МИНИСТЕРСТВО РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

«КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ»

РЕФЕРАТ

по дисциплине: Основы алгоритмизации и программирования

на тему: Перегрузка операторов в C++

Выполнил:

студент 1 курса 7 группы

факультет ИТ

Кишко Иван Петрович

Проверил:

доцент, к.т.н.

Белодед Николай Иванович

Минск 2020

Оглавление

[Цели 3](#_Toc35100861)

[Введение 3](#_Toc35100862)

[**Какие операторы можно перегружать?** 3](#_Toc35100863)

[Перегрузка основных операций 4](#_Toc35100864)

[Оператор + 4](#_Toc35100865)

[Оператор == 4](#_Toc35100866)

[Оператор [] 5](#_Toc35100867)

[Перегрузка постфиксного инкремента 5](#_Toc35100868)

[Перегрузка префиксного инкремента 6](#_Toc35100869)

[Перегрузка операторов > и < 7](#_Toc35100870)

[Выводы 8](#_Toc35100871)

# Цели

Узнать ,что такое перегрузка операторов . Какие операторы можно перегружать ,а какие – нельзя. Зачем необходима перегрузка операторов и как она реализуется

# Введение

Оператор в С++ - это некоторое действие или функция, обозначенная специальным символом. Для того что бы распространять эти действия на новые типы данных, при этом сохраняя естественный синтаксис, в С++ была введена возможность перегрузки операторов.(+, -, <<, >>, == и так далее)

Перегрузка операторов в программировании дает возможность одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.

**Какие операторы можно перегружать?**

Мы можем перегрузить почти любой оператор C++, учитывая следующие исключения и ограничения:

•Нельзя определить новый оператор, например, operator\*\*.

•Следующие операторы перегружать нельзя:

1. ?: (тернарный оператор);
2. :: (доступ к вложенным именам);
3. . (доступ к полям);
4. ->(доступ к полям по указателю);
5. sizeof, typeid и операторы приведения типов.

• Следующие операторы можно перегрузить только в качестве методов:

* 1. = (присваивание);
  2. -> (доступ к полям по указателю);
  3. () (вызов функции);
  4. [] (доступ по индексу);
  5. ->\* (доступ к указателю-на-поле по указателю);
  6. операторы конверсии и управления памятью.
* Количество операндов, порядок выполнения и ассоциативность операторов определяется стандартной версией.
* Как минимум один операнд должен быть пользовательского типа. Typedef не считается.

# Перегрузка основных операций

## Оператор +

#include <iostream>

using namespace std;

struct point {

int x;

int y;

point operator+(const point& other)

{

point temp;//объект для возврата результата

temp.x = this->x + other.x; //this - указатель на текущий объект(На

temp.y = this->y + other.y; //объект,стоящий слева от оператора)

return temp;

}

point(int x, int y)//конструктор,принимающий начальные значения объекта

{

this->x = x;

this->y = y;

}point() {} //конструктор по умолчанию

};

int main()

{

point B3(10, 15), B4(1, 3), B5;

B5 = B3 + B4;

cout << "X = " << B5.x << " Y = " << B5.y << endl;

return 0;

}



## Оператор ==

#include <iostream>

using namespace std;

struct point {

int x;

int y;

point(int x, int y)//конструктор,принимающий начальные значения объекта

{

this->x = x;

this->y = y;

}

bool operator==(const point other)

{

return this->x == other.x && this->y == other.y;//Вернуть true если соответствующие координаты равны

}

};

int main()

{

point B1(3, 5), P(5, 12);

bool result = B1 == P;

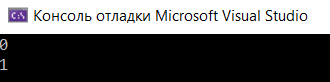
cout << result << endl; //result = 0

point H(4, 2), J(4, 2);

result = H == J;

cout << result << endl; //result = 1

}



## Оператор []

#include <iostream>

using namespace std;

class Point {

int coord[3];

public:

int& operator[](int index) //возврат по ссылке для возможности изменения данных

{

return coord[index];

}

Point(int x, int y, int z)

{

coord[0] = x;

coord[1] = y;

coord[2] = z;

}

};

int main()

{

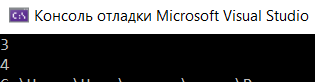
Point M(3, 4, 5);

cout<<M[0]<<endl;

M[0] = 4;

cout << M[0];

}



## Перегрузка постфиксного инкремента

#include <iostream>

using namespace std;

struct point {

int x;

int y;

point operator++(int value)//передача параметра необходима

{ //для отличия потсфиксной записи от префиксной

point temp(\*this);//конструктор копирования

this->x++;

this->y++;

return temp;

}

point(int x, int y)//конструктор,принимающий начальные значения объекта

{

this->x = x;

this->y = y;

}point() {}//конструктор по умолчанию

};

int main()

{

point A1(1, 3), B1;

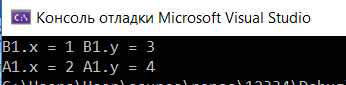
B1 = A1++;//сначала выполняется присваивание , а затем увеличение полей структуры А1

cout << "B1.x = " << B1.x << " B1.y = " << B1.y << endl;

cout << "A1.x = " << A1.x << " A1.y = " << A1.y;

point B3(10, 15), B4(1, 3), B5;

}



## Перегрузка префиксного инкремента

#include <iostream>

using namespace std;

struct point {

int x;

int y;

point operator++()//перегрузка постфиксного оператора

{

this->x++;

this->y++;

return \*this;

}

point(int x, int y)//конструктор,принимающий начальные значения объекта

{

this->x = x;

this->y = y;

}point() {}//конструктор по умолчанию

};

int main()

{

point A1(1, 3), B1;

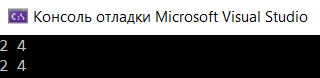
B1 = ++A1//сначала выполняется увеличение полей структуры А1,а затем присваивание

cout << B1.x << " " << B1.y << endl;

cout << A1.x << " " << A1.y;

point B3(10, 15), B4(1, 3), B5;

}



## Перегрузка операторов > и <

#include <iostream>

typedef unsigned short day;//создаем тип данных день на основе беззнакового типа short

typedef unsigned short month;//создаем тип данных месяц на основе беззнакового типа short

typedef unsigned short year;//создаем тип данных год на основе беззнакового типа short

struct Data

{

day dd;

month mm;

year yyyy;

bool operator>(Data temp)

{

return this->yyyy > temp.yyyy ||

this->yyyy == temp.yyyy && this->mm > temp.mm ||

this->yyyy == temp.yyyy && this->mm == temp.mm && this->dd > temp.dd;

}

bool operator<(Data temp)

{

return this->yyyy < temp.yyyy ||

this->yyyy == temp.yyyy && this->mm < temp.mm ||

this->yyyy == temp.yyyy && this->mm == temp.mm && this->dd < temp.dd;

}

};

int main()

{

setlocale(LC\_ALL, "ru");

Data date1 = { 7,1,1980 };

Data date2 = { 7,2,1993 };

Data date3 = { 7,1,1980 };

if (date1 < date2)std::cout << "Истина" << std::endl;

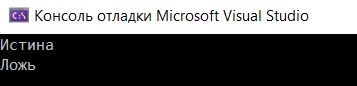
else std::cout << "Ложь" << std::endl;

if (date1 > date2)std::cout << "Истина" << std::endl;

else std::cout << "Ложь" << std::endl;

return 0;

}



# 

# Выводы

Перегрузка - это возможность поддерживать несколько функций с одним названием, но разными сигнатурами вызова. Проследил на примерах выше синтаксис реализации перегрузок. Уяснил ,какие операторы перегружать можно ,а какие нельзя. Таким образом благодаря перегрузке операторов мы можем выполнять со своими типами данных те же операции, что и со встроенными.