Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа  
“Классы”. №6**

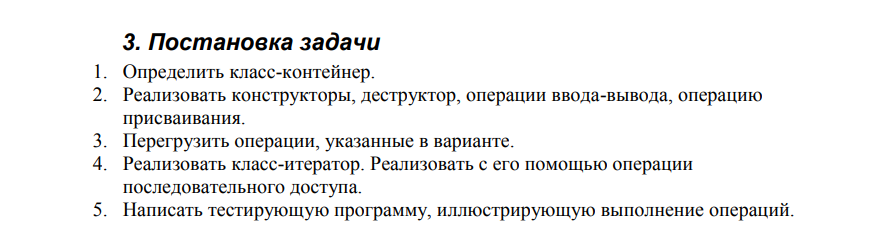
Выполнил:   
студент группы ИВТ-23-2б   
Чудинов Данил Николаевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

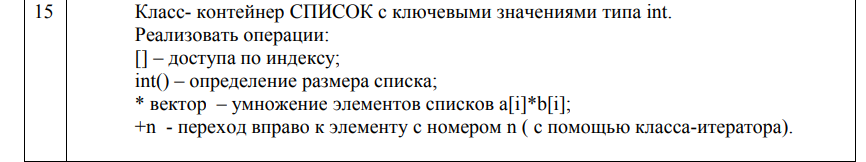
Пермь, 2024 г.

**«Классы и объекты. Принцип подстановки»**

**Анализ задачи:**

****

***Вариант 15:***

****

**Код на языке C++:**

#include <iostream>

#include <string>

using namespace std;

struct Node {

int data;

Node\* next = nullptr;

Node\* prev = nullptr;

};

class Iterator {

private:

friend class List;

Node\* elem;

public:

Iterator() {

this->elem = nullptr;

}

Iterator(const Iterator& it) {

this->elem = it.elem;

}

bool operator == (const Iterator& iter) {

return this->elem = iter.elem;

}

bool operator != (const Iterator& iter) {

return this->elem != iter.elem;

}

Iterator& operator =(const Iterator& iter) {

this->elem = iter.elem;

return \*this;

}

Iterator& operator ++ () {

this->elem = elem->next;

return \*this;

}

Iterator& operator --() {

this->elem = elem->prev;

return \*this;

}

Iterator operator + (const int number) {

Iterator temp(\*this);

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

temp.elem = temp.elem->next;

}

return temp;

}

int operator \* () {

return this->elem->data;

}

};

class List {

private:

int size;

Node\* head;

Node\* tail;

Iterator begin;

Iterator end;

public:

void pushFront(int data) {

Node\* newNode = new Node;

newNode->data = data;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

this->begin.elem = this->head;

this->end.elem = this->tail;

}

else {

head->prev = newNode;

newNode->next = head;

head = newNode;

}

this->size++;

}

void pushBack(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

this->begin.elem = this->head;

this->end.elem = this->tail;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

this->size++;

}

List(int size) {

this->size = size;

if (size > 0) {

Node\* node = new Node;

this->head = node;

this->tail = node;

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

Node\* newNode = new Node;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

List(int size, int data) {

this->size = size;

if (size > 0) {

Node\* node = new Node;

node->data = data;

this->head = node;

this->tail = node;

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

Node\* newNode = new Node;

newNode->data = data;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

List(const List& list) {

this->head = nullptr;

this->tail = nullptr;

this->size = 0;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

pushBack(current\_node->data);

current\_node = current\_node->next;

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

~List() {

Node\* node = head;

while (node != nullptr) {

Node\* newNode = node->next;

delete node;

node = newNode;

}

head = nullptr;

}

int popBack() {

int temp = 0;

if (this->tail != nullptr) {

Node\* current\_node = this->tail;

tail = current\_node->prev;

temp = current\_node->data;

tail->next = nullptr;

this->size = size - 1;

this->end.elem = this->tail;

delete current\_node;

}

return temp;

}

int popFront() {

int temp = 0;

if (this->head != nullptr) {

Node\* current\_node = this->head;

head = current\_node->next;

temp = current\_node->data;

if (head) {

head->prev = nullptr;

}

delete current\_node;

this->begin.elem = this->head;

this->size = size - 1;

}

return temp;

}

int front() {

return this->head->data;

}

int back() {

return this->tail->data;

}

bool is\_empty() {

return this->size == 0;

}

List& operator =(const List& list) {

cout << "Вызвана операция присваивания: " << endl;

if (this == &list) {

return \*this;

}

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

this->size = 0;

Node\* current = list.head;

while (current != nullptr) {

pushBack(current->data);

current = current->next;

}

this->begin.elem = this->head;

this->end.elem = this->tail;

return \*this;

}

int& operator [] (int index) {

cout << "Вызвана операция поиска по индексу: " << endl;

if (index < this->size && index >= 0) {

Node\* current = head;

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

current = current->next;

}

return current->data;

}

else {

cout << "Индекс не найден " << endl;

}

}

int operator () () {

return this->size;

}

List operator \* (List& list) {

int temp;

if (this->size > list.size) {

temp = list.size;

}

else {

temp = this->size;

}

List templ(temp, 0);

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

templ[i] = (\*this)[i] \* list[i];

}

return templ;

}

Iterator first() {

return this->begin;

}

Iterator last() {

return this->end;

}

friend istream& operator >> (istream& is, const List& list) {

cout << "Вызван оператор перегрузки »" << endl << endl;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

is >> current\_node->data;

current\_node = current\_node->next;

}

cout << "Ввод закончен" << endl << endl;

return is;

}

friend ostream& operator << (ostream& os, List& list) {

cout << "Вызван оператор перегрузки « " << endl << endl;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

os << current\_node->data << " ";

current\_node = current\_node->next;

}

cout << endl << endl << "Вывод закончен" << endl << endl;

return os;

}

};

int main() {

system("chcp 1251>NULL");

srand(time(NULL));

List l1(10, 5);

cout << l1;

List l2(10);

cin >> l2;

cout << l2;

List l3 = l2;

cout << l3;

cout << "Добавление элемента в конец: " << endl << endl;

l3.pushBack(14);

cout << l3;

cout << "Добавление элемента в начало: " << endl << endl;

l3.pushFront(14);

cout << l3;

cout << "Удаление элемента из начала: " << endl << endl;

l3.popFront();

cout << l3;

cout << "Удаление элемента из конца: " << endl << endl;

l3.popBack();

cout << l3;

cout << "Вызвана операция умножения: " << endl << endl;

List l4(6);

cin >> l4;

cout << l4;

List l5 = l4 \* l3;

cout << l5;

Iterator i = l5.first();

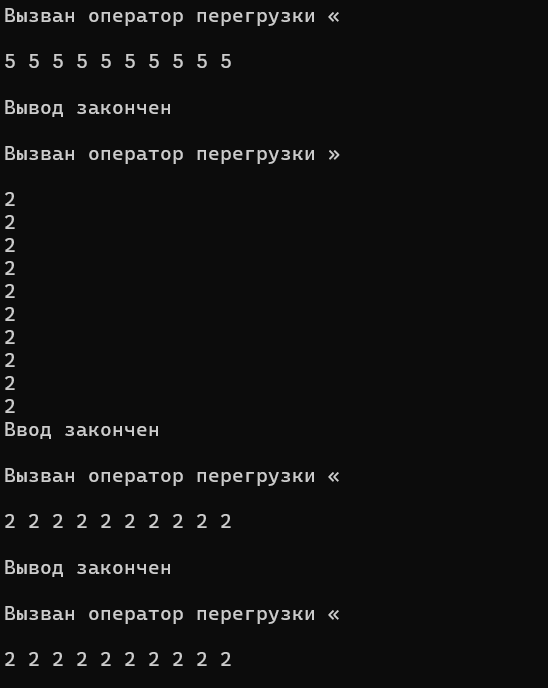
i = i + 2;

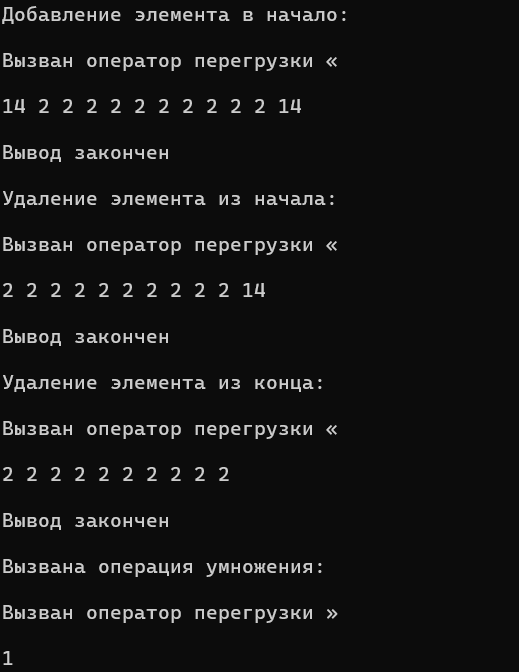
cout << "Элемент стоящий на втором месте: "<< \* i;

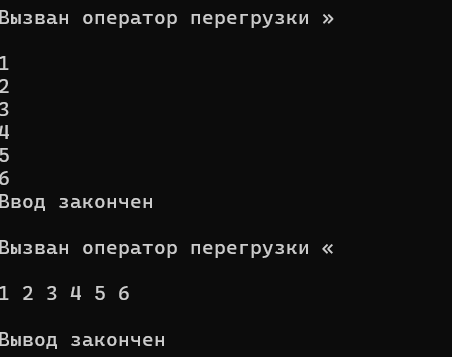
return 0;

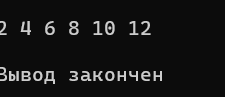
}

**Работа программы:**

****







**Ответ на вопросы:**

1. \*\*Абстрактный тип данных (АТД)\*\* — это модель данных с определенными операциями, но без конкретной реализации. Он определяет набор операций, которые могут быть выполнены с данными, но скрывает способы их реализации. Примеры АТД включают в себя стек, очередь, список, дерево и т.д.

2. \*\*Примеры абстракции через каршиеризацию\*\*:

- \*\*АТД Стек\*\*: позволяет добавлять элементы только на вершину и удалять с нее.

- \*\*АТД Очередь\*\*: работает по принципу FIFO (первым вошел, первым вышел).

- \*\*АТД Список\*\*: хранит элементы в определенном порядке, позволяя вставлять и удалять их в течение списка.

3. \*\*Контейнер\*\* — это структура данных, которая хранит элементы одного или разных типов данных. Примерами контейнеров могут служить вектор, список, множество, отображение и т.д.

4. \*\*Группы операций в контейнерах:\*\*

- \*\*Операции доступа\*\*: поиск, чтение, запись элементов.

- \*\*Операции вставки и удаления элементов\*\*: добавление и удаление элементов из контейнера.

- \*\*Операции манипулирования\*\*: изменения порядка, сортировка, фильтрация и т.д.

5. \*\*Виды доступа к элементам контейнеров\*\*:

- \*\*Последовательный доступ\*\*: итераторы, циклы.

- \*\*Доступ по индексу\*\*: векторы, массивы.

- \*\*Доступ через ключ\*\*: ассоциативные контейнеры (например, отображения).

6. \*\*Итератор\*\* — это объект, который позволяет получать доступ к элементам контейнера и перемещаться по ним. Он предоставляет интерфейс для итерации по элементам контейнера без раскрытия его внутренней структуры.

7. \*\*Принципы реализации итератора\*\*:

- \*\*Встроенный итератор\*\*: встроен в сам контейнер.

- \*\*Внешний итератор\*\*: создается отдельно от контейнера, но имеет доступ к его элементам.

8. \*\*Объединение контейнеров\*\* можно организовать путем создания нового контейнера и добавления в него элементов из двух или более исходных контейнеров.

9. Как вопрос сформулирован некорректно, я не могу предоставить точный ответ. Если предположить, что элементы предоставляют сумму, то добавление элементов в таком контейнере приведет к сложению их значений.

10. \*\*Контейнеры\*\*:

- \*\*a:\*\* Не является контейнером.

- \*\*b:\*\* Является контейнером.

- \*\*c:\*\* Не является контейнером.

- \*\*d:\*\* Не является контейнером.

11. \*\*"a"\*\* не является контейнером, так как это просто переменная типа `int`.

12. \*\*"a"\*\* будет не контейнером, так как содержит элементы только одного типа.

13. \*\*В случае контейнера как динамического массива\*\* доступ к элементам осуществляется по индексу.

14. \*\*Для контейнера как линейного списка\*\* доступ к элементам будет осуществляться последовательно, путем перехода от одного элемента к другому при помощи указателей или итераторов.

15. \*\*Да. Контейнер может\*\*.

**Вывод:** Задача была выполнена. Всё получилось.

