**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 13**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Арепьев В.О. |
| Преподаватель |  | Родионова Е.А. |

Санкт-Петербург

2022

Постановка задачи

Вариант 13

Задача: реализовать объект в виде односвязного списка с набором методов/функций. Данные, хранящиеся в списке, могут быть любого типа.

Реализовано на Visual Studio 2017.

**Классы:**

|  |  |
| --- | --- |
| Node | Элемент списка. Состоит из value (значения элемента, тип int) и next, указывающего на следующий элемент |
| List | Односвязный список. Состоит из head типа Node, указывающий на первый элемент списка, pNext, указывающий на следующий элемент списка и size – размер списка. |

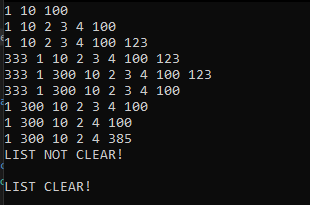
**Методы List:**

|  |  |
| --- | --- |
| push\_back(T data) | Создание списка |
| GetSize() | Текущий размер |
| delete\_indx(const int index) | Удаляет элемент по индексу |
| add\_indx(const int index, T data) | Добавляет элемент по индексу |
| add\_end(T data) | Добавляет элемент в конец |
| delete\_end() | Удаляет элемент в конце |
| add\_head(T data) | Добавляет элемент перед началом списка |
| delete\_head() | Удаляет первый элемент в списке |
| clear() | Удаление всего списка |
| shift\_element\_indx(const int index, T data) | Меняет элемент по индексу, на заданный элемент |
| add\_list\_indx(const int index, List list) | Добавляет список в список, начиная с индекса |
| void\_check() | Проверяет список на пустоту |
| operator[](const int index) | Выводит значение по индексу |

Оценка временной сложности методов

|  |  |
| --- | --- |
| void\_check() | O(1) |
| GetSize() | O(1) |
| operator[](const int index) | O(n) |
| add\_head(T data) | O(1) |
| add\_end(T data) | O(n) |
| delete\_indx(const int index) | O(n) |
| delete\_end() | O(n) |
| delete\_head() | O(1) |
| shift\_element\_indx(const int index, T data) | O(n) |
| clear() | O(n) |
| find\_last\_list\_entrance(li: LinkedList) | O(n) |
| push\_back(T data) | O(1) |
| add\_indx(const int index, T data) | O(n) |
| add\_list\_indx(const int index, List list) | O(n) |

Пример работы



Листинг

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

template <typename T>

class List

{

public:

List();

~List();

void push\_back(T data);

int GetSize() { return Size; }

void delete\_indx(const int index);

void add\_indx(const int index, T data);

void add\_end(T data);

void delete\_end();

void add\_head(T data);

void delete\_head();

void clear();

void shift\_element\_indx(const int index, T data);

void add\_list\_indx(const int index, List list);

void void\_check();

T& operator[](const int index);

private:

template <typename T>

class Node

{

public:

Node \*pNext;

T data;

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

{

this->data = data;

this->pNext = pNext;

}

};

int Size;

Node<T> \*head;

};

template <typename T>

List<T>::List()

{

Size = 0;

head = nullptr;

}

template <typename T>

List<T>::~List()

{

}

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->pNext != nullptr)

{

current = current->pNext;

}

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

}

Size++;

}

template<typename T>

void List<T>::delete\_indx(const int index)

{

Node<T> \*current = head;

Node<T> \*temp;

if (GetSize() - 1 <= index)

{

delete\_end();

}

else if (index == 0)

{

delete\_head();

}

else

{

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

{

current = current->pNext;

}

temp = current->pNext;

current->pNext = current->pNext->pNext;

delete temp;

Size--;

}

}

template<typename T>

void List<T>::add\_indx(const int index, T data)

{

Node<T> \*current = head;

Node<T> \*temp;

if (GetSize() < index)

{

add\_end(data);

}

else if (index == 0)

{

add\_head(data);

}

else

{

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

{

current = current->pNext;

}

temp = current->pNext;

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

current = current->pNext;

current->pNext = temp;

Size++;

}

}

template<typename T>

void List<T>::add\_end(T data)

{

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

while (current->pNext != nullptr)

{

current = current->pNext;

}

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

Size++;

}

template<typename T>

void List<T>::delete\_end()

{

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

while (current->pNext != nullptr)

{

current = current->pNext;

}

delete current;

Size--;

}

template<typename T>

void List<T>::add\_head(T data)

{

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

head = new Node<T>(data);

head->pNext = temp;

Size++;

}

template<typename T>

void List<T>::delete\_head()

{

Node<T> \*temp = head;

head = head->pNext;

delete temp;

Size--;

}

template<typename T>

void List<T>::clear()

{

while (Size)

{

delete\_head();

}

}

template<typename T>

void List<T>::shift\_element\_indx(const int index, T data)

{

Node<T> \*current = head;

if (index > GetSize())

{

add\_end(data);

}

else if (index == 0)

{

head->data = data;

}

else

{

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

{

current = current->pNext;

}

current->data = data;

}

}

template<typename T>

void List<T>::add\_list\_indx(const int index, List list)

{

Node<T> \*temp;

Node<T> \*current = head;

if (index == 0)

{

temp = head;

head = list.head;

current = list.head;

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = temp;

}

else

{

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

{

current = current->pNext;

}

temp = current->pNext;

current->pNext = list.head;

current = list.head;

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = temp;

}

Size = Size + list.GetSize();

}

template<typename T>

void List<T>::void\_check()

{

if (GetSize() == 0)

{

cout << "LIST CLEAR!" << endl;

}

else

{

cout << "LIST NOT CLEAR!" << endl;

}

}

template<typename T>

T & List<T>::operator[](const int index)

{

int counter = 0;

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

while (current != nullptr)

{

if (counter == index)

{

return current->data;

break;

}

current = current->pNext;

counter++;

}

}

int main()

{

setlocale(LC\_ALL, "ru");

List<int> lst;

List<int> lst\_2;

lst.push\_back(1);

lst.push\_back(10);

lst.push\_back(100);

lst\_2.push\_back(2);

lst\_2.push\_back(3);

lst\_2.push\_back(4);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.add\_list\_indx(2, lst\_2);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.add\_end(123);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.add\_head(333);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.add\_indx(1, 300);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.delete\_end();

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.delete\_head();

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.delete\_indx(3);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.shift\_element\_indx(5, 385);

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.void\_check();

lst.clear();

for (int i = 0; i < lst.GetSize(); i++)

{

cout << lst[i] << ' ';

}

cout << endl;

lst.void\_check();

return 0;

}