Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №5**

По дисциплине: “ООТПиСП”

Тема: “Последовательные контейнеры библиотеки STL

Вариант №6

**Выполнил**:

студент 2 курса

группы ПО-7

Комиссаров А.Е.

**Проверилa:**

Хацкевич М.В.

Брест, 2022

**Цель:** научиться использовать последовательные контейнеры библиотеки STL в ОО программе.

**Вариант № 6.**

Задача 1:1. Контейнер – список. 2. Тип элементов – int

Задача 2: Тип элементов Pair.

Задача 3: Параметризированный класс – Список. Найти среднее арифметическое и добавить его на заданную позицию контейнера.

Задача 4: Адаптер контейнера – очередь с приоритетами. Найти элементы ключами из заданного диапазона и удалить их из контейнера.

Задача 5: Параметризированный класс – Список. Адаптер контейнера – очередь с приоритетами. Из каждого элемента вычесть среднее арифметическое контейнера.

**Ход работы:**

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.

**Код программы:**

#include <iostream>

#include <list>

#include <algorithm>

#include <List>

using namespace std;

typedef list<int> Ls;

Ls makeList(int n) {

Ls list;

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

int a = rand() % 100 - 50;

list.push\_back(a);

}

return list;

}

void printList(Ls l) {

for (auto it = l.begin(); it != l.end(); ++it) {

cout << \*it << " ";

}

cout << endl;

}

int average(Ls l) {

int sum = 0;

for (auto it = l.begin(); it != l.end(); ++it) {

sum += \*it;

}

int size = l.size();

return sum / size;

}

void addToPosition(Ls& l, int el, int pos) {

int i = 0;

for (auto it = l.begin(); it != l.end(); ++it) {

if (i == pos) {

l.insert(it, el);

}

i++;

}

}

void deleteByKeyInRange(Ls& l, int start, int end) {

l.remove\_if([&start, &end](int n) { bool res = (n >= start && n <= end); return res; });

}

void subAverage(Ls& l, int aver) {

for (auto it = l.begin(); it != l.end(); it++) {

\*it += aver;

}

}

int main() {

srand(time(0));

try {

list<int> lst;

// Making List

int n;

cout << "Enter n: ";

cin >> n;

lst = makeList(n);

printList(lst);

// Calculating average

int el = average(lst);

// Position for insert

int pos;

cout << "Position: ";

cin >> pos;

if (pos > lst.size()) throw 1;

// Add to position

addToPosition(lst, el, pos);

printList(lst);

// Deleting by key in range

int start, end;

cout << "Start: "; cin >> start;

cout << "End: "; cin >> end;

deleteByKeyInRange(lst, start, end);

printList(lst);

// Add average to each List element

cout << "\nBefore adding average\n";

printList(lst);

int aver = average(lst);

cout << "Average value: " << aver << endl;

subAverage(lst, aver);

cout << "\nAfter adding average\n";

printList(lst);

}

catch (int) {

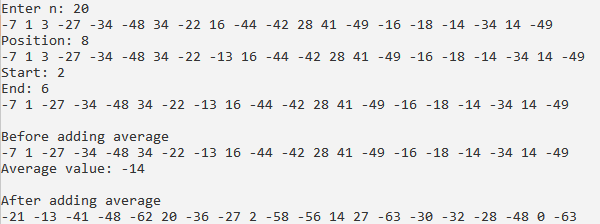
cout << "Error" << endl;

}

return 0;

}

**Результат работы программы:**



**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.

**Код программы:**

#include <iostream>

#include <list>

#include <random>

using namespace std;

class Pair {

public:

int val1; double val2;

Pair(); Pair(int, double);

Pair(const Pair&); virtual ~Pair() {};

Pair& operator=(const Pair&);

Pair operator+(const Pair&);

bool operator>(const Pair&);

bool operator<(const Pair&);

Pair operator/(const Pair&);

Pair operator/(const int&);

};

Pair::Pair() { val1 = 0; val2 = 0.0;}

Pair::Pair(int v1, double v2) { val1 = v1; val2 = v2;}

Pair::Pair(const Pair& t) { val1 = t.val1; val2 = t.val2;}

Pair& Pair::operator=(const Pair& t) { val1 = t.val1; val2 = t.val2; return \*this;}

bool Pair::operator<(const Pair& t) {

if (val1 < t.val1) return true; if (val2 < t.val2) return true;

if (val1 < t.val1 && val2 < t.val2) return true;

return false;}

bool Pair::operator>(const Pair& t) {

if (val1 > t.val1) return true; if (val2 > t.val2) return true;

if (val1 > t.val1 && val2 > t.val2) return true;

return false;}

Pair Pair::operator+(const Pair& t) {

int temp1 = val1 + t.val1; double temp2 = val2 + t.val2;

Pair p; p.val1 = temp1; p.val2 = temp2;

return p;}

Pair Pair::operator/(const Pair& t) {

int temp1 = val1 / t.val1;

double temp2 = val2 / t.val2;

Pair p; p.val1 = temp1; p.val2 = temp2;

return p;}

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

Pair p; p.val1 = val1 / t; p.val2 = val2 / t;

return p;}

ostream& operator<< (ostream& out, const Pair& t) {

out << t.val1 << " : " << t.val2 << endl;

return out;}

istream& operator>> (istream& in, Pair& t) {

cout << "Value 1: "; in >> t.val1;

cout << "Value 2: "; in >> t.val2;

return in;}

double randomDouble(int min = -50, int max = 100) {

random\_device rd; default\_random\_engine eng(rd());

uniform\_real\_distribution<double> distr(min, max);

return distr(eng);}

typedef list<Pair> Ls;

Ls makeVector(int n) {

Ls l;

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

int val1 = rand() % 100 - 50;

double val2 = randomDouble();

Pair p(val1, val2);

l.push\_back(p); } return l;}

void printVector(Ls l) {

for (auto it = l.begin(); it != l.end(); ++it) {

cout << \*it << " "; }

cout << "\n";}

Pair average(Ls l) {

int sumInt = 0;

double sumDouble = 0.0;

for (auto it = l.begin(); it != l.end(); ++it) {

sumInt += it->val1;

sumDouble += it->val2; }

int size = l.size();

Pair p(sumInt / size, sumDouble / size);

return p;}

void addToPosition(Ls& l, const Pair& el, int pos) {

int i = 0;

for (auto it = l.begin(); it != l.end(); ++it) {

if (i == pos) {

l.insert(it, el);

}

i++; }}

void deleteByKeyInRange(Ls& l, int start, int end) {

l.remove\_if([&start, &end](Pair n) {

bool res1 = (n.val1 >= start && n.val1 <= end);

bool res2 = (n.val2 >= start && n.val2 <= end);

return (res1 || res2);

});}

void subAverage(Ls& l, Pair aver) {

for (auto it = l.begin(); it != l.end(); it++) {

it->val1 += aver.val1;

it->val2 += aver.val2;

}}

int main() {

try {

srand(time(0));

list<Pair> lst;

// Making vector

int n;

cout << "Enter n: "; cin >> n;

lst = makeVector(n);

printVector(lst);

// Calculating average

Pair aver = average(lst);

int pos; cout << "Position: "; cin >> pos;

if (pos > lst.size()) throw 1;

addToPosition(lst, aver, pos);

printVector(lst);

// Deleting by key in range

int start, end;

cout << "Start: "; cin >> start;

cout << "End: "; cin >> end;

deleteByKeyInRange(lst, start, end);

printVector(lst);

// Add average to each vector element

cout << "\nBefore adding average\n";

printVector(lst);

aver = average(lst); cout << "Average value: " << aver;

subAverage(lst, aver);cout << "\nAfter adding average\n";

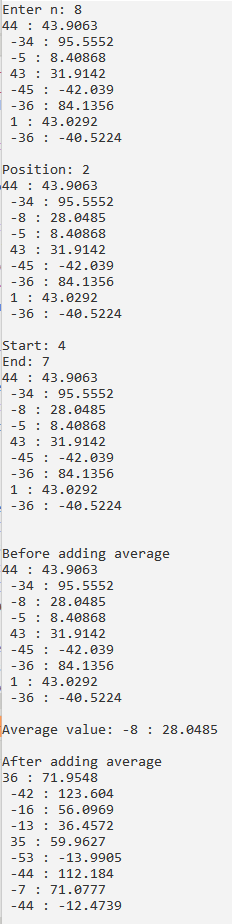
printVector(lst);

}

catch (int) { cout << "Error" << endl; }

return 0;}

**Результат работы программы:**

****

**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.

**Код программы:**

#include <iostream>

#include <list>

using namespace std;

template <class T>class List {

list<T> l; int len;

public:

List();List(int n);

void Print();

~List(); T Average();

void Add(T el, int pos);

void deleteByKeyInRange(int start, int end);

void subAverage(int aver);

};

template <class T>List<T>::List() {len = 0;}

template <class T>List<T>::~List() {}

template <class T>List<T>::List(int n) {

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

cin >> el;

l.push\_back(el);}

len = l.size();}

template <class T>void List<T>::Print() {

for (auto it = l.begin(); it != l.end(); ++it) {

cout << \*it << " ";}

cout << endl;}

template <class T>T List<T>::Average() {

T sum = 0;

for (auto it = l.begin(); it != l.end(); ++it) {

sum += \*it; }

int size = l.size();

return sum / size;}

template <class T>void List<T>::Add(T el, int pos) {

int i = 0;

for (auto it = l.begin(); it != l.end(); ++it) {

if (i == pos) {l.insert(it, el);}

i++; }}

template <class T>void List<T>::deleteByKeyInRange(int start, int end) {

l.remove\_if([&start, &end](int n){ bool res = (n >= start && n <= end); return res; });

}

template <class T>void List<T>::subAverage(int aver) {

for (auto it = l.begin(); it != l.end(); it++) {

\*it += aver; }}

int main() {

List<int> list(5); list.Print();

// Calculating average

int aver = list.Average();

int pos;

cout << "Position: ";

cin >> pos;

list.Add(aver, pos);

list.Print();

// Deleting by key in range

int start, end;

cout << "Start: "; cin >> start;

cout << "End: "; cin >> end;

list.deleteByKeyInRange(start, end);

list.Print();

// Add average to each vector element

cout << "\nBefore adding average\n";

list.Print();

aver = list.Average();

cout << "Average value: " << aver << endl;

list.subAverage(aver);

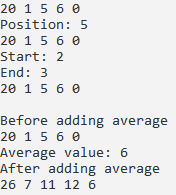
cout << "After adding average\n";

list.Print();

return 0;

}

**Результат работы программы:**

****

**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.

При работе с приоритетной очередью у нас есть доступ только к последнему и первому элементам очереди. Чтобы получить доступ к следующему элементу, элемент нужно удалить (pop). Поэтому перед просмотром содержимого очереди ее надо сохранить во вспомогательном контейнере типа list.

**Код программы:**

#include <queue>

#include <list>

#include <iostream>

#include <iostream>

#include <list>

#include <random>

using namespace std;

class Pair {

public:

int val1; double val2;

Pair(); Pair(int, double);

Pair(const Pair&); virtual ~Pair() {};

Pair& operator=(const Pair&);

Pair operator+(const Pair&);

bool operator>(const Pair&);

bool operator<(const Pair&);

Pair operator/(const Pair&);

Pair operator/(const int&);

};

Pair::Pair() { val1 = 0; val2 = 0.0; }

Pair::Pair(int v1, double v2) { val1 = v1; val2 = v2; }

Pair::Pair(const Pair& t) { val1 = t.val1; val2 = t.val2; }

Pair& Pair::operator=(const Pair& t) { val1 = t.val1; val2 = t.val2; return \*this; }

bool Pair::operator<(const Pair& t) {

if (val1 < t.val1) return true; if (val2 < t.val2) return true;

if (val1 < t.val1 && val2 < t.val2) return true;

return false;

}

bool Pair::operator>(const Pair& t) {

if (val1 > t.val1) return true; if (val2 > t.val2) return true;

if (val1 > t.val1 && val2 > t.val2) return true;

return false;

}

Pair Pair::operator+(const Pair& t) {

int temp1 = val1 + t.val1; double temp2 = val2 + t.val2;

Pair p; p.val1 = temp1; p.val2 = temp2;

return p;

}

Pair Pair::operator/(const Pair& t) {

int temp1 = val1 / t.val1;

double temp2 = val2 / t.val2;

Pair p; p.val1 = temp1; p.val2 = temp2;

return p;

}

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

Pair p; p.val1 = val1 / t; p.val2 = val2 / t;

return p;

}

ostream& operator<< (ostream& out, const Pair& t) {

out << t.val1 << " : " << t.val2 << endl;

return out;

}

istream& operator>> (istream& in, Pair& t) {

cout << "Value 1: "; in >> t.val1;

cout << "Value 2: "; in >> t.val2;

return in;

}

double randomDouble(int min = -50, int max = 100) {

random\_device rd; default\_random\_engine eng(rd());

uniform\_real\_distribution<double> distr(min, max);

return distr(eng);

}

typedef list<Pair> Ls;

Ls makeVector(int n) {

Ls l;

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

int val1 = rand() % 100 - 50;

double val2 = randomDouble();

Pair p(val1, val2);

l.push\_back(p);

} return l;

}

void printVector(Ls l) {

for (auto it = l.begin(); it != l.end(); ++it) {

cout << \*it << " ";

}

cout << "\n";

}

Pair average(Ls l) {

int sumInt = 0;

double sumDouble = 0.0;

for (auto it = l.begin(); it != l.end(); ++it) {

sumInt += it->val1;

sumDouble += it->val2;

}

int size = l.size();

Pair p(sumInt / size, sumDouble / size);

return p;

}

typedef priority\_queue<Pair> Queue;

typedef list<Pair> Ls;

Queue makeQueue(int n) {

Queue q;

Pair val;

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

cin >> val;

q.push(val);

}

return q;

}

Ls copyQueueToVector(Queue q) {

Ls list;

while (!q.empty()) {

list.push\_back(q.top());

q.pop();

}

return list;

}

Queue copyVectorToQueue(Ls list) {

Queue q;

for (auto it = list.begin(); it != list.end(); it++) {

q.push(\*it);

}

return q;

}

void printQueue(Queue q) {

Ls list = copyQueueToVector(q);

for (auto it = list.begin(); it != list.end(); it++) {

cout << \*it << " ";

}

cout << endl << endl;

}

Pair average(Queue& q) {

Ls list = copyQueueToVector(q);

int n = 1;

int sumInt = q.top().val1;

double sumDouble = q.top().val2;

q.pop();

while (!q.empty()) {

sumInt += q.top().val1;

sumDouble += q.top().val2;

q.pop();

n++;

}

q = copyVectorToQueue(list);

Pair p(sumInt / n, sumDouble / n);

return p;

}

void addToQueue(Queue& q, Pair el, int pos) {

Ls list;

Pair val;

int i = 0;

while (!q.empty()) {

val = q.top();

if (i == pos) {

list.push\_back(el);

}

list.push\_back(val);

q.pop();

i++;

}

q = copyVectorToQueue(list);

}

void deleteByKeyInRange(Queue& q, int start, int end) {

Ls list;

while (!q.empty()) {

Pair val = q.top();

if (val.val1 < start || val.val1 > end) {

list.push\_back(val);

}

q.pop();

}

q = copyVectorToQueue(list);

}

void subAverage(Queue& q, Pair aver) {

Ls list = copyQueueToVector(q);

for (auto it = list.begin(); it != list.end(); it++) {

it->val1 += aver.val1;

it->val2 += aver.val2;

}

q = copyVectorToQueue(list);

}

int main() {

Queue q;

// Making queue

int n;

cout << "Enter n: "; cin >> n;

q = makeQueue(n);

printQueue(q);

// Calculating average

Pair aver = average(q);

cout << "Average: " << aver << endl;

int pos;

cout << "Position: ";

cin >> pos;

addToQueue(q, aver, pos);

cout << "After adding average on position: " << pos << endl;

printQueue(q);

// Deleting by key in range

int start, end;

cout << "Start: "; cin >> start;

cout << "End: "; cin >> end;

deleteByKeyInRange(q, start, end);

printQueue(q);

// Add average to each vector element

cout << "\nBefore adding average\n";

printQueue(q);

aver = average(q);

cout << "Average value: " << aver;

subAverage(q, aver);

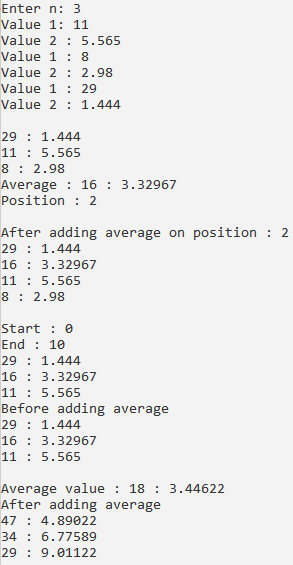
cout << "\nAfter adding average\n";

printQueue(q);

return 0;

}

**Результаты:**



**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.

**Код программы:**

#include <iostream>

#include <queue>

#include <list>

using namespace std;

template <class T>class List {

priority\_queue<T> queue; int len;

public:

List(); List(int n);

List(const List<T>&);

void Print(); ~List();

T Average(); void Add(T el, int pos);

void deleteByKeyInRange(int start, int end);

void subAverage(int aver);};

template <class T>void copyQueueToList(priority\_queue<T> q, list<T>& ls) {

while (!q.empty()) {

ls.push\_back(q.top());

q.pop(); }}

template <class T>void copyListToQueue(list<T> ls, priority\_queue<T>& q) {

for (auto it = ls.begin(); it != ls.end(); it++) {

q.push(\*it); }}

template <class T>List<T>::List() {len = 0;}

template <class T>List<T>::~List() {}

template <class T>List<T>::List(int n) {

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

cout << "Value " << i + 1 << ": "; cin >> el;

queue.push(el);}len = queue.size();}

template <class T>List<T>::List(const List<T>& item) {

len = item.len; list<T> ls; copyQueueToList(item.queue, ls); copyListToQueue(ls, queue);}

template <class T>void List<T>::Print() {

list<T> ls; copyQueueToList(queue, ls);

while (!queue.empty()) {

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

queue.pop(); } cout << endl;

copyListToQueue(ls, queue);}

template <class T>T List<T>::Average() {

T sum = 0; list<T> ls;

copyQueueToList(queue, ls);

for (auto it = ls.begin(); it != ls.end(); ++it) {

sum += \*it; }

int size = queue.size(); return sum / size;}

template <class T>void List<T>::Add(T el, int pos) {

list<T> ls; int i = 0; T val;

while (!queue.empty()) { val = queue.top();

if (i == pos) {

ls.push\_back(el);

}

ls.push\_back(val); queue.pop(); i++; }

copyListToQueue(ls, queue);}

template <class T>void List<T>::deleteByKeyInRange(int start, int end) {

list<T> ls; while (!queue.empty()) {

int val = queue.top();

if (val < start || val > end) {

ls.push\_back(val); }

queue.pop();}

copyListToQueue(ls, queue);}

template <class T>void List<T>::subAverage(int aver) {

list<T> ls; while (!queue.empty()) {

int val = queue.top() + aver;

ls.push\_back(val); queue.pop(); }

copyListToQueue(ls, queue);}

int main() {

// Making queue

List<int> ls(6); ls.Print();

// Calculating average

int aver = ls.Average(); cout << "Average: " << aver << endl;

int pos; cout << "Position: ";

cin >> pos; ls.Add(aver, pos);

cout << "After adding average on position: " << pos << endl;

ls.Print(); // Deleting by key in range

int start, end;

cout << "Start: "; cin >> start;

cout << "End: "; cin >> end;

ls.deleteByKeyInRange(start, end);

ls.Print(); // Add average to each vector element

cout << "\nBefore adding average\n";

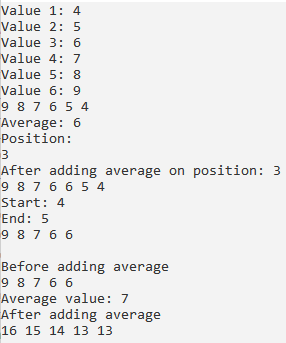
ls.Print(); aver = ls.Average(); cout << "Average value: " << aver;

ls.subAverage(aver); cout << "\nAfter adding average\n"; ls.Print();

return 0;

}

**Результат работы программы:**



**Вывод:** научился использовать последовательные контейнеры библиотеки STL в ОО программе.