**Правительство Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего профессионального образования   
"Национальный исследовательский университет   
"Высшая школа экономики"**

Московский институт электроники и математики Национального

исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

**ОТЧЕТ**

**По лабораторной работе №12**

**УКАЗАТЕЛИ. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ. ОБРАБОТКА СПИСКОВ**

**По курсу «Алгоритмизация и программирование»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ФИО студента | | Номер группы | Дата |
| Кармаев Александр Андреевич | БПМ211 | 21.11.2021 |
|  |
|  |
|  |

**Москва – 2021 г.**

**ЗАДАНИЕ (вариант №13)**

**Задача №1**

*Изображение выглядит как стол

Автоматически созданное описание*

*Изображение выглядит как текст

Автоматически созданное описание*

**РЕШЕНИЕ**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

void number\_input(char\* str, int\* p) { // функция присвоит необходимое(числовое) значение переменной в основной функции, находящейся по адресу(2-ой аргумент)

char string[256];

printf("%s", str);

fgets(string, sizeof(string), stdin);

while (sscanf(string, "%i", p) != 1) {

printf("Incorrect input, try again!! %s", str);

fgets(string, sizeof(string), stdin);

}

}

typedef struct { // определяем структуру, элементы которой будут составлять односвязный список

// они содержат:

int coefficient; // коэффициент перед Х в какой-то степени

int degree; // степень Х

struct Node\* next; // ссылка на следующий элемент в списке

} Node;

int condition(const Node\* head, int degree) { // проверка на отсутствие повторения степеней при Х

while (head) {

if (head->degree == degree)

return 1;

head = head->next;

}

return 0;

}

void push(Node\*\* head) { // функция "добавляет" новый элемент в начало списка

Node\* tmp = (Node\*)malloc(sizeof(Node)); // новый узел, который будет помещён в список

do {

number\_input("Input positive degree of the element (without repeating): ", &(tmp->degree));

} while (condition(\*head, tmp->degree) || tmp->degree < 0);

do {

number\_input("Input coefficient of the element: ", &(tmp->coefficient));

} while (tmp->coefficient == 0);

// переопределяем указатели для упорядочивания элементов в списке

tmp->next = (\*head);

(\*head) = tmp;

}

Node\* getNth(Node\* head, int n) { // функция возвращает указатель на элемент списка под индексом(параметр функции)

int counter = 1;

while (counter < n && head) {

head = head->next;

counter++;

}

return head;

}

void print\_list(const Node\* head, int n) { // функция печатает список в обратном порядке(по порядку ввода значений) с учётом всех возможных значений коэффициентов и степеней

Node\* tmp = (Node\*)malloc(sizeof(Node));

for (int i = n; i > 0; i--) {

tmp = getNth(head, i);

if (i != n) {

printf(" %c ", (tmp->coefficient > 0) ? '+' : '-');

if (tmp->degree == 0)

printf("%d", abs(tmp->coefficient));

else if (abs(tmp->coefficient) == 1)

printf("x");

else

printf("%dx", abs(tmp->coefficient));

}

else {

if (tmp->degree == 0)

printf("%d", tmp->coefficient);

else if (tmp->coefficient == 1)

printf("x");

else if (tmp->coefficient == -1)

printf("-x");

else

printf("%dx", tmp->coefficient);

}

if (tmp->degree > 1)

printf("^%d", tmp->degree);

}

printf("\n");

}

void print\_final\_polynomial(Node\* L1, Node\* L2, int len1, int len2) { // функция создаёт конечный многочлен и печатает его

Node\* L = NULL; // указатель на первый элемент финального многочлена

int n = 0;

for (int i = 0; i < len1; i++) {

Node\* tmp = (Node\*)malloc(sizeof(Node)); // переменная будет содержать данные по элементу, который мы будем добавлять в итоговый список

Node\* tmp2 = (Node\*)malloc(sizeof(Node)); // переменная будет ссылается на второй многочлен, чтобы можно было несколько раз по нему пробежаться

tmp2 = L2;

for (int j = 0; j < len2; j++) {

if (L1->degree == tmp2->degree) {

n++;

tmp->degree = L1->degree;

tmp->coefficient = (tmp2->coefficient > L1->coefficient) ? tmp2->coefficient : L1->coefficient;

tmp->next = L;

L = tmp;

break;

}

tmp2 = tmp2->next;

}

L1 = L1->next;

}

printf("L = ");

if (n != 0)

print\_list(L, n);

}

int main(void) {

Node\* head1 = NULL; // указатель на первый элемент первого многочлена

Node\* head2 = NULL; // второго многочлена

int n[] = { 0, 0 }; // массив количества элементов списков

// создаём по очереди два списка

for (int i = 0; i < 2; i++) {

int choise = 0;

printf("\nCreate %d polynomial:\n", i + 1);

do {

push((i == 0) ? &head1 : &head2);

n[i]++;

do {

number\_input("\n If you wnat to input one more element of the polynomial press 1, else 0: ", &choise);

} while (choise != 1 && choise != 0);

} while (choise != 0);

}

// выводим по очереди все многочлены

printf("\n Your polynomials:\nL1 = ");

print\_list(head1, n[0]);

printf("L2 = ");

print\_list(head2, n[1]);

print\_final\_polynomial(head1, head2, n[0], n[1]);

return 0;

}

**ТЕСТЫ**

**Тест № 1** // пример содержит все возможные обрабатываемые исключения

**Изображение выглядит как текст

Автоматически созданное описание**

**Тест № 2** // обычный пример

Изображение выглядит как текст

Автоматически созданное описание

*Изображение выглядит как стол

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание*

**Задача №2  
РЕШЕНИЕ**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int latin\_letter(char ch) { // функция проверяет, является ли символ допустимым для использования (строчные латинские буквы)

char aval[] = "qwertyuiopasdfghjklzxcvbnm"; // массив допустимых символов

for (int i = 0; i < strlen(aval); i++) {

if (aval[i] == ch) // если находим совпадение, то возвращаем 1 (правда - символ является допустимым)

return 1;

}

return 0; // если не будет возвращена 1, то возвращаем 0 (ложь)

}

char\* reverse(char\* str) { // функция возвращает строку, обратную введённой(аргумент функции)

for (int i = 0; i < strlen(str) / 2; i++) {

char tmp = str[i];

str[i] = str[strlen(str) - i - 1];

str[strlen(str) - i - 1] = tmp;

}

return str;

}

typedef struct Node { // создаём структуру, элемент которой будет представлять из себя один узел двусвязного списка

char\* value; // значение узла (слово)

struct Node\* next; // адрес следующего узла

struct Node\* prev; // адрес предыдущего узла

} Node;

typedef struct Doble\_Linked\_List { // создаём структуру, элемент которой будет представлять из себя сам двусвязный список

// списки хранят в себе:

size\_t size; // количество элементов

Node\* head; // указатель на первый элемент списка

Node\* tail; // указатель на последний элемент списка

} List;

void push\_back(List\* list, char\* value) { // функция "добавляет" новый элемент в конец списка

Node\* tmp = (Node\*)malloc(sizeof(Node)); // указатель на новый элемент

tmp->value = value; // присваиваем значению элемента значение аргумента функции

// переопределяем указатели для упорядочивания элементов в списке

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail != NULL) { // переопределяем "хвост" списка

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) { // переопределяем "голову" списка - необходимо для ситуации, когда не было добавлено ни одного элемента

list->head = tmp;

}

list->size++; // увеличиваем значение размера списка

}

Node\* getNth(List\* list, size\_t index) { // функция возвращает указатель на узел списка по индексу в параметре(первый элемент имеет индекс 0)

Node\* tmp = NULL;

size\_t i;

if (index < list->size / 2) { // делаем максимально эффективный подход к нужному узлу в зависимости от его местоположения(ближе к началу или концу)

i = 0;

tmp = list->head;

while (tmp && i < index) {

tmp = tmp->next;

i++;

}

}

else {

i = list->size - 1;

tmp = list->tail;

while (tmp && i > index) {

tmp = tmp->prev;

i--;

}

}

return tmp;

}

void insertNth(List\* list, size\_t index, char\* value) { // функция "вставляет" новый узел перед элементом по индексу

Node\* elm = NULL;

Node\* ins = NULL;

elm = getNth(list, index); // получаем указатель на элемент по индексу

ins = (Node\*)malloc(sizeof(Node));

ins->value = value;

// переопределяем указатели для упорядочивания элементов в списке

ins->next = elm;

ins->prev = elm->prev;

if (elm->prev != NULL) {

elm->prev->next = ins;

}

elm->prev = ins;

if (ins->prev == NULL) {

list->head = ins;

}

if (elm->next == NULL) {

list->tail = elm;

}

list->size++;

}

void print\_list(List\* list) { // функция печатает предложение, состоящее из слов в списке

Node\* tmp = list->head;

printf("%s", tmp->value);

tmp = tmp->next;

while (tmp) {

printf(" %s", tmp->value);

tmp = tmp->next;

}

printf(".\n");

}

int condition\_and\_answer(List\* list, char str[256]) { // функция создаёт список и печатает две строки(исходную и финальную)

int str\_length = strlen(str); // переменная хранит размер изначальной строки, введённой пользователем

if (str[str\_length - 1] != '.' || str[0] == '.') // если в конце нет точки, то дальше можно и не смотреть или если пользователь введёт только точку(могли бы возникнуть ошибки доступа при попытке вывода пустого списка

return 1;

char\* word = NULL; // динамическая строка, в которую будем записывать наши слова

int word\_len = 0; // длинна слов

for (int i = 0; i < str\_length; i++) { // пробегаемся по каждому символу изначальной строки

if (latin\_letter(str[i])) { // если символ допустим, то мы добавляем его в наше слово

word\_len++;

word = realloc(word, word\_len \* sizeof(char));

word[word\_len - 1] = str[i];

}

else if (str[i] == ' ' || str[i] == '.') { // если символ недопустим, но это точка или пробел, то мы добавляем слово в список

if (word\_len == 0) // проверяем, что слово состоит хотя бы из одного символа

continue; // то есть учитываем возможность ввода нескольких разделительных пробелов

word[word\_len] = '\0'; // ставим символ конца строки в конец нашего слова

push\_back(list, word); // "вставляем" слово в список

word = NULL; // обнуляем наше слово, чтобы записывать в него новые

word\_len = 0;

}

else

return 1;

}

printf("\nThe original string: "); // выводим изначальное предложение

print\_list(list);

int index = 0; // переменная будет хранить индекс

for (int i = 1; i < list->size; i++) { // получаем индекс минимального слова в списке

if (strcmp(getNth(list, index)->value, getNth(list, i)->value) < 0 ) {

index = i;

}

}

word = malloc(sizeof(getNth(list, index)->value)); // переменная будет хранить копию слово под нужным индексом, чтобы потом вставить его реверсивную запись перед этим словом

strcpy(word, getNth(list, index)->value);

insertNth(list, index, reverse(word));

printf("\nThe final string: "); // выводим итоговое предложение

print\_list(list);

return 0;

}

int main(void) {

List\* list = malloc(sizeof(List)); // создаём экземпляр двусвязного списка

char str[256];

do {

list->size = 0;

list->head = list->tail = NULL;

printf("Input string by task condition: ");

gets(str);

} while (condition\_and\_answer(list, str)); // просим пользователя ввести строку, пока она не будет введена в соответствии с условием задачи и потом печатаем ответ

return 0;

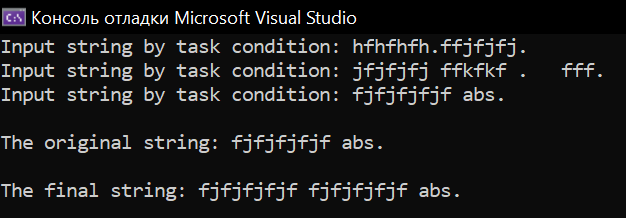
}

**ТЕСТЫ**

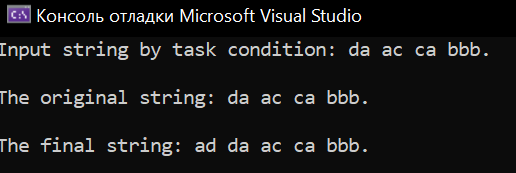
**Тест № 1** // пример содержит обработку исключений

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Тест № 2**

****

**Тест № 3**

**Изображение выглядит как текст

Автоматически созданное описание**

**Тест № 4**

**Изображение выглядит как текст

Автоматически созданное описание**

**Тест № 5**

Изображение выглядит как текст

Автоматически созданное описание