**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №2**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірила:

студент групи ІМ-31 Молчанова А. А.  
Литвиненко Сергій Андрійович  
номер у списку групи: 12

Київ 2024

**Завдання**

1. Створити список з **n** (**n** > 0) елементів (**n** вводиться з клавіатури), якщо iнша кiлькiсть елементiв не вказана у конкретному завданні за варiантом.
2. Тип ключiв (iнформацiйних полiв) задано за варiантом.
3. Вид списку (черга, стек, дек, прямий однозв’язний лiнiйний список, обернений однозв’язний лiнiйний список, двозв’язний лiнiйний список, однозв’язний кiльцевий список, двозв’язний кiльцевий список) вибрати самостiйно з метою найбiльш доцiльного розв’язку поставленої за варiантом задачi.
4. Створити функцiї (або процедури) для роботи зi списком (для створення, обробки, додавання чи видалення елементiв, виводу даних зi списку в консоль, звiльнення пам’ятi тощо).
5. Значення елементiв списку взяти самостiйно такими, щоб можна було продемонструвати коректнiсть роботи алгоритму програми. Введення значень елементiв списку можна виконати довiльним способом (випадковi числа, формування значень за формулою, введення з файлу чи з клавiатури).
6. Виконати над створеним списком дiї, вказанi за варiантом, та коректне звiльнення пам’ятi списку.
7. **При виконаннi заданих дiй, виводi значень елементiв та звiльненнi пам’ятi списку вважати, що довжина списку (кiлькiсть елементiв) невiдома на момент виконання цих дiй.** Тобто, не дозволяється зберiгати довжину списку як константу, змiнну чи додаткове поле.

При проєктуваннi програм слiд врахувати наступне:

1. при виконаннi завдання кiлькiсть операцiй (зокрема, операцiй читання й запису) має бути мiнiмiзованою, а також максимально мають використовуватися властивостi спискiв;
2. повторюванi частини алгоритму необхiдно оформити у виглядi процедур або функцiй (для створення, обробки, виведення та звiльнення пам’ятi спискiв) з передачею списку за допомогою параметра(iв).
3. у таких видiв спискiв, як черга, стек, дек функцiї для роботи зi списком мають забезпечувати роботу зi списком, вiдповiдну тому чи iншому виду списку (наприклад, не можна додавати новi елементи всередину черги);
4. програми мають бути написанi мовою програмування С.

**Варіант 12**

Ключами елементiв списку є цiлi числа. Обчислити значення виразу: , де – i-тий елемент списку.

**Текст програми**

1. Реалізація на основі двохзв’язного циклічного списку:

#include <stdlib.h>

#include <stdio.h>

typedef struct linkedList {

int data;

struct linkedList\* prev;

struct linkedList\* next;

} linkedList;

linkedList\* listInit() {

return NULL;

}

linkedList\* listPush(linkedList\* list, int data) {

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

if (node == NULL) return NULL;

if (list == NULL) {

\*node = (linkedList){ data, node, node };

return node;

}

linkedList\* tail = list->prev;

\*node = (linkedList){ data, tail, list };

tail->next = node;

list->prev = node;

return node;

}

void listDestroy(linkedList\* list) {

linkedList\* tail = list->prev;

linkedList\* prev = NULL;

while (tail != list) {

prev = tail->prev;

free(tail);

tail = prev;

}

free(list);

}

int calculate(linkedList\* list) {

linkedList\* tail = list->prev;

linkedList\* head = list;

int result = 1;

while (tail != list) {

result \*= (tail->data + tail->prev->data + 2 \* head->data);

tail = tail->prev;

head = head->next;

}

return result;

}

int main(int argc, char const \*argv[]) {

linkedList\* list = listInit();

int length, data, success = 1;

while (1) {

printf("Enter the count of elements you are going to enter: ");

scanf("%d", &length);

if (length > 1) break;

printf("The value must be greater than 1\n");

}

for (int i = 1; i <= length; i++) {

printf("Enter the element #%d: ", i);

scanf("%d", &data);

list = listPush(list, data);

if (list == NULL) {

printf("Error: There is not enough memory");

success = 0;

break;

}

}

if (success) {

printf("Result: %d\n", calculate(list));

}

listDestroy(list);

return 0;

}

1. Реалізація на основі двохзв’язного списку з заголовком:

#include <stdlib.h>

#include <stdio.h>

typedef struct node {

int data;

struct node\* previos;

struct node\* next;

} node;

typedef struct {

struct node\* first;

struct node\* last;

} linkedList;

linkedList\* listInit() {

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

if (list == NULL) return NULL;

\*list = (linkedList){ NULL, NULL };

return list;

}

node\* nodeInit(int data, node\* previos, node\* next) {

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

if (newNode == NULL) return NULL;

\*newNode = (node){ data, previos, next };

return newNode;

}

node\* listPush(linkedList\* list, int data) {

node\* newNode = nodeInit(data, list->last, NULL);

if (newNode == NULL) return NULL;

if (list->first == NULL) list->first = newNode;

else list->last->next = newNode;

list->last = newNode;

return newNode;

}

void listDestroy(linkedList\* list) {

node\* curent = list->first;

node\* next = NULL;

while (curent != NULL) {

next = curent->next;

free(curent);

curent = next;

}

free(list);

}

int calculate(linkedList\* list) {

int result = 1;

node\* first = list->first;

node\* last = list->last;

while (first->next != NULL) {

result \*= (first->data + first->next->data + 2 \* last->data);

first = first->next;

last = last->previos;

}

return result;

}

int main(int argc, char const \*argv[]) {

linkedList\* list = listInit();

if (list == NULL) {

printf("Error: There is not enough memory");

return 0;

}

int length, data, success = 1;

while (1) {

printf("Enter the count of elements you are going to enter: ");

scanf("%d", &length);

if (length > 1) break;

printf("The value must be greater than 1\n");

}

for (int i = 1; i <= length; i++) {

printf("Enter the element #%d: ", i);

scanf("%d", &data);

node\* item = listPush(list, data);

if (item == NULL) {

printf("Error: There is not enough memory");

success = 0;

break;

}

}

if (success) {

printf("Result: %d\n", calculate(list));

}

listDestroy(list);

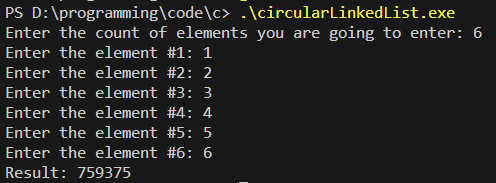
return 0;

}

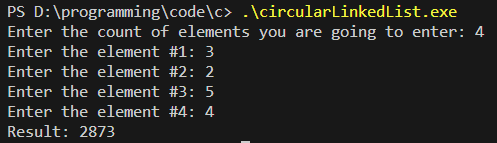
**Тестування програм**

Формула:

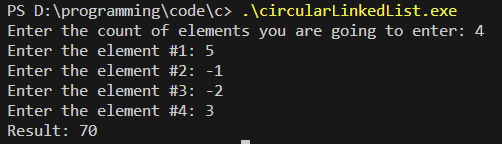
1. Реалізація на основі двохзв’язного циклічного списку:
2. (1, 2, 3, 4, 5, 6):



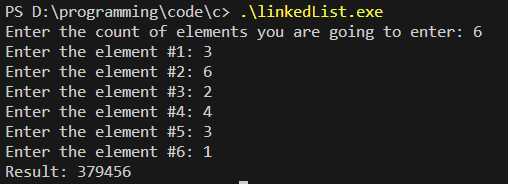
1. (3, 2, 5, 4):



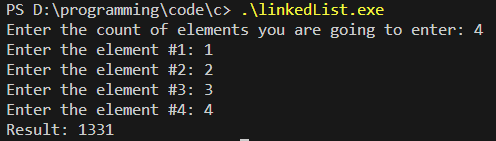
1. (5, -1, -2, 3):



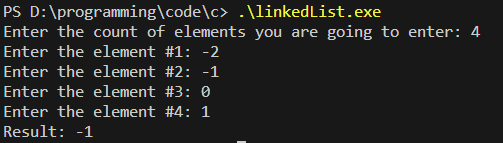
1. Реалізація на основі двохзв’язного списку з заголовком:
2. (3, 6, 2, 4, 3, 1):



1. (1, 2, 3, 4):



1. (-2, -1, 0, 1):



Результати збігаються, отже програми написані правильно.

**Висновок**

Протягом виконання лабораторної роботи я навчився розробляти зв’язані динамічні структури даних. Для вирішення своєї задачі я вибрав два типи списків: двохзв’язний циклічний список та двохзв’язного список з заголовком, адже мені потрібно мати доступ до сусідніх елементів в обидві сторони, та мати можливість звернутися до першого доданого елементу без циклу. Обидві ці структури задовільнять ці вимоги. Кожна реалізація має свої переваги та недоліки. Двохзв’язний циклічний список займає менше пам’яті, адже не потрібно зберігати окремо заголовок з двома вказівниками. З іншої сторони, у двохзв’язного списку з заголовком завжди є доступ до першого та останнього елементу, також за допомогою цього синтаксису можна зробити зручніший інтерфейс взаємодії зі списком. Проте, насправді, ці реалізації сильно не відрізняються ні по швидкодії ні по вимоги до пам’яті.

Не для всіх задач підходить статичний масив, адже не завжди ми можемо знати кількість даних на момент написання програми. Для вирішення цієї проблеми можна використовувати динамічні структури даних. Динамічний масив не завжди можна використовувати через низьку швидкість додавання та видалення нових елементів, також не завжди в пам’яті може знайтися безперервна вільна ділянка. Зв’язані структури даних вирішують обидві проблеми. Додавання та видалення елементів виконується за O(1), а самі дані не повинні зберігатися послідовно в пам’яті. Проте, такий спосіб зберігання даних має свої недоліки. Зв’язані структури даних займають більше пам’яті, адже окрім самих даних потрібно зберігати один або більше вказівників на наступні елементи. Також доступ до елементів списку відбувається за O(n).

Зв’язані структури даних слід використовувати коли значення часто додаються або видаляються зі списку, а доступ до даних всередині відбувається рідко.