Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

**ОТЧЁТ**

Тема: «Последовательные контейнеры библиотеки STL»

Выполнил

Студент РИС-22-2б

Зубов Р.А.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

**Постановка задачи**

Задача 1.

* + 1. Создать последовательный контейнер.
    2. Заполнить его элементами стандартного типа (тип указан в варианте).
    3. Добавить элементы в соответствии с заданием
    4. Удалить элементы в соответствии с заданием.
    5. Выполнить задание варианта для полученного контейнера.
    6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 2.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 3

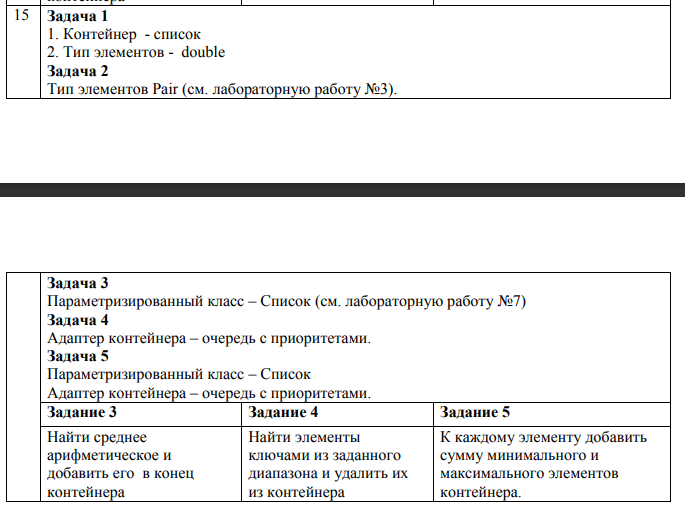
1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

Задача 4

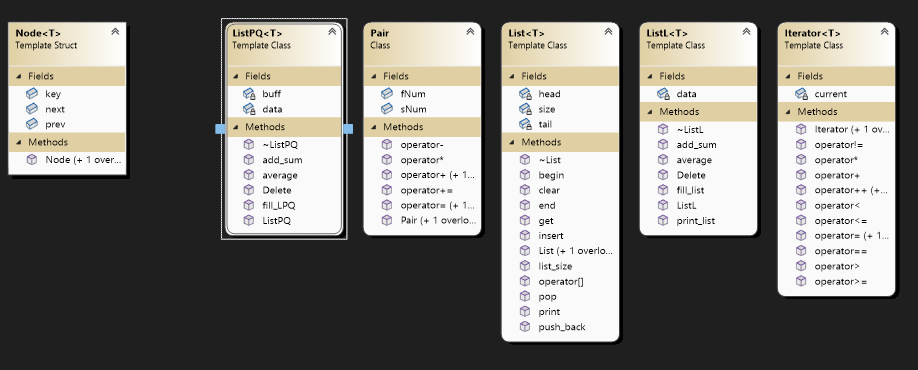
1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

****

**UML**

****

**Код программы**

**Pair.h:** #pragma once

#include <iostream>

class Pair

{

public:

int fNum = 0;

double sNum = 0;

Pair() {}

Pair(int first, double second) {

fNum = first;

sNum = second;

}

Pair& operator=(const Pair& t);

Pair& operator=(const int& t);

Pair operator\*(Pair& t);

friend std::ostream& operator<<(std::ostream& out, const Pair& t);

friend std::istream& operator>>(std::istream& in, const Pair& t);

};

**Pair.cpp:**

#include "Pair.h"

Pair& Pair::operator=(const Pair& pair) {

this->fNum = pair.fNum;

this->sNum = pair.sNum;

return \*this;

}

Pair& Pair::operator=(const int& t) {

this->fNum = t;

this->sNum = t;

return \*this;

}

Pair Pair::operator\*(Pair& pair) {

Pair new\_pair;

new\_pair.fNum = this->fNum \* pair.fNum;

new\_pair.sNum = this->sNum \* pair.sNum;

return new\_pair;

}

std::ostream& operator<<(std::ostream& out, const Pair& pair) {

out << pair.fNum << " : " << pair.sNum;

return out;

}

std::istream& operator>>(std::istream& in, Pair& pair) {

in >> pair.fNum >> pair.sNum;

return in;

}

**List.h:** #pragma once

#include <cstddef>

#include <iostream>

template <typename T>

struct Node {

T key;

Node<T>\* next = NULL;

Node<T>\* prev = NULL;

Node();

Node(T k, Node<T>\* n, Node<T>\* p);

};

template<typename T>

class Iterator {

template<typename T>

friend class List;

private:

Node<T>\* current = NULL;

public:

Iterator() { current = NULL; }

Iterator(Node<T>\* node) : current(node) {}

bool operator==(const Iterator<T>& other) const { //ñðàâíèâàþò òåêóùèé óçåë

return current == other.current; // è óçåë äðóãîãî èòåðàòîðà

} //

bool operator!=(const Iterator<T>& other) const { // íà ðàâåíñòâî è íåðàâåíñòâî.

return current != other.current;

}

T& operator\*() const { // îïåðàöèÿ ðàçûìåíîâàíèÿ èòåðàòîðà

return current->key;

}

Iterator& operator++() { //++i

current = current->next;

return \*this;

}

Iterator operator++(int) { // i++

Iterator<T> old = \*this;

current = current->next;

return old;

}

Iterator operator+(int n) const { // ïåðåõîä âïðàâî ê n ýëåìåíòó

Iterator<T> it(\*this);

while (n > 0 && it.current != NULL) {

it.current = it.current->next;

n--;

}

return it;

}

};

template<typename T>

class List {

Node<T>\* head;

Node<T>\* tail;

int size;

public:

List() {

head = NULL;

tail = NULL;

size = 0;

}

~List() {

clear();

}

T& get(int index);

T& operator[](int index) {

return get(index); // âîçâðàùàåì ññûëêó íà ýëåìåíò ïî èíäåêñó

}

// Ìåòîä äîáàâëåíèÿ ýëåìåíòà â êîíåö ñïèñêà

void push\_back(T k);

void pop(T key);

int list\_size() {

return size;

}

void clear();

void print() const;

friend std::ostream& operator<<(std::ostream& out, const List<T>& list)

{

list.print();

return out;

}

friend std::istream& operator>>(std::istream& in, List<T>& list)

{

int k;

in >> k;

list.push\_back(k);

return in;

}

Iterator<T> beg() {

return Iterator<T>(head); // âîçâðàùàåì èòåðàòîð íà ïåðâûé ýëåìåíò

}

Iterator<T> end() {

return Iterator<T>(tail->next); // âîçâðàùàåì èòåðàòîð íà ïîñëåäíèé ýëåìåíò

}

friend List<T> operator\*(List<T>& left, List<T>& right) {

List<T> result; // ñîçäàåì íîâûé ñïèñîê

Iterator<T> it1 = left.beg(); // èòåðàòîð ïî ïåðâîìó ñïèñêó

Iterator<T> it2 = right.beg(); // èòåðàòîð ïî âòîðîìó ñïèñêó

while (it1 != left.end() && it2 != right.end()) {

T res = (\*it1) \* (\*it2);

result.push\_back(res); // äîáàâëÿåì ïðîèçâåäåíèå ýëåìåíòîâ â íîâûé ñïèñîê

++it1; // ïåðåõîäèì ê ñëåäóþùåìó ýëåìåíòó ïåðâîãî ñïèñêà

++it2; // ïåðåõîäèì ê ñëåäóþùåìó ýëåìåíòó âòîðîãî ñïèñêà

}

return result; // âîçâðàùàåì íîâûé ñïèñîê

}

};

template<typename T>

Node<T>::Node(T k, Node<T>\* n, Node<T>\* p)

{

key = k;

next = n;

prev = p;

}

template<typename T>

T& List<T>::get(int index)

{

Node<T>\* current = head; // íà÷èíàåì ñ ãîëîâû ñïèñêà

int i = 0;

while (i < index && current->next != NULL) {

current = current->next; // ïåðåõîäèì ê ñëåäóþùåìó óçëó

}

return current->key; // âîçâðàùàåì ññûëêó íà äàííûå óçëà

}

template<typename T>

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

{

Node<T>\* newNode = new Node<T>(k, NULL, tail); // ñîçäàåì íîâûé óçåë ñ êëþ÷îì key, óêàçàòåëåì íà ñëåäóþùèé óçåë ðàâíûì NULL è óêàçàòåëåì íà ïðåäûäóùèé óçåë ðàâíûì tail

if (tail != NULL) {

tail->next = newNode; // åñëè tail íå ðàâåí NULL, òî óñòàíàâëèâàåì óêàçàòåëü íà ñëåäóþùèé óçåë ó tail íà íîâûé óçåë

}

tail = newNode; // óñòàíàâëèâàåì tail íà íîâûé óçåë

if (head == NULL) {

head = newNode; // åñëè ñïèñîê áûë ïóñòîé, òî óñòàíàâëèâàåì head íà íîâûé óçåë

}

size++;

}

template<typename T>

void List<T>::pop(T key) {

Node<T>\* p = head;

while (p != NULL && p->key != key) { // Ïðîõîäèì ïî ñïèñêó äî íóæíîãî êëþ÷à

p = p->next;

}

if (p != NULL) { // Åñëè íàøëè óçåë ïî ïîçèöèè

if (p->prev != NULL) { // Åñëè óçåë íå ÿâëÿåòñÿ ãîëîâíûì

p->prev->next = p->next;

}

else { // Åñëè óçåë ÿâëÿåòñÿ ãîëîâíûì

head = p->next;

}

if (p->next != NULL) { // Åñëè óçåë íå ÿâëÿåòñÿ õâîñòîâûì

p->next->prev = p->prev;

}

else { // Åñëè óçåë ÿâëÿåòñÿ õâîñòîâûì

tail = p->prev;

}

delete p; // Óäàëÿåì òåêóùèé óçåë èç ïàìÿòè

size--;

}

}

template<typename T>

void List<T>::clear()

{

Node<T>\* p = head;

while (p != NULL) {

Node<T>\* q = p;

p = p->next;

delete q; // Óäàëÿåì òåêóùèé óçåë èç ïàìÿòè

}

head = NULL;

tail = NULL;

size = 0;

}

template<typename T>

void List<T>::print() const

{

Node<T>\* node = head;

while (node != NULL) {

std::cout << node->key << "->";

node = node->next;

}

std::cout << std::endl;

}

**ListPQ.h:** #pragma once

#include <iostream>

#include <queue>

using namespace std;

template<typename T>

class ListPQ {

private:

priority\_queue<T> data;

priority\_queue<T> buff;

public:

ListPQ() {}

~ListPQ() {}

void fill\_LPQ() {

int n;

cout << "Enter size: ";

cin >> n;

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

T element;

cout << "Enter elem: ";

cin >> element;

data.push(element);

}

}

void average()

{

T sum = 0;

while (!data.empty())

{

cout << data.top() << " | ";

sum = sum + data.top();

buff.push(data.top());

data.pop();

}

sum /= buff.size();

buff.push(sum);

cout << endl;

cout << "Find avg and push back to container: \n";

while (!buff.empty())

{

cout << buff.top() << " | ";

data.push(buff.top());

buff.pop();

}

cout << endl;

}

void Delete(int s\_range, int e\_range)

{

int ch = 1;

while (!data.empty())

{

if ((ch < s\_range || ch > e\_range) )

buff.push(data.top());

data.pop();

ch++;

}

while (!buff.empty())

{

cout << buff.top() << " | ";

data.push(buff.top());

buff.pop();

}

cout << endl;

}

void add\_sum()

{

cout << "Adding sum to elems: \n";

T max = data.top();

T min = data.top();

while (!data.empty())

{

if (data.top() > max) max = data.top();

if (data.top() < min) min = data.top();

buff.push(data.top());

data.pop();

}

while (!buff.empty())

{

data.push(max + min + buff.top());

buff.pop();

}

while (!data.empty())

{

buff.push(data.top());

cout << data.top() << " | ";

data.pop();

}

cout << endl;

}

};

**ListL.h:** #pragma once

#include<iostream>

#include <list>

#include<algorithm>

using namespace std;

template <typename T>

class ListL {

private:

list<T> data;

public:

ListL() {}

~ListL(){}

void average() {

T sum = 0;

for (const auto& element : data) {

sum += element;

}

T avg = sum / data.size();

data.push\_back(avg);

}

void Delete(int s\_range, int e\_range)

{

auto s\_it = data.begin();

advance(s\_it, s\_range);

auto e\_it = data.begin();

advance(e\_it, e\_range);

data.erase(s\_it, e\_it);

}

void add\_sum() {

T min\_value = \*min\_element(data.begin(), data.end());

T max\_value = \*max\_element(data.begin(), data.end());

for (auto& element : data) {

element += (min\_value + max\_value);

}

}

void print\_list() {

cout << "List elements: ";

for (const auto& element : data) {

cout << element << " | ";

}

cout << std::endl;

}

void fill\_list(int num\_elements, bool is\_integer = true) {

data.clear();

for (int i = 0; i < num\_elements; ++i) {

double random\_num = is\_integer ? rand() % 101 : rand() / (RAND\_MAX + 1.0) \* 100;

data.push\_back(random\_num);

}

}

};

**Source.cpp:**

#include<iostream>

#include <list>

#include <queue>

#include <algorithm>

#include "List.h"

#include "ListL.h"

#include "ListPQ.h"

#include "Pair.h"

using namespace std;

template<typename T>

void showL(list<T>& L)

{

for (T num : L)

cout << num << " - ";

cout << "\n";

}

// Задание 1

void task1()

{

list<double> container;

for (int i = 0; i < 5; ++i) {

double a = (double)rand() / 100.00;

container.push\_back(a);

}

showL(container);

//Find avg

double sum = 0;

for (double num : container)

sum += num;

container.push\_back(sum / container.size());

cout << "Find avg and push back to container: \n";

showL(container);

//find elems

int d1, d2;

cout << "Enter range from: "; cin >> d1;

cout << " to: "; cin >> d2;

double key;

cout << "Enter key to delete: "; cin >> key;

auto s\_it = container.begin();

advance(s\_it, d1);

auto e\_it = container.begin();

advance(e\_it, d2);

for (auto it = s\_it; it != e\_it; ++it)

{

if (\*it == key) {

container.erase(it);

break;

}

}

cout << "After deleting: \n";

showL(container);

//sum max min

cout << "Adding sum to elems: \n";

double MMsum = \*min\_element(container.begin(), container.end())

+ \*max\_element(container.begin(), container.end());

for (double& num : container)

num += MMsum;

showL(container);

}

// Задание 2

void task2()

{

list<Pair> PP;

PP.push\_back(Pair(1, 1.1));

PP.push\_back(Pair(2, 2.2));

PP.push\_back(Pair(3, 3.3));

PP.push\_back(Pair(4, 4.4));

showL(PP);

//Find avg

Pair sum (0,0.0);

for (Pair num : PP)

sum = num + sum;

sum.fNum /= PP.size();

sum.sNum /= PP.size();

PP.push\_back(sum);

cout << "Find avg and push back to container: \n";

showL(PP);

//find elems

int d1, d2;

cout << "Enter range from: "; cin >> d1;

cout << " to: "; cin >> d2;

Pair key;

cout << "Enter key to delete: "; cin >> key;

auto s\_it = PP.begin();

advance(s\_it, d1);

auto e\_it = PP.begin();

advance(e\_it, d2);

for (auto it = s\_it; it != e\_it; ++it)

{

if (\*it == key) {

PP.erase(it);

break;

}

}

cout << "After deleting: \n";

showL(PP);

//sum max min

cout << "Adding sum to elems: \n";

Pair MMSUM = \*min\_element(PP.begin(), PP.end())

+ \*max\_element(PP.begin(), PP.end());

for (Pair& num : PP)

num = num + MMSUM;

showL(PP);

}

// Задание 3

void task3()

{

ListL<double> L;

L.fill\_list(6.0);

L.print\_list();

cout << "Find avg and push back to container: \n";

L.average();

L.print\_list();

int d1, d2;

cout << "Enter range from: "; cin >> d1;

cout << " to: "; cin >> d2;

L.Delete(d1,d2);

L.print\_list();

cout << "After deleting: \n";

L.print\_list();

cout << "Adding sum to elems: \n";

L.add\_sum();

L.print\_list();

}

// Задание 4

void task4()

{

priority\_queue<Pair> QP;

priority\_queue<Pair> QPbuf;

QP.push(Pair(1, 1.1));

QP.push(Pair(2, 2.2));

QP.push(Pair(3, 3.3));

QP.push(Pair(4, 4.4));

//Find avg

Pair sum(0, 0.0);

while (!QP.empty())

{

cout << QP.top() << " | ";

sum = sum + QP.top();

QPbuf.push(QP.top());

QP.pop();

}

sum.fNum /= QPbuf.size();

sum.sNum /= QPbuf.size();

QPbuf.push(sum);

cout << endl;

cout << "Find avg and push back to container: \n";

while (!QPbuf.empty())

{

cout << QPbuf.top() << " | ";

QP.push(QPbuf.top());

QPbuf.pop();

}

cout << endl;

//find elems

int ch = 1, d1, d2;

cout << "Enter range from: "; cin >> d1;

cout << " to: "; cin >> d2;

Pair key;

cout << "Enter key to delete: "; cin >> key;

while (!QP.empty())

{

if ((ch < d1 || ch > d2) || key != QP.top())

QPbuf.push(QP.top());

QP.pop();

ch++;

}

while (!QPbuf.empty())

{

cout << QPbuf.top() << " | ";

QP.push(QPbuf.top());

QPbuf.pop();

}

cout << endl;

//sum max min

cout << "Adding sum to elems: \n";

Pair max = QP.top();

Pair min = QP.top();

while(!QP.empty())

{

if (QP.top() > max) max = QP.top();

if (QP.top() < min) min = QP.top();

QPbuf.push(QP.top());

QP.pop();

}

while (!QPbuf.empty())

{

QP.push(max + min + QPbuf.top());

QPbuf.pop();

}

while (!QP.empty())

{

QPbuf.push(QP.top());

cout << QP.top() << " | ";

QP.pop();

}

cout << endl;

}

// Задание 5

void task5()

{

ListPQ <double> LP;

LP.fill\_LPQ();

LP.average();

int d1, d2;

cout << "Enter range to delete: "; cin >> d1; cin >> d2;

LP.Delete(d1,d2);

LP.add\_sum();

}

int main()

{

srand(time(NULL));

cout << "Results: \n";

cout << "=================\n\n";

cout << "-------TASK1-------\n";

task1();

cout << "-------TASK2-------\n";

task2();

cout << "-------TASK3-------\n";

task3();

cout << "-------TASK4-------\n";

task4();

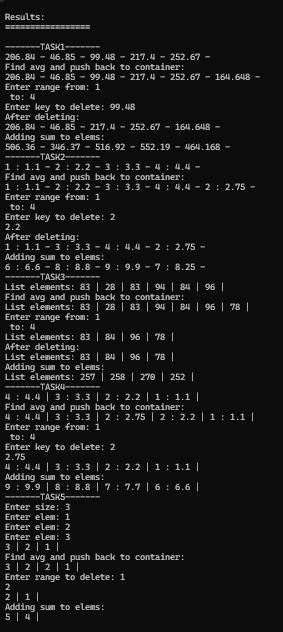
cout << "-------TASK5-------\n";

task5();

return 0;

}

**Результаты программ**



**Контрольные вопросы**

1. Из каких частей состоит библиотека STL?

STL – *Standard Template Library*, стандартная библиотека шаблонов состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов.

1. Какие типы контейнеров существуют в STL?

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные

1. Что нужно сделать для использования контейнера STL в своей программе?

Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

#include <vector>

#include <list>

#include “person.h”

…..

vector<int> v;

list<person> l;

1. Что представляет собой итератор?

Функцию можно использовать для поиска элементов в массиве любого типа, но использовать ее для списка нельзя, поэтому авторы STL ввели понятие итератора. Итератор более общее понятие, чем указатель. Тип iterator определен для всех контейнерных классов STL

1. Какие операции можно выполнять над итераторами?

К основным операциям, выполняемым с любыми итераторами, относятся:

• Разыменование итератора: если р — итератор, то \*р — значение объекта, на который он ссылается.

• Присваивание одного итератора другому.

• Сравнение итераторов на равенство и неравенство (== и !=).

• Перемещение его по всем элементам контейнера с помощью префиксного (++р) или постфиксного (р++) инкремента.

1. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

Организация циклов просмотра элементов контейнеров тоже имеет некоторую специфику. Так, если i — некоторый итератор, то вместо привычной формы for (i =0; i < n; ++i) используется следующая:

for (i = first; i != last; ++i), где first - значение итератора, указывающее на первый элемент в контейнере, a last — значение итератора, указывающее на воображаемый элемент, который следует за последним элементом контейнера.

Операция сравнения < заменена на операцию ! =, т. к. операции < и > для итераторов в общем случае не поддерживаются.

1. Какие типы итераторов существуют?

В STL существуют следующие типы итераторов:

• входные,

• выходные,

• прямые,

• двунаправленные итераторы,

• итераторы произвольного доступа.

1. Перечислить операции и методы общие для всех контейнеров.

|  |  |
| --- | --- |
| **Операция или метод** | **Пояснение** |
| Операции равенства (==) и  неравенства (!=) | Возвращают значение true или false |
| Операция присваивания (=) | Копирует один контейнер в другой |
| clear | Удаляет все элементы |
| insert | Добавляет один элемент или диапазон элементов |
| erase | Удаляет один элемент или диапазон элементов |
| size\_type size() const | Возвращает число элементов |
| size\_type max\_size() const | Возвращает максимально допустимый размер  контейнера |
| bool empty0 const | Возвращает true, если контейнер пуст |
| iterator begin() | Возвращают итератор на начало контейнера (итерации будут производиться в прямом  направлении) |
| iterator end() | Возвращают итератор на конец контейнера  (итерации в прямом направлении будут закончены) |
| reverse\_iterator begin() | Возвращают реверсивный итератор на конец  контейнера (итерации будут производиться в обратном направлении) |
| reverse\_iterator end() | Возвращают реверсивный итератор на начало  контейнера (итерации в обратном направлении будут закончены |

1. Какие операции являются эффективными для контейнера vector? Почему?

Контейнер вектор является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации [] или метода at. Однако вставка элемента в любую позицию, кроме конца вектора, неэффективна. Для этого потребуется сдвинуть все последующие элементы путем копирования их значений. По этой же причине неэффективным является удаление любого элемента, кроме последнего.

1. Какие операции являются эффективными для контейнера list? Почему?

Контейнер список организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке. Однако список не поддерживает произвольного доступа к своим элементам: например, для выборки n-го элемента нужно последовательно выбрать предыдущие п-1 элементов.

1. Какие операции являются эффективными для контейнера deque? Почему?

Контейнер двусторонняя очередь во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего).

1. Перечислить методы, которые поддерживает последовательный контейнер vector.

|  |  |
| --- | --- |
| Vector | |
| push\_back() | добавление  в конец |
| pop\_back() | удаление из конца |
| insert | Вставка в  произвольное место |
| erase | удаление из произвольного места |
| []  at | доступ к произвольному элементу |
| clear() | очистить  вектор |
| swap | Обмен векторов |

1. Перечислить методы, которые поддерживает последовательный контейнер list.

|  |  |
| --- | --- |
| List | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольное место |
| erase | удаление из произвольного места |
| clear() | Очистить вектор |
| swap | Обмен списков |
| splice | Сцепка списков |

1. Перечислить методы, которые поддерживает последовательный контейнер deque

|  |  |
| --- | --- |
| Deque | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольн ое место |
| erase | удаление из произвольн  ог оместа |
| []  at | доступ к произвольн ому  элементу |

1. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

vector<int> l;

vector<int>::iterator it\_start = l.begin();

vector<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер vector. Как удалить из него последний элемент?

vector<double> l;

l.pop\_back();

1. Задан контейнер list. Как удалить из него элементы со 2 по 5?

list<string> l;

list<string>::iterator it\_start = l.begin();

list<string>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер list. Как удалить из него последний элемент?

list<double> l;

l.pop\_back();

1. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

deque<int> l;

deque<int>::iterator it\_start = l.begin();

deque<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер deque. Как удалить из него последний элемент?

deque<double> l;

l.pop\_back();

1. Написать функцию для печати последовательного контейнера с использованием итератора.

void print\_list(LinkedList l)

{

cout << "\nСейчас список такой: \n";

copy(l.begin(), l.end(), ostream\_iterator<double>(cout, " "));

}

1. Что представляют собой адаптеры контейнеров?

Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе рассмотренных выше классов, поэтому они называются адаптерами контейнеров.

1. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

Объявление stack<int> s создает стек на базе двусторонней очереди (по умолчанию). Если по каким-то причинам нас это не устраивает, и мы хотим создать стек на базе списка, то объявление будет выглядеть следующим образом:

stack<int, list<int> > s;

Смысл такой реализации заключается в том, что специализированный класс просто переопределяет интерфейс класса-прототипа, ограничивая его только теми методами, которые нужны новому классу. Cтек не позволяет выполнить произвольный доступ к своим элементам, а также не дает возможности пошагового перемещения, т. е. итераторы в стеке не поддерживаются

1. Перечислить методы, которые поддерживает контейнер stack.

Методы класса stack:

• push () - добавление в конец;

• pop () - удаление из конца;

• top () - получение текущего элемента стека;

• empty() - проверка пустой стек или нет;

• size () – получение размера стека.

1. Перечислить методы, которые поддерживает контейнер queue.

Методы класса queue:

• push () – добавление в конец очереди;

• pop () – удаление из начала очереди;

• front () – получение первого элемента очереди;

• back()- получение последнего элемента очереди;

• empty () – проверка пустая очередь или нет;

• size() – получение размера очереди.

1. Чем отличаются друг от друга контейнеры queue и priority\_queue?

Шаблонный класс priority\_queue (заголовочный файл <queue>) поддерживает такие же операции, как и класс queue, но реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере. Поэтому после каждого изменения состояния очереди максимальный элемент из оставшихся сдвигается в начало контейнера.

1. Задан контейнер stack. Как удалить из него элемент с заданным номером?

template <class T>

void del(stack<T>& st, int index)

{

stack<T> temp;

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

{

temp.push(st.top());

st.pop();

}ss

st.pop();

while (temp.size())

{

st.push(temp.top());

temp.pop();

}

}

Но так делать не надо, потому что stack не предназначен для того, чтобы из него удалялись элементы не по порядку. В данном случае будет гораздо логичнее применить структуру данных список или любой контейнер STL.

1. Задан контейнер queue. Как удалить из него элемент с заданным номером?

std::deque<int> dq{1, 2, 3, 4, 5};

int index = 2;

auto it = std::next(dq.begin(), index);

dq.erase(it);

1. Написать функцию для печати контейнера stack с использованием итератора.

Это невозможно, так как у контейнера stack даже итераторов нет.

Вот пример как можно посмотреть все, что лежит в stack, но так делать тоже не надо, потому что это ломает весь смысл работы с данным контейнером, гораздо логичнее будет взять любой другой контейнер, например отлично подойдет однонаправленный список.

template <class T>

void printStack(stack<T> s)

{

while (!s.empty()) {

cout << s.top() << " ";

s.pop();

}}

1. Написать функцию для печати контейнера queue с использованием итератора.

template <class T>

void printDeque(deque<T> dq)

{

for (auto i = dq.begin(); i != dq.end(); ++i)

cout << \*i << endl;

}