Файл Iterator.h

#pragma once

template<class T>

class Iterator

{

protected:

//флаг, показывающий, достиг ли итератор концв списка

//должен поддерживаться производными классами

int iterationComplete;

public:

//конструктор

Iterator() : iterationComplete{ 0 }

{}

//обязательные методы итератора

virtual void next() = 0;

virtual void reset() = 0;

//методы выборки/модификации данных

virtual T& data() = 0;

//проверка конца списка

virtual int endOfList() const {

return iterationComplete;

}

};

Файл List.h

#pragma once

template<class T>

class List

{

//число элементов списка, обновляемое производным классом

int size;

public:

//конструктор устанавливает size в 0

List()

{

this->size = 0;

}

//веврнуть размер списка

int getSize() {

return this->size;

}

//проверить, пуст ли список

int listEmpty() {

return this->size == 0;

}

//найти

virtual int find(T& item) = 0;

//вставить элемент

virtual void insertItem(const T& item) = 0;

//удалить элемент

virtual void deleteItem(const T& item) = 0;

//очистить список

virtual void clearList() = 0;

};

Файл Node.h

#pragma once

template <typename T>

class Node // шаблонный класс узла списка

{

private:

// данные узла списка

T data;

// указатель на пред. узел

Node\* prev;

// указатель на след. узел

Node\* next;

public:

// конструктор по умолчанию

Node()

{

set\_prev(nullptr);

set\_next(nullptr);

}

// конструктор по умолчанию

Node(const T& temp)

{

set\_data(temp);

set\_prev(nullptr);

set\_next(nullptr);

}

// деструктор

~Node()

{

if (next != nullptr) // пока не дошли до последнего элемента

{

delete next; // удаляем след. элемент

next = nullptr; // на всякий случай (чтобы случайно не

использовать рандомную память)

prev = nullptr; // та же причина

}

}

// устанавливаем данные (по ссылке, для экономии памяти)

void set\_data(const T& temp)

{

data = temp;

}

// возвращаем данные

T& get\_data()

{

return data;

}

// устанавливаем указатель на предыдущий узел

void set\_prev(Node\* temp)

{

prev = temp;

}

// возвращаем указатель на предыдущий узел

Node\* &get\_prev()

{

return prev;

}

// устанавливаем указатель на следующий узел

void set\_next(Node\* temp)

{

next = temp;

}

// возвращаем указатель на следующий узел

Node\* &get\_next()

{

return next;

}

};

Файл List2.h

#pragma once

#include "Node.h" // шаблонный класс узла списка

template <class T>

class List2 // шаблонный класс для создания списка

{ // данные методы используются другими, пользователю не нужен к ним доступ, но я использую их в LinkedList

private:

// добавление головы списка

void add\_first(Node<T>\*& h, T& temp)

{

h = new Node<T>(temp); // выделяем память под h = узлу списка.

Заполняем поле data = temp

}

// добавление эллемета, усли голова уже есть

void add\_end(Node<T>\*& h, T& temp)

{

Node<T>\* ptr = h; // чтобы head не поменялся

while (ptr->get\_next() != nullptr) // доходим до последнего эллемента

списка

{

ptr = ptr->get\_next();

}

Node<T>\* t = new Node<T>(temp); /\* создаём следующей узел

инициализируем указатель на шаблонный класс узла

выделяем под него память

в поле data записываем temp \*/

t->set\_prev(ptr); // для нового узла предыдущий –

последний

ptr->set\_next(t); // для последнего следующий –

новый

}

public:

// добавление элемента в список

void add(Node<T>\* &h, T& temp)

{

if (h == nullptr) // если список пуст, то

заполняем первый элемент

{

add\_first(h, temp);

}

else // если список не пуст

добавляем в конец

{

add\_end(h, temp);

}

}

// очистка списка

void del(Node<T>\*& h)

{

delete h; // вызовется деструктор

Node

}

};

Файл List2.h

#pragma once

#include "List2.h" // класс для создания очереди

#include "Node.h" // шаблонный класс узла списка

#include <time.h> // для рандома

#include <iostream> // ввод/вывод

template <typename T> /\*шаблонный класс для работы со списками

тк класс наследуется от Node и List в нём

будет весь их функционал\*/

class LinkedList : public Node<T>, public List2<T>

{

private:

// указатель на головной узел

Node<T>\* head;

public:

// конструктор по умолчанию

LinkedList() {

set\_head(nullptr);

}

// конструктор с указателем на голову

LinkedList(Node<T>\*& h) {

set\_head(h);

}

//вставка в конце

void insertRear(T& temp)

{

if (this->head != nullptr) {

Node<T>\* tempPtr = new Node<T>(temp);

tempPtr->set\_next(this->head);

this->head->set\_prev(tempPtr);

this->head = tempPtr;

}

else {

this->head = new Node<T>(temp);

}

}

// переопределение (запоминаем head)

void add(Node<T>\*& h, T& temp)

{

if (h == nullptr) //если список пуст, то заполняем

первый элемент

{

this->add\_first(h, temp); //добавляем в список (тк метод protected

можно его использовать в наследнике, но

нельзя в объектах)

}

else //если список не пуст добавляем в конец

{

this->add\_end(h, temp); //добавляем в список (тк метод protected

можно его использовать в наследнике, но

нельзя в объектах)

}

if (this->head == nullptr) //если голова списка не установлена

set\_head(searc\_head(h)); //ищем и запоминаем первый элемент (голова

списка)

}

// получение указателя на голову

Node<T>\*& get\_head()

{

return head;

}

// установка указателя на голову

void set\_head(Node<T>\* h)

{

head = h;

}

// сортировка списка по возрастанию

void sort()

{

if (empty()) // проверка очереди на пустоту

throw std::invalid\_argument("List is empty");

Node<T>\* temp; // указатель на текущий узел

Node<T>\* a; // указатель на следующего узел

bool flag = true;

while (flag) // пока порядок элементов списка меняется

(пока не отсортирован)

{

temp = head; // текущий узел

a = temp->get\_next(); // следующий узел

flag = false;

while (a) // пока не дошли до конца

{

if (temp->get\_data() > a->get\_data())

{ // если надо - меняем данные

в узлах местами

swap(temp, a); // меняем значения узлов

местами

flag = true; // запоминаем, что изменяли

порядок

}

temp = temp->get\_next(); // переходим к следующего

узлам

a = a->get\_next();

}

}

}

// престановка данных в узлах \_1, \_2 местами

void swap(Node<T>\*& \_1, Node<T>\*& \_2)

{

T temp = \_1->get\_data(); // тут всё стандартно

\_1->set\_data(\_2->get\_data());

\_2->set\_data(temp);

}

// удаление ptr - узла списка

Node<T>\* del\_ellement(Node<T>\*& ptr)

{

// a / ptr / b - фрагмент узла, для наглядности

// нужно соеденить a и b и очистить память ptr

Node<T>\* a = ptr->get\_prev(); // эллемент перед ptr

Node<T>\* b = ptr->get\_next(); // эллемент после ptr

if (a)

a->set\_next(b);

else // если хотим удалить головной узел

set\_head(b); // указатель на голову переходит к

следующего узлу

if (b) // если удаляем любой, кроме

последнего узла

b->set\_prev(a);

return b; // возвращаем указатель на узел,

следующий за удалённым

ptr->set\_next(nullptr); // отделяем удаляемый узел от списка

ptr->set\_prev(nullptr);

delete ptr; // очищаем память

}

// удаление дубликатов

void remove\_dupl()

{

if (empty()) // проверка на пустоту списка

throw std::invalid\_argument("List is empty");

Node<T>\* ptr = get\_head();

bool b = false; // флаг удаления узла

while (ptr->get\_next() != nullptr) // проверка следующего узла на

существование

{

b = false; // обнуление флага

if (ptr->get\_data() == (ptr->get\_next())->get\_data())

{

del\_ellement(ptr->get\_next()); // удаление узла

b = true; // запоминаем, что удаляли узел

}

if (!b) // если удаления не было

ptr = ptr->get\_next(); // ptr переходит на следующего

узел, иначе остаётся на прежнем

}

}

// поиск головы списка ptr - указатель на любой узел списка

Node<int>\* searc\_head(Node<T>\*& ptr)

{

if (ptr == nullptr)

throw std::invalid\_argument("NULL pointer");

Node<T>\* h = ptr; // чтобы ptr не стал указывать на голову (чтобы не

менялся)

while (h->get\_prev()) // идём до первого элемента (головы списка)

{

h = h->get\_prev();

}

return h; // возвращаем указатель на голову

}

//добавление n-го кол-ва узлов случайными числами

//возвращение указателя на голову списка

void add(int n)

{

int r = 0; // случайное число

srand(time(NULL)); // для рандома

for (int i = 0; i < n; i++) // n раз добавляем новый узел

{

r = (-25 + rand() % (50 - 0 + 1)); // рандом [-25 ; 25]

this->add(head, r);

}

}

// проверка на пустоту списка (true, если пусто)

int empty()

{

if (head == nullptr)

return 1;

else

return 0;

}

};

Файл SeqList.h

#pragma once

#include"List.h"

#include"LinkedList/LinkedList.h"

template<class T>

class SeqListIterator;

template<class T>

class SeqList{

protected:

int size;

LinkedList<T> llist;

public:

//конструктор

SeqList() {

llist = LinkedList<T>();

size = 0;

}

//поиск

int find(const T& item) {

Node<T>\* temp = llist.get\_head();

if (temp == nullptr) //если голова пустая

return 0;

if (temp->get\_data() == item) // если голова является нужным объектом

return 1;

else {

while (temp->get\_next() != nullptr) {

temp = temp->get\_next();

if (temp->get\_data() == item)

return 1;

}

return 0;

}

}

//получение данных

T getData(const int& pos) {

if (pos > this->size)

return;

else {

Node<T> tempPtr = llist.get\_head();

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

tempPtr = tempPtr->get\_next();

return tempPtr->get\_data();

}

}

//вставка

void insertItem(T& item) {

llist.insertRear(item);

this->size++;

}

//удалить элемент

void deleteItem(const T& item) {

// a / ptr / b - фрагмент узла, для наглядности

// нужно соеденить a и b и очистить память ptr

Node<T>\* ptr;

while (ptr->get\_data() != item) {

if (ptr->get\_next != nullptr)

ptr->get\_next();

else {

std::cerr << "Error: item is not found!";

exit(1);

}

}

Node<T>\* a = ptr->get\_prev(); // элемент перед ptr

Node<T>\* b = ptr->get\_next(); // элемент после ptr

if (a)

a->set\_next(b);

else // если хотим удалить головной узел

llist.set\_head(b); // указатель на голову переходит к следующему

узлу

if (b) // если удаляем любой, кроме последнего узла

b->set\_prev(a);

return b; // возвращаем указатель на узел, следующий за

удалённым

ptr->set\_next(nullptr); // отделяем удаляемый узел от списка

ptr->set\_prev(nullptr);

delete ptr; // очищаем память

this->size--;

}

//очистить список

void clearList() {

Node<T>\* temp = llist.get\_head();

while (!llist.empty()) {

if (temp->get\_next()->get\_next() != nullptr) {

temp = temp->get\_next();

}

else {

llist.del\_ellement(temp->get\_next());

temp = temp->get\_prev();

}

if (temp == llist.get\_head()) {

llist.del\_ellement(temp);

}

}

this->size = 0;

}

//удалить начало

T deleteFront() {

Node<T>\* tempPtr = llist.get\_head();

T temp;

llist.set\_head() = tempPtr->get\_next();

temp = tempPtr->get\_data();

delete tempPtr;

this->size--;

return temp;

}

friend class SeqListIterator<T>;

};

Файл SeqListIterator.h

#pragma once

#include <iostream>

#include "SeqList.h"

#include "Iterator.h"

#include "LinkedList/Node.h"

template <class T>

class SeqListIterator: public Iterator<T> {

SeqList<T>\* listPtr; //локальный указатель на объект SeqList

Node<T>\* prevPtr, \*currPtr; //предыдущая и текущая позиция

public:

//конструктор

SeqListIterator(SeqList<T>& lst) {

this->setList(lst);

}

//продвинуться к следующему элементу списка

virtual void next(){

if (currPtr == nullptr) //currPtr == nullptr, мы в конце

списка

return;

this->prevPtr = this->currPtr; //передвинуть указатель prevPtr на

один узел вперед

this->currPtr = this->currPtr->get\_next();//currPtr на один узел вперед

if (this->currPtr == nullptr) //если обнаружен конец связанного

списка

this->iterationComplete = 1; //установить флаг "итерация завершена"

}

//переход к началу списка

virtual void reset() {

this->iterationComplete = this->listPtr->llist.empty();//переприсвоить

состояние итерации

if (this->listPtr->llist.get\_head() == nullptr) //вернуться если список

пуст

return;

this->prevPtr = nullptr; //установить механизмы

прохождения списка с первого узла

this->currPtr = this->listPtr->llist.get\_head();

}

//возвратить данные, расположенные в текущем элементе списка

virtual T& data() {

if ((this->listPtr->llist.empty()) || (this->currPtr == nullptr))//если список

пуст или прохождение уже

завершено

{

std::cerr << "Error: недопустимая ссылка!"; //возвращаем ошибку

exit(1);

}

return this->currPtr->get\_data();

}

//сейчас итератор должен проходить список lst

//переназначте listPtr и вызовите reset

void setList(SeqList<T> &lst) {

this->listPtr = &lst;

reset(); //инициализировать механизм прохождения для списка lst

}

};

Файл labaIterator.cpp

#include <iostream>

#include <fstream>

#include "SeqList.h"

#include "SeqListIterator.h"

using namespace std;

struct SalesPerson {

int idno; //личный номер продовца

int units; //количестао проданного товара

};

//оператор == сравнивает служащих по их личному номеру

int operator == (const SalesPerson& a, const SalesPerson& b) {

return a.idno == b.idno;

}

//взять id в качестве ключа и пройти список

//суммировать количество товара, проданного определенным сотрудником

//распечатать результат

void printTotalSales(SeqList<SalesPerson>& l, int id) {

SalesPerson salesP{ id,0 }; //объявление переменной типа SalesPerson и инициализация

поля записи

SeqListIterator<SalesPerson> iter(l); //объявление итератора

последовательного списка

//использование его для прохождения

списка

for (iter.reset(); !iter.endOfList(); iter.next())

if (iter.data() == salesP) //если происходит совпадение id,

прибавить количество товара

salesP.units += (iter.data()).units;

cout << "Служащий: " << salesP.idno //печать

<< "Количество проданного товара: " << salesP.units << endl;

}

int main()

{

setlocale(0, "ru"); //руссификация

SeqList<SalesPerson> salesList; //список, содержащий записи типа SalesPerson

SeqList<int> idList; //список личных номеров сотрудников

ifstream salesFile; //входной файл

SalesPerson salesP; //переменная для ввода

salesFile.open("sales.dat", ios::in| ios::\_Nocreate); //открыть входной файл

if (!salesFile) {

cerr << "Файле sales.dat не найден!";

exit(1);

}

while (!salesFile.eof()) { //чтение данных в форме "личный номер количество

товара"

salesFile >> salesP.idno >> salesP.units; //ввести поля данных и вставитт в список

salesList

salesList.insertItem(salesP);

if (!idList.find(salesP.idno)) //если id отсутствует в idList, включить этот id

idList.insertItem(salesP.idno);

}

SeqListIterator<int> idIter(idList);//создание итератора

for (idIter.reset(); !idIter.endOfList(); idIter.next()) //сканировать список личных

номеров и

printTotalSales(salesList, idIter.data()); //передовать каждый номер в

функцию printTotalSales

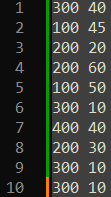
//для добавления количества

проданного товара к общему

//числу его продаж

}

Файл sales.dat



Вывод

