Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Тема: «АТД. Контейнеры»

Выполнил работу

Студент группы РИС-22-2Б

Третьяков Н. А

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

1. Определить класс-контейнер.

2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.

3. Перегрузить операции, указанные в варианте.

4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.

5. Написать тестирующую программу, иллюстрирующую выполнение операций.

**Ответы на контрольные вопросы**

1. Абстрактный тип данных (АТД) - это математическая модель, описывающая поведение некоторой абстрактной структуры данных и набор операций, которые могут быть выполнены над этой структурой. Примеры АТД: классы, стек, очередь, дерево, граф.
2. Абстракция через параметризацию - это создание обобщенной структуры данных, которая может работать с разными типами данных. Примеры: шаблоны классов и функций в C++.
3. Абстракция через спецификацию - это определение интерфейса для работы со структурой данных, скрывающее детали реализации. Примеры: абстрактные классы в С++.
4. Контейнер - это структура данных, которая хранит и управляет коллекцией элементов определенного типа. Примеры: вектор, список, множество, словарь.
5. Группы операций, выделяемых в контейнерах, включают вставку, удаление, поиск, доступ к элементам, сортировку и обход элементов.
6. Существуют два вида доступа к элементам контейнера: доступ по индексу (например, в векторе) и доступ через итераторы (например, в списке).
7. Итератор - это объект, который позволяет последовательно перебирать элементы контейнера. Он предоставляет интерфейс для доступа к элементам контейнера и управления их обходом.
8. Итератор может быть реализован как указатель на элемент контейнера, как объект с методами next() и hasNext(), как указатель на функцию и т.д.
9. Объединение контейнеров можно организовать, используя операции объединения, пересечения и разности множеств, которые определены для соответствующих контейнеров.
10. Контейнер, состоящий из элементов "ключ-значение", предоставляет доступ к элементам по ключу.
11. Контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера, называется стеком.
12. Из объектов (a,b,c,d) только c. struct {char name[30]; int age;} mas; является контейнером.
13. Из объектов (a,b,c,d) только d. int mas; не является контейнером.
14. Если контейнер реализован как динамический массив и имеет операцию доступа по индексу, то доступ к элементам осуществляется путем указания индекса элемента. Например, чтобы получить значение элемента с индексом i, нужно написать container[i].
15. Если контейнер реализован как линейный список, то доступ к элементам осуществляется путем перебора всех элементов списка от начала до нужного элемента.

**Диаграмма класса**

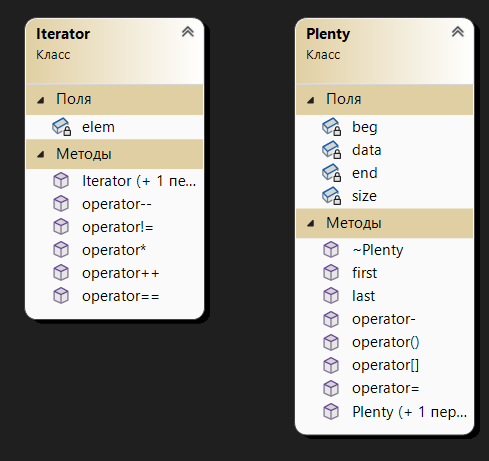


Рисунок 1 – классы Plenty и Iterator

**Код программы**

Plenty.h

#pragma once

#include <iostream>

using namespace std;

class Iterator

{

friend class Plenty;

int\* elem;

public:

Iterator() { elem = 0; };

Iterator(const Iterator& it) { elem = it.elem; };

bool operator == (const Iterator& it) { return elem == it.elem;};

bool operator != (const Iterator& it) { return elem != it.elem; };

void operator++() { ++elem; };

void operator--() { --elem; };

int& operator\*() const { return\*elem; };

};

class Plenty

{

int size;

int\* data;

Iterator beg;

Iterator end;

public:

Plenty(int s);

Plenty(const Plenty&);

Plenty& operator = (const Plenty&);

Plenty& operator-(const Plenty&);

int& operator[](int index);

int operator() ();

friend ostream& operator<<(ostream& out, const Plenty&);

friend istream& operator>>(istream& in, Plenty&);

Iterator first() { return beg; };

Iterator last() { return end; };

~Plenty();

};

Plenty.cpp

#include <iostream>

#include "Plenty.h"

using namespace std;

Plenty::Plenty(int s)

{

size = s;

data = new int[size];

for (int k = 0; k < s; k++)

data[k] = 0;

beg.elem = &data[0];

end.elem = &data[size];

}

Plenty::Plenty(const Plenty& p)

{

size = p.size;

data = new int[size];

for (int k = 0; k < size; k++)

data[k] = p.data[k];

beg.elem = &data[0];

end.elem = &data[size];

}

Plenty::~Plenty()

{

delete[] data;

size = 0;

}

Plenty& Plenty::operator=(const Plenty& p)

{

if (this == &p) return \*this;

if (size > p.size)

{

for (int i = 0; i < p.size; i++)

{

data[i] = p.data[i];

}

}

else

{

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

{

data[i] = p.data[i];

}

}

return \*this;

}

Plenty& Plenty::operator-(const Plenty& p)

{

if (size != p.size)

{

cout << "" << endl;

}

else

{

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

{

data[i] -= p.data[i];

}

return \*this;

}

}

int& Plenty::operator[](int index)

{

if (index < size) return data[index];

else cout << "" << endl;

}

int Plenty::operator()()

{

return size;

}

ostream& operator<<(ostream& out, const Plenty& p)

{

for (int i = 0; i < p.size;++i)

{

out << p.data[i] << " ";

}

return out;

}

istream& operator>>(istream& in, Plenty& p)

{

for (int i = 0; i < p.size; ++i)

{

in >> p.data[i];

}

return in;

}

Main.cpp

#include <iostream>

#include "Plenty.h"

using namespace std;

int main()

{

Plenty p(5);

cout << p << endl;

cin >> p;

cout << p << endl;

p[2] = 100;

cout << p << endl;

Plenty pl(10);

pl = p;

cout << pl << endl;

cout << "" << p() << endl;

cout << \*(p.first()) << endl;

Iterator i = p.first();

++i;

cout << \*i << endl;

--i;

cout << \*i << endl;

for (i = p.first(); i != p.last(); ++i)

cout << \*i << endl;

return 0;

}