

**Объектно-ориентированное
программирование
с использованием языка**

C++



Урок №4

Объектно-
ориентированное
программирование
с использованием
языка C++

Содержание

Перегрузка инкремента и декремента	3
Перегрузка оператора индексирования.....	10
Заключительная глава	13
Домашнее задание	14

Перегрузка инкремента и декремента

В прошлом уроке мы с вами разбирали перегрузку операторов и рассматривали пример класса, реализующего работу со строкой. В этом классе была использована перегрузка бинарных операторов "=" и "+". Однако, нам с вами было бы неплохо рассмотреть перегрузку и унарных операторов. В частности, инкремента и декремента. Кроме свойства унарности, каждый из этих операторов обладает двумя формами, а это при их перегрузке имеет большое значение.

Следует отметить, что в начальных версиях языка C++ при перегрузке операций ++ и -- не делалось различия между постфиксной и префиксной формами. Например:

```
#include <iostream>
using namespace std;
class Digit{
    //Целое число.
    int N;

public:
    //Конструктор с параметром
    Digit(int n)
    {
        N = n;
    }

    //функция для показа числа на экран
    void display()
```

```

    {
        cout << "\nDigit: N = " << N << "\n";
    }
    //Компонентная функция (форма не различается):
    Digit& operator -- ()
    {
        //Уменьшаем содержимое объекта
        //в десять раз и возвращаем его
        //на место вызова оператора
        N /= 10;
        return *this;
    }
};

void main()
{
    //создаем объект Z именно с ним
    //мы и будем экспериментировать
    Digit Z(100);

    //показ объекта в первоначальном виде
    cout<<"\nObject Z (before):\n";
    Z.display();

    cout<<"\n-----\n";

    //присваиваем объекту Pref выражение
    //с префиксной формой (в данном случае
    //сначала изменится Z, а затем произойдет
    //присваивание).
    Digit Pref=--Z;

    //показываем результат работы
    //префиксной формы
    cout<<"\nPrefix\n";
    cout<<"\nObject Pref:\n";
    Pref.display();
}

```

```

    cout<<"\nObject Z (after):\n";
    Z.display();
    cout<<"\n-----\n";

    //присваиваем объекту Postf выражение
    //с постфиксной формой (в данном случае
    //(к сожалению) снова сначала
    //изменится Z, а затем произойдет
    //присваивание).
    Digit Postf=Z--;

    //показываем результат работы
    //постфиксной формы
    cout<<"\nPostfix\n";
    cout<<"\nObject Postf:\n";
    Postf.display();
    cout<<"\nObject Z (after):\n";
    Z.display();
}

```

Результат работы программы:

Object Z (before):
 Digit: N = 100

 Prefix
 Object Pref:
 Digit: N = 10
 Object Z (after):
 Digit: N = 10

 Postfix
 Object Postf:
 Digit: N = 1
 Object Z (after):
 Digit: N = 1

В современной же версии языка C++ принято следующее соглашение:

- Перегрузка префиксных операций ++ и -- ничем не отличается от перегрузки других унарных операций. Другими словами, функции конкретного класса: `operator++` и `operator--`, определяют префиксные операции для этого класса.
- При определении постфиксных операций "++" и "--" операции-функции должны иметь еще один дополнительный параметр типа `int`. Когда в программе будет использовано постфиксное выражение, то вызывается версия функции с параметром типа `int`. При этом параметр передавать не нужно!, а значение его в функции будет равно нулю.

```
#include <iostream>
using namespace std;

//Класс, реализующий работу с "парой чисел"
class Pair{
    //Целое число.
    int N;
    //Вещественное число.
    double x;
public:
    //Конструктор с параметрами
    Pair(int n, double xn)
    {
        N = n;
        x = xn;
    }

    //функция для показа данных на экран
    void display()
    {
        cout << "\nPair: N = " << N << " x = " << x << "\n";
    }
}
```

```

//Компонентная функция (префиксная --):
Pair& operator -- ()
{
    //Уменьшаем содержимое объекта
    //в десять раз и возвращаем его
    //на место вызова оператора
    N /= 10;
    x /= 10;
    return *this;
}

//Компонентная функция (постфиксная --):
Pair& operator -- (int k)
{
    //временно сохраняем содержимое
    //объекта в независимую
    //переменную типа Pair
    //(Попытка использовать здесь
    //значение дополнительного параметра
    //int k подтверждает его равенство 0.)
    Pair temp(0,0.0);
    temp.N=N+k;
    temp.x=x+k;

    //уменьшаем объект в 10 раз
    N /= 10;
    x /= 10;
    //возвращаем прежнее значение объекта.
    //таким "тактическим ходом"
    //мы добиваемся эффекта постфиксной
    //формы, т. е. в ситуации A=B++
    //в A записывается текущее
    //значение объекта B, тогда как сам B
    //изменяется
    return temp;
}
};

```

```

void main()
{
    //создаем объект Z именно с ним
    //мы и будем экспериментировать
    Pair Z(10,20.2);

    //показ объекта в первоизданном виде
    cout<<"\nObject Z (before):\n";
    Z.display();

    cout<<"\n-----\n";

    //присваиваем объекту Pref выражение
    //с префиксной формой (в данном случае
    //сначала изменится Z, а затем произойдет
    //присваивание) .
    Pair Pref=--Z;
    //показываем результат работы
    //префиксной формы
    cout<<"\nPrefix\n";
    cout<<"\nObject Pref:\n";
    Pref.display();
    cout<<"\nObject Z (after):\n";
    Z.display();

    cout<<"\n-----\n";

    //присваиваем объекту Postf выражение
    //с постфиксной формой (в данном случае
    //сначала произойдет присваивание,
    //а затем изменится Z) .

    Pair Postf=Z--;

    //показываем результат работы
    //постфиксной формы

```



```

    cout<<"\nPostfix\n";
    cout<<"\nObject Postf:\n";
    Postf.display();
    cout<<"\nObject Z (after):\n";
    Z.display();
}

```

Program output:

```

Object Z (before):
Pair: N = 10 x = 20.2
-----

```

```

Prefix
Object Pref:
Pair: N = 1 x = 2.02
Object Z (after):
Pair: N = 1 x = 2.02

```

```

-----
Postfix
Object Postf:
Pair: N = 1 x = 2.02
Object Z (after):
Pair: N = 0 x = 0.202

```

Примечание: В двух, вышеописанных примерах, мы не используем конструктор копирования, несмотря на то, что здесь присутствует инициализация одного объекта другим при создании. Это связано с тем, что в этом нет необходимости, так как здесь побитовое копирование не несет критического характера. Так что нет смысла перегружать код лишней конструкцией.

Перегрузка оператора индексирования

Только что мы с вами разобрали особенности перегрузки инкремента и декремента. Теперь поближе познакомимся с еще одним "особенным" оператором — оператором индексирования (`[]` — квадратные скобки).

Итак, вполне логично предположить, что выражение `A[i]`, где `A` — объект абстрактного типа `class`, представляется компилятором как `A.operator[](i)`. Рассмотрим пример:

```
#include <iostream>
using namespace std;
class A{

    //массив из 10 элементов
    //типа int
    int a[10];

    //размер массива
    int size;

public:

    //конструктор без параметров
    A(){
        size=10;
        for (int i = 0; i < 10; i++)

        //очевидно, что операция [], использованная
        //здесь, в конструкторе класса A, является
```

```

//стандартной, так как она выполняется над именем
//массива типа int.
        a [i] = i + 1;
    }
    //перегрузка оператора индексирования
    //возврат по ссылке осуществлен
    //для ситуации ОБЪЕКТ[i]=ЗНАЧЕНИЕ
    //на место вызова индексирования
    //вернется сам объект
    int& operator[] (int j){

        //возврат конкретного объекта
        return a [j];
    }
    //функция которая возвращает размер массива
    int Get () const {
        return size;
    }
};

void main () {
    int i,j;
    //Работа с одним объектом типа A
    A object;
    cout<<"\nOne object:\n";
    for(i=0;i<object.Get();i++){
        //выражение array[i] интерпретируется как
        //object.operator [] (j).
        cout<<object[i]<<" ";
    }

    cout<<"\n\n";

    //Работа с массивом объектов типа A
    A array [3];

```

```

cout<<"\nArray of objects:\n";
for (i=0;i<3;i++){
    for(j=0;j<object.Get();j++){

        //выражение array[i][j] интерпретируется как
        //(array [i]).operator [] (j).
        //Первая из двух операций [] является стандартной,
        //так как выполняется над именем массива.
        //При этом неважно, какой тип имеют его элементы.
        //Вторая операция [] - переопределенная,
        //так как результатом первой операции []
        //является объект типа A.

        cout << array [i][j] << " ";

    }
    cout <<"\n\n";
}
}

```

Результат работы программы:

One object:

1 2 3 4 5 6 7 8 9 10

Array of objects:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

Примечание: Обратите внимание!!! В данном примере мы не перегружаем двойные квадратные скобки для двумерного массива. Мы просто создаем массив объектов класса, в котором перегружен оператор [].

Заключительная глава

...для тех, кто уходит на специализацию, отличную от программирования. Все дело в том, что со следующего урока произойдет распределение по специализациям. И, те, студенты, которые выбрали дизайн и администрирование прощаются с нами. В связи с этим, сегодня вы имеете уникальную возможность подтянуть все свои "хвосты". А, именно, сдать экзамен по С, задания на который, получили в девятнадцатом уроке. А, также зачет по основам С++, подтверждением которого является наличие всех выполненных домашних заданий по данному курсу.

Кроме того, в данном уроке вы найдете тест по всем изученным темам курса С++ и домашнее задание для тех, кто остается с нами.

ЖЕЛАЕМ УДАЧИ!!!

Домашнее задание

1. Создайте класс динамического массива, в котором реализована проверка выхода за границы массива. Перегрузите операторы: `[]`, `=`, `+`, `-`, `++` (добавление элемента в конец массива), `--` (удаление элемента из конца массива).