Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 6

Тема: «АТД. Контейнеры»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*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.

*Задание вариант 1:*

Класс – контейнер ВЕКТОР с элементами типа int

Реализовать операции:

[] – доступ по индексу;

() – определение размера вектора;

+ число – добавляет константу ко всем элементам вектора;

++ - переход к следующему элементу (с помощью класса-итератора).

*Анализ задачи:*

Основываясь на задании необходимо реализовать 2 класса, один итератор, для доступа к элементам вектора и сам класс вектор. Класс вектор будет иметь один конструктор по умолчанию, а также соответствующие перегруженные операции из варианта задания. Основные поля класса m\_data – указатель на область памяти, m\_capacity – кол-во выделенной памяти из кучи, m\_size – кол-во элементов в векторе и private функция realloc(), которая позволяет динамически выделять память на базе capacity и копировать данные массива из старой памяти в новую.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

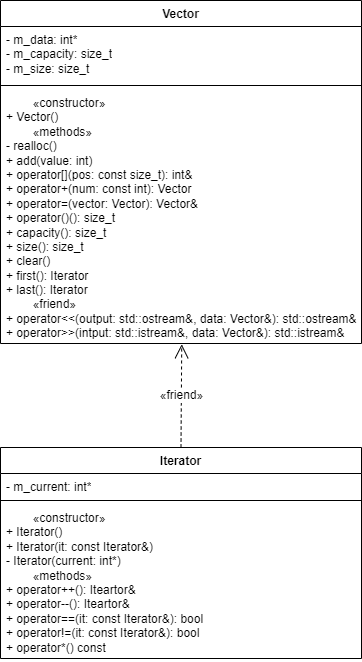


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

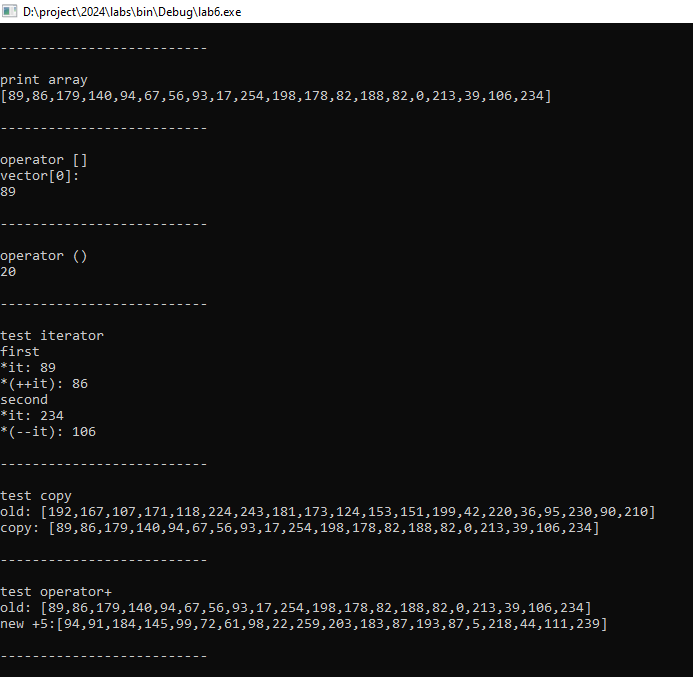


Рисунок 2 – Результат работы программы

*Контрольные вопросы:*

1. Что такое абстрактный тип данных? Привести примеры АТД.

Абстрактный тип данных (АТД) — это тип данных, который определяет данные и операции с ними, абстрагируясь от их реализации. Примеры АТД: стек, очередь, список, множество, дерево.

class Stack {

public:

void push(int value);

int pop();

bool isEmpty();

};

1. Привести примеры абстракции через параметризацию.

Абстракция через параметризацию позволяет создавать обобщенные типы, которые могут работать с различными типами данных.

template <typename T>

class Holder {

public:

void setValue(T value) { value\_ = value; }

T getValue() { return value\_; }

private:

T value\_;

};

1. Привести примеры абстракции через спецификацию.

Абстракция через спецификацию подразумевает описание интерфейса без указания реализации (например, используя чистые виртуальные функции).

class Shape {

public:

virtual double area() = 0; // чистая виртуальная функция

};

1. Что такое контейнер? Привести примеры.

Контейнер — это объект, который может хранить множество других объектов. Примеры: массивы, векторы, списки, карты.

1. Какие группы операций выделяют в контейнерах?

В контейнерах выделяют следующие группы операций:

- Вставка

- Удаление

- Доступ (чтение/запись)

- Обход

1. Какие виды доступа к элементам контейнера существуют? Привести примеры.

Существуют два основных вида доступа:

- Доступ по индексу (например, массивы, векторы)

- Доступ через итераторы (например, списки)

std::vector<int> vec = {1, 2, 3};

int firstElement = vec[0]; // доступ по индексу

1. Что такое итератор?

Итератор — это объект, который позволяет проходить по элементам контейнера и

выполнять операции с ними, предоставляя единый интерфейс доступа.

1. Каким образом может быть реализован итератор?

Итератор может быть реализован как класс с методом для перехода к следующему элементу (например, метод next() и оператор \* для разыменования).

class ArrayIterator {

public:

ArrayIterator(int\* ptr) : current(ptr) {}

int& operator\*() { return \*current; }

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

private:

int\* current;

};

1. Каким образом можно организовать объединение контейнеров?

Объединение контейнеров можно осуществить через методы, которые позволяют добавлять элементы одного контейнера в другой.

std::vector<int> vec1 = {1, 2, 3};

std::vector<int> vec2 = {4, 5, 6};

vec1.insert(vec1.end(), vec2.begin(), vec2.end());

10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов

«ключ-значение»?

Контейнер «ключ-значение», как правило, предоставляет доступ к элементам по ключу.

11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?

Такой контейнер называется стеком.

12. Какой из объектов (a,b,c,d) является контейнером?

a. int mas=10;

b. 2. int mas;

c. 3. struct {char name[30]; int age;} mas;

d. 4. int mas[100];

с.

13. Какой из объектов (a,b,c,d) не является контейнером?

a. int a[]={1,2,3,4,5};

b. 2. int mas[30];

c. 3. struct {char name[30]; int age;} mas[30];

d. 4. int mas;

d. это просто переменная

14. Контейнер реализован как динамический массив, в нем определена операция

доступ по индексу. Каким будет доступ к элементам контейнера?

Доступ к элементам контейнера в этом случае осуществляется по индексу за константное время O(1). Это означает, что для получения элемента по индексу не требуется перебор элементов, и доступ происходит мгновенно, поскольку используются вычисления с указателем.

#include <iostream>

int main() {

int arr[] = {10, 20, 30, 40, 50}; // Динамический массив

int index = 2;

std::cout << "Элемент по индексу " << index << ": " << arr[index] << std::endl; // Доступ по индексу

return 0;

}

15. Контейнер реализован как линейный список. Каким будет доступ к элементам

контейнера?

В этом случае доступ к элементам контейнера осуществляется за линейное время O(n), так как для получения элемента по индексу необходимо пройти по всем предыдущим элементам списка до достижения нужного индекса.

#include <iostream>

struct Node {

int data;

Node\* next;

};

int getElement(Node\* head, int index) {

Node\* current = head;

for (int i = 0; i < index && current != nullptr; ++i) {

current = current->next;

}

return current ? current->data : -1; // Возвращаем -1, если индекс вне диапазона

}

int main() {

// Создаем линейный список: 10 -> 20 -> 30

Node\* head = new Node{10, new Node{20, new Node{30, nullptr}}};

int index = 1;

std::cout << "Элемент по индексу " << index << ": " << getElement(head, index) << std::endl; // Доступ по индексу

return 0;

}

**ПРИЛОЖЕНИЕ 1**

lab6.cpp

//-------------------------------------

#include <iostream>

#include <ctime>

#include "lab6\_vector.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

srand(time(0));

Vector vector;

for(size\_t i = 0; i < 20; ++i)

vector.add(std::rand()%256);

IDENT\_PRINT;

//print array

std::cout << "print array" << std::endl;

std::cout << vector << std::endl;

IDENT\_PRINT;

//operator []

std::cout << "operator []\nvector[0]:" << std::endl;

std::cout << vector[0] << std::endl;

IDENT\_PRINT;

//operator()

std::cout << "operator ()" << std::endl;

std::cout << vector() << std::endl;

IDENT\_PRINT;

//test iterator

std::cout << "test iterator" << std::endl;

Iterator it = vector.first();

std::cout << "first\n";

std::cout << "\*it: " << \*it << std::endl;

std::cout << "\*(++it): " << \*(++it) << std::endl;

it = vector.last();

std::cout << "second\n";

std::cout << "\*it: " << \*it << std::endl;

std::cout << "\*(--it): " << \*(--it) << std::endl;

IDENT\_PRINT;

Vector vector1;

for(size\_t i = 0; i < 20; ++i)

vector1.add(std::rand()%256);

//test copy

std::cout << "test copy" << std::endl;

std::cout << "old: " << vector1 << std::endl;

vector1 = vector;

std::cout << "copy: " << vector1 << std::endl;

IDENT\_PRINT;

//test operator+

std::cout << "test operator+" << std::endl;

std::cout << "old: " << vector << std::endl;

vector = vector + 5;

std::cout << "new +5:" << vector << std::endl;

IDENT\_PRINT;

return 0;

}

//-------------------------------------

lab6\_vector.hpp

//-------------------------------------

#ifndef LAB6\_VECTOR\_HPP\_INCLUDED

#define LAB6\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <exception>

//-------------------------------------

#define BASE\_SIZE 4

//-------------------------------------

class Vector;

class Iterator {

friend Vector;

public:

Iterator() { m\_current = nullptr; }

Iterator(const Iterator& it) { m\_current = it.m\_current; }

~Iterator() { }

Iterator& operator++() { m\_current++; return \*this; }

Iterator& operator--() { m\_current--; return \*this; }

bool operator==(const Iterator& it) { return \*m\_current == \*it.m\_current; }

bool operator!=(const Iterator& it) { return !(\*this == it); }

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

private:

Iterator(int\* current) { m\_current = current; }

int\* m\_current;

};

//-------------------------------------

class Vector {

public:

Vector();

~Vector();

void add(int value);

int& operator[](const size\_t pos);

Vector operator+(const int num);

Vector& operator=(Vector vector);

size\_t operator()(){ return m\_size; }

size\_t capacity() { return m\_capacity; }

size\_t size() { return m\_size; }

void clear();

Iterator first() { return { m\_data }; }

Iterator last() { return { (m\_data + m\_size) - 1 }; }

friend std::ostream& operator<<(std::ostream& output, Vector& data);

friend std::istream& operator>>(std::istream& input, Vector& data);

private:

int\* m\_data;

size\_t m\_capacity,

m\_size;

void realloc();

};

//-------------------------------------

#endif // LAB6\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

lab6\_vector.cpp

//-------------------------------------

#include "lab6\_vector.hpp"

//-------------------------------------

Vector::Vector() {

m\_size = 0;

m\_capacity = 0;

this->realloc();

}

//-------------------------------------

Vector::~Vector() {

this->clear();

}

//-------------------------------------

void Vector::realloc() {

if(m\_size < m\_capacity)

return;

if(m\_capacity == 0) {

m\_capacity = BASE\_SIZE;

m\_data = new int[m\_capacity];

return;

}

m\_capacity\*=2;

int\* array = new int[m\_capacity];

for(size\_t i = 0; i < m\_size; ++i)

array[i] = m\_data[i];

delete []m\_data;

m\_data = array;

}

//-------------------------------------

void Vector::add(int value) {

this->realloc();

m\_data[m\_size] = value;

m\_size++;

}

//-------------------------------------

int& Vector::operator[](const size\_t pos) {

if(m\_size < pos)

throw std::range\_error("m\_size < pos");

return m\_data[pos];

}

//-------------------------------------

Vector Vector::operator+(const int num) {

Vector vector;

for(size\_t i = 0; i < this->size(); ++i)

vector.add((\*this)[i]+num);

return vector;

}

//-------------------------------------

Vector& Vector::operator=(Vector vector) {

if(&vector == this)

return \*this;

this->clear();

for(size\_t i = 0; i < vector.size(); ++i)

this->add(vector[i]);

return \*this;

}

//-------------------------------------

void Vector::clear() {

if(m\_data != nullptr)

delete []m\_data;

m\_size = 0;

m\_capacity = 0;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Vector& data) {

output << "[";

for(size\_t i = 0; i < data.size(); ++i) {

output << data[i];

if(data.size()-1 != i)

output << ",";

}

output << "]";

return output;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Vector& data) {

std::cout << "add " << data.size() << " elements: ";

int a;

input >> a;

data.add(a);

return input;

}

//-------------------------------------