевой элемент списка.

10. Выводы

В ходе выполнения данной работы на примере программы, выполняющей преобразование выражения и вычисление результата, а также удаление чисел из выражения и добавление чисел в выражение были рассмотрены базовые принципы работы со структурами данных на основе односвязных списков в языке Си:

1. Основные алгоритмы по работе с односвязными списками (объявление, добавление, удаление элементов).
2. Организация системы подпрограмм, находящихся в разных файлах.
3. Создание пользовательского интерфейса в виде меню с выбором действий (выбор).
4. Создание и работа с заголовочными файлами (. h) в языке Си.