Программирование на C++

Минцифры России

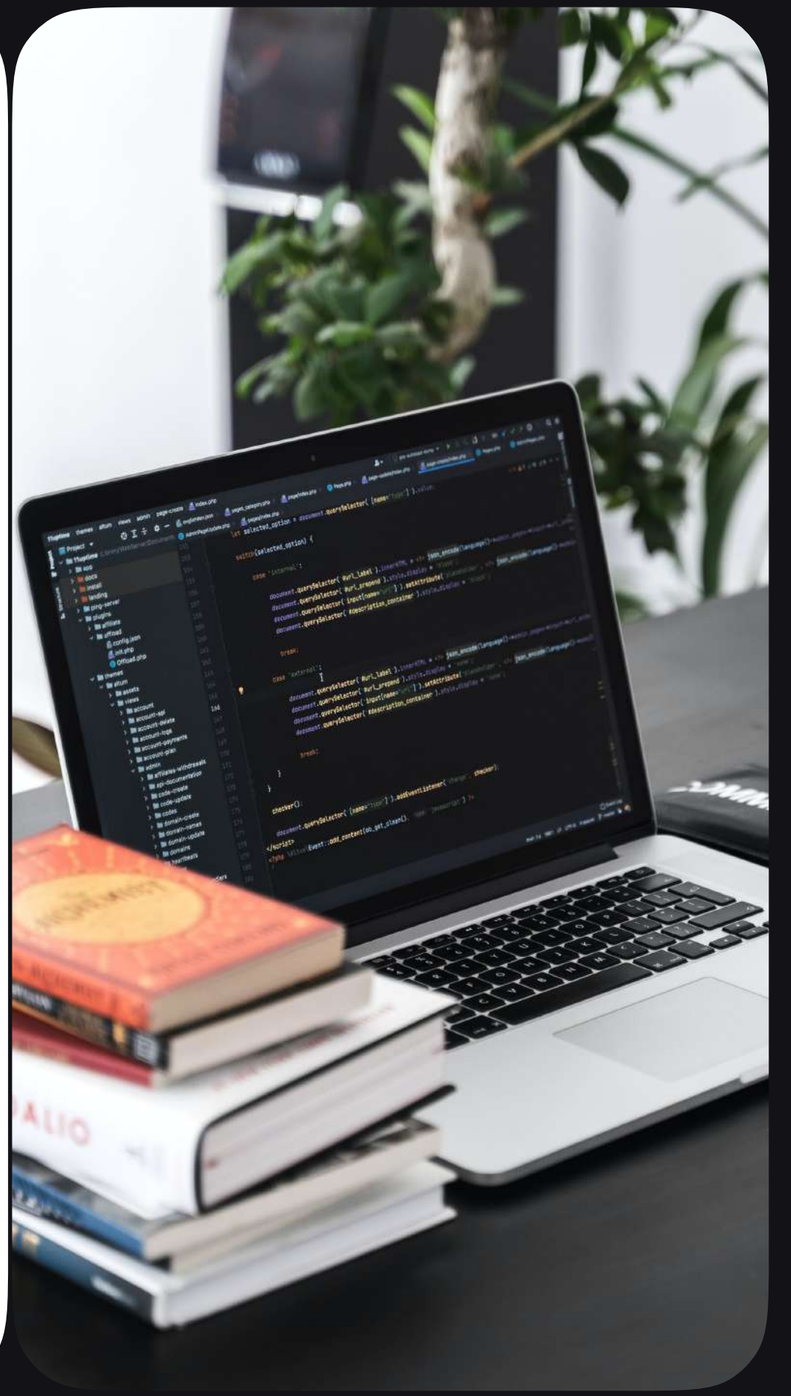
UCHi DOMA

20.35 УНИВЕРСИТЕТ

Урок 10 Модуль 3

Перегрузка операций в C++

Полезные материалы



Цели урока

- изучить перегрузку операций
- отработать на практике написание перегруженных операций на C++

Перегрузка операций

- Кроме перегрузки функций C++ позволяет организовать перегрузку операций.
- Механизм перегрузки операций позволяет обеспечить более традиционную и удобную запись действий над объектами.
- Для перегрузки встроенных операторов используется ключевое слово **operator**.

Перегрузка операций

Синтаксис перегрузки операции:

```
тип operator @ (список_параметров-операндов)
{
// тело функции
}
```

где @ — знак перегружаемой операции (-, +, * и т.д.)

тип — тип возвращаемого значения

Тип возвращаемого значения должен быть отличным от void, если необходимо использовать перегруженную операцию внутри другого выражения.

Перегрузка операций

Можно переопределять следующие операции:

+	–	*	/	%	^	&	|	~	!
=	<	>	+=	–=	*=	/=	%=	^=	&=
|=	<<	>>	>>=	<<=	==	!=	<=	>=	&&
||	++	––	[]	()	new	delete			

Перегрузка операций

Нельзя переопределять следующие операции:

.	прямой выбор члена объекта класса
.*	обращение к члену через указатель на него
? :	условная тернарная операция
::	операция указания области видимости (разрешение контекста)
sizeof	операция вычисления размера в байтах
#	препроцессорная операция

Правила перегрузки операций

- Язык C++ не допускает определения для операций нового лексического символа, кроме уже определенных в языке.
- Например, нельзя определить в качестве знака операции @.
- Не допускается перегрузка операций для встроенных типов данных. Нельзя, например, переопределить операцию сложения целых чисел:

```
int operator +(int i, int j)
```

- Нельзя переопределить приоритет операции.
- Нельзя изменить синтаксис операции в выражении. Например, если некоторая операция определена как унарная, то ее нельзя определить как бинарную. Если для операции используется префиксная форма записи, то ее нельзя переопределить в постфиксную.
- Например, **!a** нельзя переопределить как **a!**

Правила перегрузки операций

Перегружать можно только операции, для которых хотя бы один аргумент представляет тип данных, определенный пользователем. Функция-операция должна быть определена либо как функция-член класса, либо как внешняя функция, но дружественная классу.

Функция- член класса

```
class String
{
  ...
public:
  String operator + (const String &);
  ...
};
```

Дружественная функция

```
class String
{
  ...
public:
  friend String operator +(String &, String &);
  ...
};
```

Перегрузка унарной операции

Если унарная операция перегружается как функция-член, то она не должна иметь аргументов, так как в этом случае ей передается неявный аргумент-указатель this на текущий объект.

Если унарная операция перегружается дружественной функцией, то она должна иметь один аргумент — объект, для которого она выполняется.

Таким образом, для любой унарной операции @ aa@ или @aa может интерпретироваться или как aa.operator@(), или как operator @(aa).

Если определена и та, и другая, то и aa@ и @aa являются ошибками.

Функция- член класса

```
class A
{
  ...
public:
  A operator !();
  ...
};
```

Дружественная функция

```
class A
{
  ...
public:
  friend A operator !(A);
  ...
};
```

Перегрузка бинарной операции

Если бинарная операция перегружается с использованием метода класса, то в качестве своего первого аргумента она получает неявно переданную переменную класса (указатель this на объект), а в качестве второго — аргумент из списка параметров.

То есть, фактически бинарная операция, перегружаемая методом класса, имеет один аргумент (правый операнд), а левый передается неявно через указатель this.

Если бинарная операция перегружается дружественной функцией, то в списке параметров она должна иметь оба аргумента

Пример

Рассмотрим пример с классом Counter, который представляет секундомер и хранит количество секунд:

```
#include <iostream>
using namespace std;
class Counter
{
public:
    Counter(int sec)
    {
        seconds = sec;
    }
    void display()
    {
        cout << seconds << " seconds" << endl;
    }
int seconds;
};

Counter operator + (Counter c1, Counter c2)
{
    return Counter(c1.seconds + c2.seconds);
}

int main()
{
    Counter c1(20);
    Counter c2(10);
    Counter c3 = c1 + c2;
    c3.display();   // 30 seconds
    return 0;
}
```

Пример

Если функция оператора определена как член класса, то левый операнд доступен через указатель this и представляет текущий объект, а правый операнд передается в подобную функцию в качестве единственного параметра

```cpp
#include <iostream>
using namespace std;
class Counter
{
public:
    Counter(int sec)
    {
        seconds = sec;
    }
    void display()
    {
        cout << seconds << " seconds" << endl;
    }
    Counter operator + (Counter c2)
    {
        return Counter(this->seconds + c2.seconds);
    }
    int operator + (int s)
    {
        return this->seconds + s;
    }
int seconds;
};

int main()
{
    Counter c1(20);
    Counter c2(10);
    Counter c3 = c1 + c2;
    c3.display();           // 30 seconds
    int seconds = c1 + 25;  // 45
    return 0;
}
```