**ЛАБОРАТОРНА РОБОТА № 3**

**Зв’язний список, стек, черга. Зворотній польський запис**

***Мета:*** ознайомитися з основами роботи з двозв'язним списком, однозв'язним списком, стеком та чергою. Розробити основні функції для обчислення арифметичного виразу, записаного з використанням зворотнього польского запису

**Зміст роботи**

Завдання 1. Доповнив функції роботи з двозв’язним списком у прикладі, що розглядався на лекції.

Завдання 2. Створив програму демонстрації роботи з двозв’язним списком (доповнив приклад з лекції функціями вставки та витягнення з середини по номеру у списку та зв’язок останньої ланки з першою).

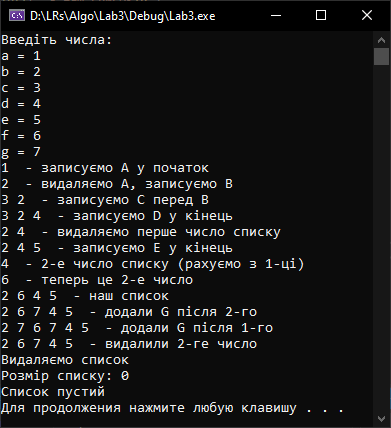


Рис.3.1 – демонстрація роботи зі списком

Лістинг програми:

Доповнений файл з функціями (додано push- та popBack() ):

#ifndef \_CLIST\_H\_

#define \_CLIST\_H\_

#include <stdlib.h>

#include <stdbool.h>

typedef int elemtype; // Тип элемента списка

struct elem {

elemtype value; // Значение переменной

struct elem\* next; // Ссылка на следующий элемент списка

struct elem\* prev; // Ссылка на предыдущий элемент списка

};

struct myList {

struct elem\* head; // Первый элемент списка

struct elem\* tail; // Последний элемент списка

size\_t size; // Количество элементов в списке

};

typedef struct elem cNode;

typedef struct myList cList;

// Создание пустого списка

cList\* createList(void);

// Удаление списка

int deleteList(cList\* list);

// Проверка списка на пустоту

bool isEmptyList(cList\* list);

// Добавление нового узла в начало списка

int pushFront(cList\* list, elemtype\* data);

// Извлечение узла из начала списка

int popFront(cList\* list, elemtype\* data);

// Добавление нового узла в конец списка

int pushBack(cList\* list, elemtype\* data);

// Извлечение узла из конца списка

int popBack(cList\* list, elemtype\* data);

// Добавление нового узла в нужное место (index), начиная с 0 и заканчивая последним в списке

int pushInside(cList\* list, elemtype\* data, int index);

// Извлечение узла из нужного места (index) в списке

int popInside(cList\* list, elemtype\* data, int index);

// Чтение произвольного узла списка

cNode\* getNode(cList\* list, int index);

// Вывод списка в консоль

void printList(cList\* list, void (\*fun)(elemtype\*));

#endif //\_CLIST\_H\_

Файл реалізації функцій:

#include "clist.h"

// Создание пустого списка

cList\* createList(void) {

cList\* list = (cList\*) malloc(sizeof(cList));

if(list) {

list->size = 0;

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

}

return list;

}

// Удаление списка

int deleteList(cList \*list) {

cNode\* head = list->head;

cNode\* next = NULL;

while (head) {

next = head->next;

free(head);

head = next;

}

free(list);

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

list->size = 0;

if (list->head == NULL) {

return 0;

}

else

{

return -1;

}

}

// Проверка списка на пустоту

bool isEmptyList(cList\* list) {

return ((list->head == NULL) || (list->tail == NULL));

}

// Добавление нового узла в начало списка

int pushFront(cList\* list, elemtype\* data) {

cNode\* node = (cNode\*) malloc(sizeof(cNode));

if (!node) {

return(-1);

}

node->value = \*data;

node->next = list->head;

node->prev = NULL;

if (!isEmptyList(list)) {

list->head->prev = node;

}

else {

list->tail = node;

}

list->head = node;

list->size++;

return(0);

}

// Извлечение узла из начала списка

int popFront(cList\* list, elemtype\* data) {

cNode\* node;

if (isEmptyList(list)) {

return(-2);

}

node = list->head;

list->head = list->head->next;

if (!isEmptyList(list)) {

list->head->prev = list->tail;

}

else {

list->tail = NULL;

}

\*data = node->value;

list->size--;

free(node);

return(0);

}

// Добавление нового узла в конец списка

int pushBack(cList\* list, elemtype\* data) {

cNode\* node = (cNode\*) malloc(sizeof(cNode));

if (!node) {

return(-3);

}

node->value = \*data;

node->next = NULL;

node->prev = list->tail;

if (!isEmptyList(list)) {

list->tail->next = node;

}

else {

list->head = node;

}

list->tail = node;

list->size++;

return(0);

}

// Извлечение узла из конца списка

int popBack(cList\* list, elemtype\* data) {

cNode \*node = NULL;

if (isEmptyList(list)) {

return(-4);

}

node = list->tail;

list->tail = list->tail->prev;

if (!isEmptyList(list)) {

list->tail->next = list->head;

}

else {

list->head = NULL;

}

\*data = node->value;

list->size--;

free(node);

return(0);

}

// Добавление нового узла в нужное место (index), начиная с 0 и заканчивая последним в списке.

int pushInside(cList\* list, elemtype\* data, int index) {

cNode\* newNode = (cNode\*)malloc(sizeof(cNode));

if (!newNode) {

return(-5);

}

if (index >= list->size || isEmptyList(list)) {

return(NULL);

}

cNode\* node;

if (index == 0) {

pushFront(list, data);

return 0;

}

else if (index == list->size-1) {

pushBack(list, data);

return 0;

}

else if (index < list->size / 2) {

int i = 0;

node = list->head;

while (node && i != index - 1)

{

node = node->next;

i++;

}

newNode->value = \*data;

newNode->next = node->next;

newNode->prev = node;

node->next->prev = newNode;

node->next = newNode;

}

else {

int i = list->size - 1;

cNode\* node = list->tail;

while (node && i != index)

{

node = node->prev;

i--;

}

newNode->value = \*data;

newNode->prev = node->prev;

newNode->next = node;

node->prev->next = newNode;

node->prev = newNode;

}

list->size++;

return(0);

}

// Извлечение узла из нужного места (index) в списке

int popInside(cList\* list, elemtype\* data, int index) {

cNode\* node;

if (isEmptyList(list)) {

return(-6);

}

if (index == 0) {

popFront(list, data);

return(0);

}

else if (index == list->size - 1) {

popBack(list, data);

return(0);

}

else if (index < list->size/2) {

int i = 0;

node = list->head;

while (node && i < index)

{

node = node->next;

i++;

}

}

else {

int i = list->size - 1;

node = list->head;

while (node && i > index)

{

node = node->prev;

i--;

}

}

node->prev->next = node->next;

node->next->prev = node->prev;

\*data = node->value;

list->size--;

free(node);

return(0);

}

// Чтение произвольного узла списка

cNode\* getNode(cList\* list, int index) {

cNode \*node = NULL;

int i;

if (index >= list->size) {

return (NULL);

}

if (index < list->size/2) {

i = 0;

node = list->head;

while (node && i < index) {

node = node->next;

i++;

}

}

else {

i = list->size - 1;

node = list->tail;

while (node && i > index) {

node = node->prev;

i--;

}

}

return node;

}

// Вывод списка в консоль

void printList(cList\* list, void (\*func)(elemtype\*)) {

cNode\* node = list->head;

if (isEmptyList(list)) {

return;

}

int i = 0;

while (node && i < list->size) {

func(&node->value);

node = node->next;

i++;

}

}

Завдання 3. Розробив програму обчислення виразів, поданих у оберненому польському записі з використанням дій додавання, віднімання, множення, ділення, піднесення до степеня та знаходження квадратного кореня.

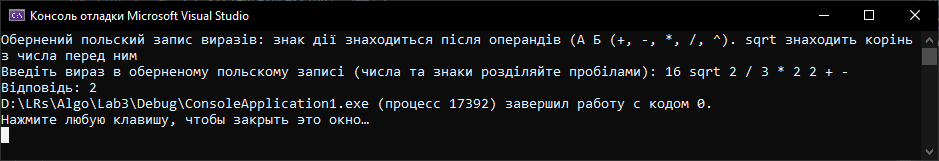


Рис.3.2 – демонстрація роботи обчислення

Приклад:

Лістинг програми:

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <string>

#include <string.h>

#include <D:\LRs\Algo\Lab3\Lab3\clist.h>

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Обернений польский запис виразів: знак дії знаходиться після операндів (А Б (+, -, \*, /, ^). sqrt знаходить корінь з числа перед ним\n");

printf("Введіть вираз в оберненому польскому записі (числа та знаки розділяйте пробілами): ");

char expression[50];

char num[20];

gets\_s(expression);

cList\* stack = createList();

int k = 0;

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

if (expression[i] == ' ' && i == 0) {

for (int j = 0; j < strlen(expression); j++) {

expression[j] = expression[j + 1];

}

i--;

continue;

}

if (expression[i] != ' ' && expression[i] != '+' && expression[i]

!= '-' && expression[i] != '\*' && expression[i] != '/' && expression[i] != 's' && expression[i] != '^') {

num[k] = expression[i];

k++;

}

else if (expression[i] == ' ' && expression[i - 1] != '+' && expression[i - 1] != '-'

&& expression[i - 1] != '\*' && expression[i - 1] != '/' && expression[i - 1] != 't' && expression[i - 1] != '^') {

num[k] = NULL;

int a = atoi(num);

pushBack(stack, &a);

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

num[j] = NULL;

}

k = 0;

}

else {

int tmp;

switch (expression[i])

{

case '+':

stack->tail->prev->value += stack->tail->value;

popBack(stack, &tmp);

break;

case '-':

stack->tail->prev->value -= stack->tail->value;

popBack(stack, &tmp);

break;

case '\*':

stack->tail->prev->value \*= stack->tail->value;

popBack(stack, &tmp);

break;

case '/':

stack->tail->prev->value /= stack->tail->value;

popBack(stack, &tmp);

break;

case '^':

stack->tail->prev->value = pow(stack->tail->prev->value, stack->tail->value);

popBack(stack, &tmp);

break;

case 's':

if (expression[i + 1] == 'q' && expression[i + 2] == 'r' && expression[i + 3] == 't') {

stack->tail->value = sqrt(stack->tail->value);

i += 4;

}

else {

do {

i++;

} while (expression[i] != ' ');

}

break;

default:

break;

}

}

}

popBack(stack, &k);

printf("Відповідь: %d", k);

}

***Висновок:*** ознайомився з основами роботи з двозв’язним та однозв’язними списками, стеком, чергою; розробив функції для обчислення арифметичного виразу у зворотному польському записі.