ДЗ на 25.04.2023

0)Написать метод очистки списка (освобождения памяти)

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class List

{

struct Item

{

T inf;

Item\* next;

Item(T x, Item\* ref = nullptr) : inf(x), next(ref) {}

} \*first = nullptr, \* last = nullptr;

public:

void udalyaem() {

while (first != nullptr) {

Item\* tmp = first;

first = first->next;

delete tmp;

}

last = nullptr;

}

bool empty() {

return first == 0;

}

void add(T x)

{

Item\* item = new Item(x);

if (last != nullptr)

last->next = item;

last = item;

if (first == nullptr)

first = last;

}

void print()

{

for (Item\* t = first; t != nullptr; t = t->next)

cout << t->inf << " ";

cout << endl;

}

void duplicateEven()

{

for (Item\* t = first; t != nullptr; t = t->next)

{

if (t->inf % 2 == 0)

{

Item\* newItem = new Item(t->inf, t->next);

t->next = newItem;

}

}

}

~List() {

while (first != nullptr) {

this -> udalyaem();

}

}

};

int main()

{

List<int> lst;

lst.add(1);

lst.add(2);

lst.add(3);

lst.add(4);

lst.add(5);

lst.print();

lst.udalyaem();

lst.print();

lst.~List();

return 0;

}

1)Чего не хватает в методе duplicateEven (два момента). Исправьте, чтобы работал как задумано: вставлял после каждого чётного ещё один такой же.

void duplicateEven()

{

Item\* t = first;

while (t != nullptr)

{

if (t->inf % 2 == 0)

{

Item\* newItem = new Item(t->inf, t->next);

t->next = newItem;

if (t == last) {

last = newItem;

}

t = t->next;

}

t = t->next;

}

}

2) Создать конструктор копирования списка (сигнатура List(const List &other)).

List() : first(nullptr), last(nullptr){}

List(const List& other) {

this->add(other.first->inf);

Item\* t = other.first;

t = t->next;

while (t != nullptr) {

this->add(t->inf);

t = t->next;

}

}

3)Создать класс SimpleVector (на основе обычного динамического массива), который умеет делать добавление элемента в конец (push\_back) и перегружает оператор индексации [] для взятия элемента по номеру.

4)Научить свой класс SimpleVector выделять в два раза больше памяти (и переносить данные), если при выполнении push\_back не хватило места. Также дописать деструктор, освобождающий память.

template <typename T>

class SimpleVector {

private:

T\* data;

int size;

int capacity;

public:

SimpleVector() {

data = nullptr;

size = 0;

capacity = 0;

}

void push\_back(T value) {

if (size + 1 < capacity) {

data[size] = value;

size++;

}

else if (size == 0) {

data = new T[capacity];

\*data = value;

size++;

}

else {

capacity \*= 2;

T\* temp = new T[capacity];

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

\*(temp + i) = \*(data + i);

}

\*(temp + size) = value;

delete[]data;

data = temp;

size++;

}

}

T& operator[](int index) {

return data[index];

}

~SimpleVector() {

if (size >= 0) {

delete[]data;

}

}

};