**Отчет**

**по лабораторной работе №2**

**на тему**

**«Перегрузка операторов»**

Конов Михаил

группа ИУ8-24

**I. Цель работы**

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих перегрузку стандартных операций.

**II. Условие задачи**

Дан класс (например, с именем Vector), задающий вектор размерности n. Поля класса: указатель на массив, задающий вектор (тип элемента int или double в зависимости от варианта), массив должен создаваться динамически, число элементов (размерность) вектора (тип int). Класс включает: конструктор без параметров, задающий пустой вектор (число элементов равно 0), конструктор, создающий объект вектор на основе обычного одномерного массива размерности n, конструктор копирования, конструктор перемещения, деструктор.

Необходимо перегрузить операции и продемонстрировать их работу. Перегрузить операцию [] (обращение к элементу вектора по индексу), операцию = (присваивание с копированием), операцию = (присваивание с перемещением), а также операцию вставки (<<) объекта в поток cout или в файл (объект класса ostream) и операцию извлечения (>>) объекта из потока cin или из файла (объект класса istream).

**III. Вариант 10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание операции перегруженной операции | Тип элемента вектора (массива) | Типы операндов и результата для перегруженной операции | | |
| Первый операнд | Второй операнд | Результат |
| Префиксная форма ++ (инкремент), при этом каждый элемент исходного вектора увеличивается на 1, результат операции - измененный вектор. | int | Vector | \_\_\_\_\_ | Vector |

**IV. Текст программы**

#include <iostream>

#include <fstream>

using namespace std;

//тип массива - int

//Префиксная форма ++ (инкремент), при этом каждый элемент исходного вектора увеличивается на 1, результат операции - измененный вектор.

class Vector{

public:

int \*p;

int n;

int get\_n() const{return n;}

int\* get\_p() const{return p;}

//конструктор без параметров, задающий пустой вектор

Vector(){

p = nullptr;

n = 0;

p = new int[n];

cout<<"Vector()"<<endl;

}

Vector(int \*p, int n) //Коструктор на входе массив, задающий вектор

{

this->n = n; // Задаем число элементов

this->p = new int[n]; // Выделяем память

for (int i = 0; i < n; i++) this->p[i] = p[i]; // Копируем один массив в другой

cout<<"Vector(int \*p, int n)"<<endl;

}

//Конструктор копирования

Vector(const Vector& V)

{

n = V.n;

p = new int[n];

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

p[i] = V.p[i];

cout<<"Vector(const Vector & V) n="<<n<<endl;

}

// Конструктор перемещения

Vector(Vector&& V) // Параметр - правосторонняя ссылка

{

p = V.p; // Или инициализировать так

n = V.n;

// Присвойте данным-членам исходного объекта значения по умолчанию. Это не позволяет деструктору освобождать память

V.p = nullptr;

V.n = 0;

}

int& operator [](int index){

return p[index];

}

friend operator ++(Vector &V){

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

++V[i];

}

}

//разница между конструкторами копированием и перемещением

//разница между операциями присваивания с копированиям и перемещением

Vector & operator =(const Vector& v2) // Оператор- функция копирования объекта

{

if (this!=&v2) // Запрет копирования вектора самого в себя

{

n = v2.n;

if (p != nullptr) delete[] p; // Освобождаем память старого вектора

p = new int[n]; // Выделяем память для нового вектора

for (int i = 0; i < n; i++) p[i] = v2.p[i];

}

cout<<"Vector & operator =(const Vector& v2)"<<endl;

return \*this; // Возвращаем ссылку на текущий объект

}

Vector & operator =(Vector&& v2) // Оператор- функция перемещения объекта

{

if (this!=&v2) // Запрет перемещения вектора самого в себя

{

n = v2.n;

if (p != nullptr) delete[] p; // Освобождаем память старого вектора

p = v2.p; // Перемещаем данные

v2.p=nullptr; // Перемещаемый вектор пуст

}

cout<<"Vector & operator =(Vector&& v2)"<<endl;

return \*this; // Возвращаем ссылку на текущий объект

}

~Vector() // Деструктор

{

cout<<"~Vector() n="<<n <<endl;

if (p!=nullptr) delete[] p; // Освобождаем память

}

};

ostream& operator<< (ostream& stream, Vector& V){

stream << "V.n = ";

stream << V.get\_n() << endl;

if (V.get\_p() != nullptr){

for(int i = 0; i < V.get\_n(); ++i){

stream << "V.p[" << i << "] = ";

stream << V.get\_p()[i] << "\t";

}

}

else{

stream << "Вектор пуст!" << endl;

}

return stream;

}

istream& operator >> (istream& stream, Vector& V){

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

stream >> V.p[i];

}

return stream;

}

fstream& operator >> (fstream& stream, const Vector& V){

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

stream >> V.p[i];

}

return stream;

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

fstream vs\_file;

vs\_file.open("input.txt");

int quantity;

int \*l;

int cnt;

vs\_file >> cnt;

l = new int[cnt];

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

vs\_file >> l[i];

}

//открываем файл для вывода

fstream res\_file;

res\_file.open("output.txt");

res\_file << "Начало новой компиляции...\n";

//пример создания вектора из одномерного массива, считанного из файла

Vector V0(l, cnt);

res\_file << "Вектор V0: \n" << V0 << endl;

//пример создания пустого вектора, заполняемого данными из файла, используя перегрузку потокового оператора >>

vs\_file >> quantity;

res\_file << "quantity = " << quantity << endl;

Vector V1;

V1.n = quantity;

vs\_file >> V1;

res\_file << "Вектор V1: \n" << V1 << endl;

//ниже продемонстрирована работа перегрузки операции инкремента для объектов класса Vector

++V0;

res\_file << "Вектор 0 после применения операции инкремента: \n" << V0 << endl;

++V1;

res\_file << "Вектор 1 после применения операции инкремента: \n" << V1 << endl;

//в случае конструктора копирования, пустому вектору присваиваются значения из внешнего массива

//применим к новому вектору V2 конструктор копирования из V1

res\_file << "Применим конструктор копирования из вектора V1 для создания вектора V2: \n";

Vector V2(V1);

res\_file << "Вектор V2: \n" << V2 << endl;

res\_file << "Вектор V1: \n" << V1 << endl;

//как видим, значения вектора V2 не изменились, а V3 принял значения V2

//в случае конструктора перемещения, значения из используемого для перемещения вектора удаляются

res\_file << "Применим конструктор перемещения из вектора V1 для создания вектора V3: \n";

Vector V3(move(V1));

res\_file << "Вектор V3: \n" << V3 << endl;

res\_file << "Вектор V1: \n" << V1 << endl;

//значения вектора V1 переместились в вектор V3 и удалились из вектора V1

//разница между операциями с копированием и с перемещением

//оператор копирования присваивает значения вектора справа вектору слева, притом не изменяя вектор справа

res\_file << "Применим операцию копирования из вектора V3 в вектор V0:\n";

V0 = V3;

res\_file << "Вектор V0: \n" << V0 << endl;

res\_file << "Вектор V3: \n" << V3 << endl;

//оператор перемещения присваивает значения вектора справа вектору слева, но очищает вектор справа

res\_file << "Создадим вектор V5\n";

int m2[] = {2, 6, 7, 3, 2};

Vector V5(m2, 5);

res\_file << "Вектор V5: \n" << V5 << endl;

res\_file << "Переместим значения из него в V1:\n";

V1 = move(V5);

res\_file << "Вектор V1: \n" << V1 << endl;

res\_file << "Вектор V5: \n" << V5 << endl;

cout << "Конец программы." << endl;

return 0;

}