Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗВІТ

До лабораторної роботи №2

З навчальної дисципліни “Алгоритми та структури даних”

Тема: Абстрактний тип даних “Cписок“

Підготував:

Студент групи СП-11

Гавриленко Артем

Тернопіль 2022

**Мета роботи:** набути навичок з реалізації АТД “Cписок”.

**Теоретичні відомості:**  
абстрактний тип даних список LIST можна реалізовувати у вигляді однозвязного та двозвязного списку.

У випадку одновзяний списку, стуктура, яка описує кожен елемент списку має вигляд:

**struct** celltype

elementtype element

celltype \*next

Для двозвязного списку, в якому містяться два вказівники на попередній та на наступний елементи, стуктура елементу має вигляд:



**struct** celltype

elementtype element

celltype \*next

celltype \*prev

**Завдання 1:**

Написати код, реалізації логіки роботи основних операцій однозвязного списку:

- MAKENULL (створює порожній список)

- END (повертає кінець списку)

- FIRST (повертає початок списку)

- INSERT (додає елемент до списку в задану позицію)

- DELETE (видаляє елемент з списку)

- LOCATE (знаходить позицію елементу в списку)

- RETRIEVE (повертає значення елементу списку)

- NEXT (повертає вказівник на наступний елемент списку)

**Лістинг програми:**

#include <iostream>

#include "Windows.h"

using namespace std;

struct Node;

template<typename T>

class List

{

public:

List();

void push\_back(T data);

void pop\_front();

void Clear();

void push\_front(T data);

void pop\_back();

int Get\_Size()

{

return size;

}

T RETRIEVE(const int index);

T LOCATE(T data);

Node\* END(int index);

Node\* FIRST();

T NEXT(int index);

void Delete(int index);

void INSERT(T data, int index);

void MAKENULL();

void printlist();

~List();

private:

template<typename T>

struct Node

{

public:

Node\* next;

T data;

Node(T data = T(), Node\* next = nullptr)

{

this->data = data;

this->next = next;

}

};

int size;

Node <T>\* head;

};

template<typename T>

List<T>::List()

{

size = 0;

this->MAKENULL();

}

template<typename T>

void List<T>::Clear()

{

while (size)

{

pop\_front();

}

}

template<typename T>

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node<T>(data);

}

else

{

Node<T>\* current = this->head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = new Node<T>(data);

}

size++;

}

template<typename T>

void List<T>::pop\_front()

{

Node<T>\* del = head;

head = head->next;

delete del;

size--;

}

template<typename T>

void List<T>::push\_front(T data)

{

head = new Node<T>(data, head);

}

template<typename T>

void List<T>::pop\_back()

{

Delete(size - 1);

}

template<typename T>

T List<T>::LOCATE(T data)

{

Node<T>\* current = this->head;

for (int i = 0; i < size; i++)

{

if (current->data == data)

{

return i;

}

current = current->next;

}

}

template<typename T>

Node\* List<T>::END(int index)

{

Node <T>\* current = this->head;

while (current != nullptr)

{

current = current->next;

}

return current;

}

template<typename T>

Node\* List<T>::FIRST()

{

return head;

}

template<typename T>

T List<T>::RETRIEVE(int index)

{

if (index > size - 1)

{

cout << "Елемент під індексом " << index << " не знайдено" << endl;

}

else

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->next;

counter++;

}

}

}

template<typename T>

T List<T>::NEXT(int index)

{

int nex = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (nex == index)

{

return current->next->data;

}

current = current->next;

nex++;

}

}

template<typename T>

void List<T>::Delete(int index)

{

if (index == 0)

{

pop\_front();

}

else

{

Node<T>\* current = this->head;

for (int i = 0; i < index - 1; i++)

{

current = current->next;

}

Node<T>\* del = current->next;

current->next = del->next;

delete del;

size--;

}

}

template<typename T>

void List<T>::INSERT(T data, int index)

{

if (index == 0)

{

push\_front(data);

}

else

{

Node<T>\* current = this->head;

for (int i = 0; i < index - 1; i++)

{

current = current->next;

}

Node<T>\* next\_put = new Node<int>(data, current->next);

current->next = next\_put;

size++;

}

}

template<typename T>

void List<T>::MAKENULL()

{

head = nullptr;

}

template<typename T>

void List<T>::printlist()

{

int Size = size;

Node<T>\* current = this->head;

cout << "Список: ";

while (Size)

{

cout << current->data << "; ";

current = current->next;

Size--;

}

cout << endl;

}

template<typename T>

List<T>::~List()

{

Clear();

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

List<int> a;

for (int i = 0; i < 6; i++)

{

a.push\_back(i + 1);

}

a.printlist();

cout << endl;

cout << "В списку " << a.Get\_Size() << " елементів " << endl << endl;

cout << "Вставлення в список елементів (7,2) і (9,5)" << endl;

a.INSERT(7, 2);

a.INSERT(9, 5);

a.printlist();

cout << endl;

cout << endl << "Видалення з списку елемента під індексом 2" << endl;

a.Delete(2);

a.printlist();

cout << endl;

cout << "Значення елемента під індексом 2: " << a.RETRIEVE(2);

cout << endl << endl;

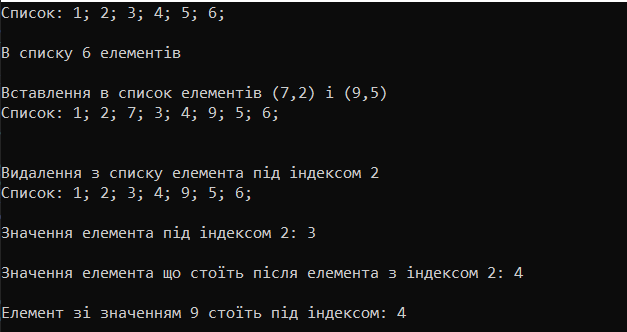
cout << "Значення елемента що стоїть після елемента з індексом 2: " << a.NEXT(2) << endl << endl;

cout << "Елемент зі значенням 9 стоїть під індексом: " << a.LOCATE(9) << endl;

return 0;

}

**Результат:**



**Завдання 2 і 3.**

2. Написати код, реалізації логіки роботи основних операцій двозвязного списку:

- MAKENULL (створює порожній список)

- END (повертає кінець списку)

- FIRST (повертає початок списку)

- INSERT (додає елемент до списку в задану позицію)

- DELETE (видаляє елемент з списку)

- LOCATE (знаходить позицію елементу в списку)

- RETRIEVE (повертає значення елементу списку)

- PREVIOUS (повертає вказівник на попередній елемент списку)

- NEXT (повертає вказівник на наступний елемент списку)

3. З використанням операцій АТД Список написати реалізацію функції PRINTLIST, яка друкує перелік усіх елементі, що є у списку.

**Лістинг програми:**

#include <iostream>

#include "Windows.h"

using namespace std;

struct Node;

template<typename T>

class List

{

public:

List();

void push\_back(T data);

void pop\_front();

void Clear();

void push\_front(T data);

void pop\_back();

int Get\_Size()

{

return size;

}

T RETRIEVE(int index);

T LOCATE(T data);

Node\* END(int index);

Node\* FIRST();

inline T PREVIOUS(int index);

T NEXT(int index);

void Delete(int index);

void INSERT(T data, int index);

void MAKENULL();

void printlist();

~List();

private:

template<typename T>

struct Node

{

public:

Node\* next;

Node\* prev;

T data;

Node(T data = T(), Node\* next = nullptr, Node\* prev=nullptr)

{

this->data = data;

this->next = next;

this->prev = prev;

}

};

int size;

Node <T>\* head;

Node<T>\* tail;

};

template<typename T> //

List<T>::List()

{

size = 0;

this->MAKENULL();

}

template<typename T>//

void List<T>::Clear()

{

while (size)

{

pop\_front();

}

}

template<typename T>//

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node<T>(data);

}

else

{

Node<T>\* current = this->head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = new Node<T>(data);

}

size++;

}

template<typename T>//

void List<T>::pop\_front()

{

Node<T>\* del = head;

head = head->next;

delete del;

size--;

}

template<typename T>//

void List<T>::push\_front(T data)

{

head = new Node<T>(data, head);

}

template<typename T>//

void List<T>::pop\_back()

{

Delete(size - 1);

}

template<typename T>//

T List<T>::LOCATE(T data)

{

Node<T>\* current = this->head;

int aa =0 ;

for (int i = 0; i < size; i++)

{

if (current->data == data)

{

return aa;

}

current = current->next;

aa++;

}

}

template<typename T> //

Node\* List<T>::END(int index)

{

return tail;

}

template<typename T>//

Node\* List<T>::FIRST()

{

return head;

}

template<typename T> //

inline T List<T>::PREVIOUS(int index)

{

if (index < size/2)

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->prev->data;

}

current = current->next;

counter++;

}

}

else

{

int counter = 0;

Node<T>\* current = this->tail;

while (current != nullptr)

{

if (counter == index)

{

return current->prev->data;

}

current = current->prev;

counter--;

}

}

}

template<typename T>//

T List<T>::RETRIEVE(int index)

{

if (index > size - 1)

{

cout << "Елемент під індексом " << index << " не знайдено" << endl;

}

else

{

if (index < size/2)

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->next;

counter++;

}

}

else

{

int counter = size;

Node<T>\* current = this->tail;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->prev;

counter--;

}

}

}

}

template<typename T>

T List<T>::NEXT(int index)

{

if (index > size - 1)

{

cout << "Елемент під індексом " << index << " не знайдено" << endl;

}

else

{

if (index < size / 2)

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->next->data;

}

current = current->next;

counter++;

}

}

else

{

int counter = size;

Node<T>\* current = this->tail;

while (current != nullptr)

{

if (counter == index)

{

return current->next->data;

}

current = current->prev;

counter--;

}

}

}

}

template<typename T> //

void List<T>::Delete(int index)

{

if (index == 0)

{

pop\_front();

}

else

{

if (index < size/2)

{

Node<T>\* current = this->head;

for (int i = 0; i < index - 1; i++)

{

current = current->next;

}

Node<T>\* del = current->next;

current->next = del->next;

delete del;

}

else

{

Node<T>\* current = this->tail;

for (int i = size-1; i > index; i--)

{

current = current->prev;

}

Node<T>\* del = current->next;

current->next = del->next;

delete del;

}

size--;

}

}

template<typename T> //

void List<T>::INSERT(T data, int index)

{

if (index == 0)

{

push\_front(data);

}

else

{

if (index < size/2)

{

Node<T>\* current = this->head;

for (int i = 0; i < index - 1; i++)

{

current = current->next;

}

Node<T>\* next\_put = new Node<T>(data, current->next, current->prev);

current->next = next\_put;

current->prev = next\_put->prev;

size++;

}

else

{

Node<T>\* current = this->tail;

for (int i = size; i > index - 1; i--)

{

current = current->prev;

}

Node<T>\* next\_put = new Node<T>(data, current->next, current->prev);

current->next = next\_put;

current->prev = next\_put->prev;

size++;

}

}

}

template<typename T> //

void List<T>::MAKENULL()

{

head = nullptr;

tail = nullptr;

}

template<typename T>//

void List<T>::printlist()

{

int Size = size;

Node<T>\* out = this->head;

cout << "Список: ";

while (Size)

{

cout << out->data << "; ";

out = out->next;

Size--;

}

cout << endl;

}

template<typename T>//

List<T>::~List()

{

Clear();

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

List<int> a;

for (int i = 0; i < 6; i++)

{

a.push\_back(i + 1);

}

a.printlist();

cout << endl;

cout << "В списку " << a.Get\_Size() << " елементів " << endl << endl;

cout << "Вставлення в список елементів (7,2) і (9,5)" << endl;

a.INSERT(7, 2);

a.printlist();

cout << endl;

cout << endl << "Видалення з списку елемента під індексом 2" << endl;

a.Delete(2);

a.printlist();

cout << endl;

cout << "Значення елемента під індексом 2: " << a.RETRIEVE(2);

cout << endl << endl;

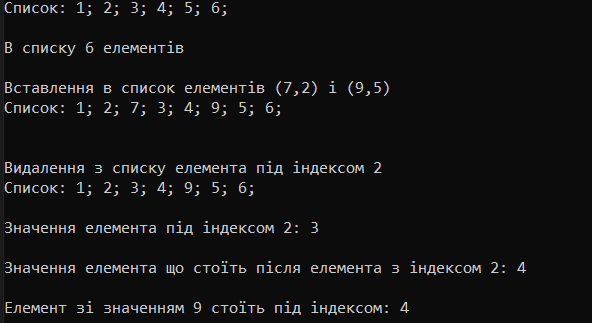
cout << "Значення елемента що стоїть після елемента з індексом 2: " << a.NEXT(2) << endl << endl;

cout << "Елемент зі значенням 9 стоїть під індексом: " << a.LOCATE(9) << endl;

return 0;

}

**Результат:**



**Висновки:**

-Набуті навички з реалізації АТД “Cписок”.