Министерство образования и науки Российской Федерации

Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №18.6**

Дисциплина: «Информатика»

Тема: Объектно-ориентированное программирование. АТД контейнеры

Вариант 8

Выполнил:

Студент группы РИС-20-1б

Тараканов Д. М.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

Пермь

2021 год

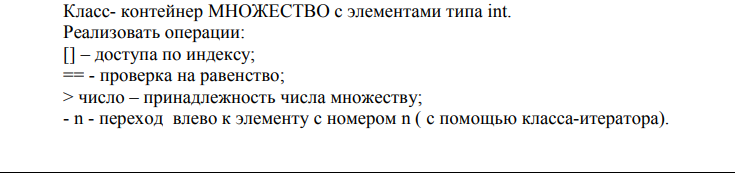
# **Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация класса-контейнера.

# **Постановка задачи**

1. Определитель класс-контейнер.
2. Реализовать конструкторы, деструкторы, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
5. Перегрузить операции ввода и вывода объектов.
6. Определить производный класс.
7. Написать демонстрационную программу, иллюстрирующую выполнение задачи.

(8 вариант)



# **Анализ задачи**

1. Определить какие действия предстоит выполнить:

* Разработать класс Set с полями data, size, max\_size, begin, end.
* Разработать класс Iterator с полем element дружественным к классу Set.
* Организовать методы для ввода данных, конструкторы и деструкторы, перегрузки операторов для класса Set и Iterator.

class Iterator {

friend class Set;

private:

int\* element;

public:

Iterator() {

element = 0;

}

Iterator(const Iterator& it) {

element = it.element;

}

bool operator == (const Iterator& it){

if (element == it.element) {

return true;

}

else false;

}

void operator ++ () {

++element;

}

void operator -- () {

--element;

}

void operator - (int shift) {

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

--element;

}

}

int& operator \*() {

return\*element;

}

};

class Set {

private:

int\* data;

int size;

int max\_size;

Iterator begin;

Iterator end;

public:

Set();

Set(Set&);

~Set();

int get\_size() {

return size;

}

Iterator first() {

return begin;

}

Iterator last() {

return end;

}

Set& operator = (const Set&);

void Insert(int);

int operator [] (int);

bool operator == (const Set&);

bool operator > (int);

friend istream& operator >> (istream&, Set&);

friend ostream& operator << (ostream&, const Set&);

};

* Протестировать работу контейнера в функции main.

void main() {

system("chcp 1251");

Set a, b;

int number, size\_a;

cout << "\nЗаполнение множества a\n"; cin >> a; cout << "\nМножество a " << a;

b.Insert(12);

b.Insert(4);

b.Insert(10);

b.Insert(0);

system("pause");

cout << "\nЭлемент\n> "; cin >> number;

a.Insert(number);

cout << "\nЭлемент " << number << " добавлен во множество a\n";

system("pause");

cout << "\nСравнение множеств\n\nМножество a " << a << "\nМножество b " << b;

if (a == b) {

cout << "\nМножества равны!\n";

}

else {

cout << "\nМножества не равны!\n";

}

system("pause");

cout << "\nПоиск числа во множестве a\nЧисло\n> "; cin >> number;

if (a > number) {

cout << "\nЧисло во множестве есть!\n";

}

else {

cout << "\nЧисла во множестве нет!\n";

}

system("pause");

int size = a.get\_size();

cout << "\nДоступ к элементу множества a\n\nНомер элемента";

do {

cout << "\n> ";

cin >> number;

if (number > size) {

cout << "\nВведите индекс меньший длины множества\n";

}

if (number < 1) {

cout << "\nВведите положительный индекс\n";

}

} while (number > size || number < 1);

cout << "\na[" << number << "] = " << a[number-1] << endl;

system("pause");

cout << "\nМножество a\n\nПервый элемент множества - " << \*(a.first()) << "\nПоследний элемент множества - " << \*(a.last()) << endl;

Iterator it = a.last();

size\_a = a.get\_size();

cout << "\nИтератор = " << \*(it);

cout << "\nДлина множества а = " << size\_a << "\n\nНа сколько элементов сдвинуть итератор?\n";

do {

cout << "\nЧисло\n> "; cin >> number;

if (number >= size\_a) {

cout << "\nВведите число меньше длины множества\n";

}

if (number < 0) {

cout << "\nВведите положительное число\n";

}

} while (number >= size\_a || number < 0);

it - number;

cout << "\nИтератор = " << \*(it) << endl;

system("pause");

}

1. C какими типами данных предстояло работать:

* Для хранения данных реализована переменная data типа int\*.

int\* data;

* Для хранения длины множества реализована переменная size типа int.

int size;

* Для хранения максимальной длины множества реализована переменная max\_size типа int.

int max\_size;

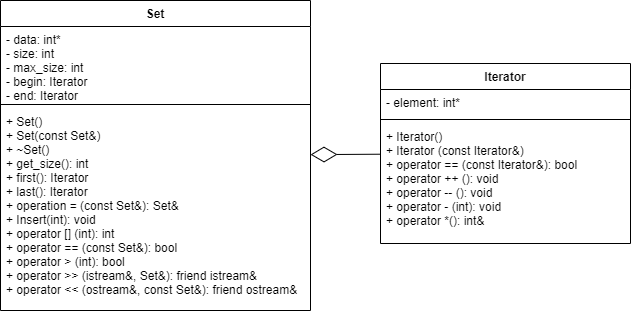
* Для хранения указателя на первый и последний элемент множества реализованы переменные begin и end типа Iterator.

Iterator beg, end;

* Для хранения значения итератора реализована переменная element типа int\*.

int\* size;

# **UML диаграмма**



# **Код программы на языке C++**

Заголовочный файл Set.h

#pragma once

#include <iostream>

using namespace std;

const int Lenght = 10;

class Iterator {

friend class Set;

private:

int\* element;

public:

Iterator() {

element = 0;

}

Iterator(const Iterator& it) {

element = it.element;

}

bool operator == (const Iterator& it){

if (element == it.element) {

return true;

}

else false;

}

void operator ++ () {

++element;

}

void operator -- () {

--element;

}

void operator - (int shift) {

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

--element;

}

}

int& operator \*() {

return\*element;

}

};

class Set {

private:

int\* data;

int size;

int max\_size;

Iterator begin;

Iterator end;

public:

Set();

Set(Set&);

~Set();

int get\_size() {

return size;

}

Iterator first() {

return begin;

}

Iterator last() {

return end;

}

Set& operator = (const Set&);

void Insert(int);

int operator [] (int);

bool operator == (const Set&);

bool operator > (int);

friend istream& operator >> (istream&, Set&);

friend ostream& operator << (ostream&, const Set&);

};

Обьявление класса в Set.cpp

#include "Set.h"

#include <iostream>

#include <algorithm>

Set::Set() {

size = 0;

max\_size = Lenght;

data = new int [max\_size];

begin.element = &data[0];

end.element = &data[size];

}

Set::Set(Set& temp\_Set) {

size = temp\_Set.size;

max\_size = temp\_Set.max\_size;

data = new int[max\_size];

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

data[i] = temp\_Set.data[i];

}

begin = temp\_Set.begin;

end = temp\_Set.end;

}

Set::~Set() {

delete[] data;

}

Set& Set::operator = (const Set& temp\_Set) {

if (\*this == temp\_Set) {

return \*this;

}

size = temp\_Set.size;

delete[] data;

data = temp\_Set.data;

begin = temp\_Set.begin;

end = temp\_Set.end;

return \*this;

}

void Set::Insert(int value) {

int i;

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

if (data[i] == value) {

i = size;

}

}

if (i != size + 1) {

data[size] = value;

if (i > 0) {

sort(data, data + size + 1);

}

begin.element = &data[0];

end.element = &data[size];

size++;

if (size == max\_size) {

max\_size += max\_size;

int\* temp\_data = new int[max\_size];

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

temp\_data[i] = data[i];

}

delete[] data;

data = temp\_data;

}

}

}

int Set::operator [] (int index) {

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

return data[index];

}

else {

cout << "\nОшибка! Неизвестный индекс!\n";

return -1;

}

}

bool Set::operator == (const Set& temp\_Set) {

if (size == temp\_Set.size) {

int i;

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

if (data[i] != temp\_Set.data[i]) {

i = size;

}

}

if (i == size) {

return true;

}

}

return false;

}

bool Set::operator > (int value) {

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

if (data[i] == value) {

return true;

}

}

return false;

}

istream& operator >> (istream& in, Set& temp\_Set) {

int number, quantity;

cout << "\nКол-во элментов - "; in >> quantity;

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

cout << i + 1 << ") ";

in >> number;

temp\_Set.Insert(number);

}

return in;

}

ostream& operator << (ostream& out, const Set& temp\_Set) {

out << "{ ";

for (int i = 0; i < temp\_Set.size; i++) {

out << temp\_Set.data[i] << " ";

}

out << "}\n";

return out;

}

Main файл

#include <iostream>

#include "Set.h"

using namespace std;

void main() {

system("chcp 1251");

Set a, b;

int number, size\_a;

cout << "\nЗаполнение множества a\n"; cin >> a; cout << "\nМножество a " << a;

b.Insert(12);

b.Insert(4);

b.Insert(10);

b.Insert(0);

system("pause");

cout << "\nЭлемент\n> "; cin >> number;

a.Insert(number);

cout << "\nЭлемент " << number << " добавлен во множество a\n";

system("pause");

cout << "\nСравнение множеств\n\nМножество a " << a << "\nМножество b " << b;

if (a == b) {

cout << "\nМножества равны!\n";

}

else {

cout << "\nМножества не равны!\n";

}

system("pause");

cout << "\nПоиск числа во множестве a\nЧисло\n> "; cin >> number;

if (a > number) {

cout << "\nЧисло во множестве есть!\n";

}

else {

cout << "\nЧисла во множестве нет!\n";

}

system("pause");

int size = a.get\_size();

cout << "\nДоступ к элементу множества a\n\nНомер элемента";

do {

cout << "\n> ";

cin >> number;

if (number > size) {

cout << "\nВведите индекс меньший длины множества\n";

}

if (number < 1) {

cout << "\nВведите положительный индекс\n";

}

} while (number > size || number < 1);

cout << "\na[" << number << "] = " << a[number-1] << endl;

system("pause");

cout << "\nМножество a\n\nПервый элемент множества - " << \*(a.first()) << "\nПоследний элемент множества - " << \*(a.last()) << endl;

Iterator it = a.last();

size\_a = a.get\_size();

cout << "\nИтератор = " << \*(it);

cout << "\nДлина множества а = " << size\_a << "\n\nНа сколько элементов сдвинуть итератор?\n";

do {

cout << "\nЧисло\n> "; cin >> number;

if (number >= size\_a) {

cout << "\nВведите число меньше длины множества\n";

}

if (number < 0) {

cout << "\nВведите положительное число\n";

}

} while (number >= size\_a || number < 0);

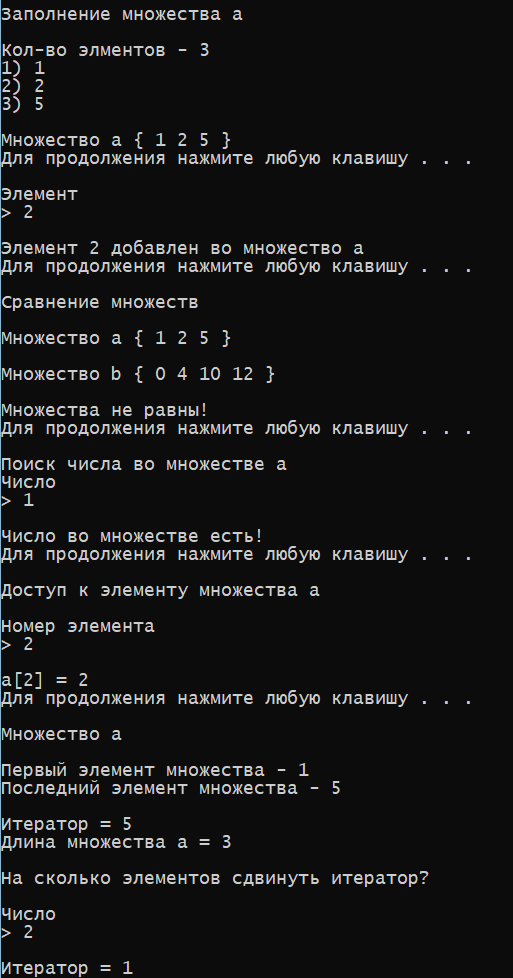
it - number;

cout << "\nИтератор = " << \*(it) << endl;

system("pause");

}

# **Скриншоты тестов**



**Контрольные вопросы**











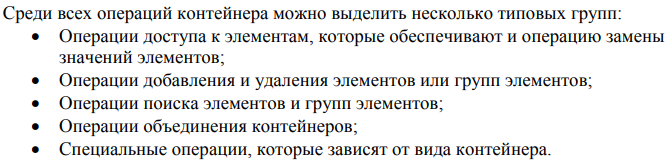




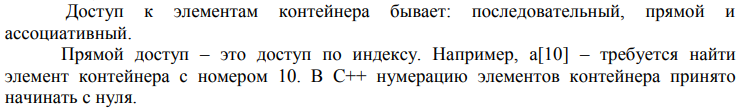


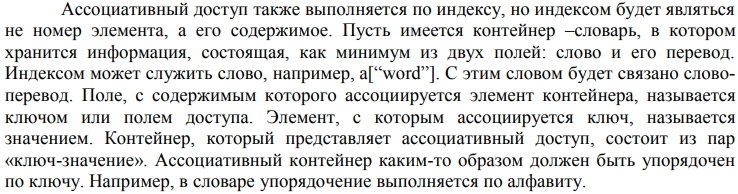


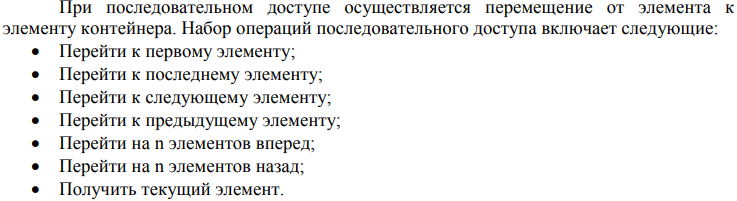




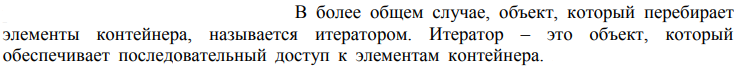






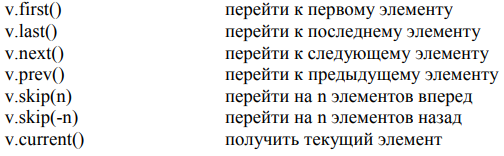




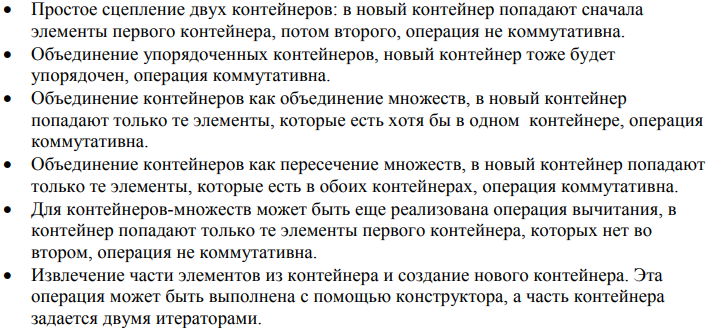




Итератор может быть реализован как часть класса-контейнера в виде набора методов:





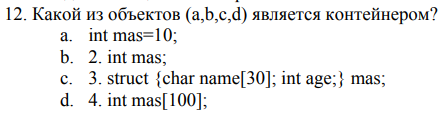




Ассоциативный доступ.



Стек.



d.





d.



Прямой доступ.



Последовательный доступ.