top КОЛЛЕДЖ

Часть 4. Перегрузка операторов

Как выполнять операции +,-,*,/ и тому подобные с любыми данными

Вопрос

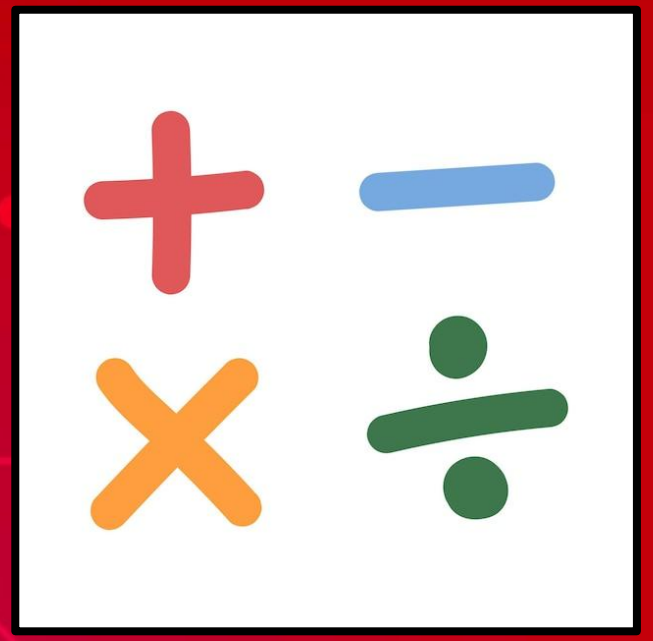
top КОЛЛЕДЖ

Какие операторы в языке C++ вы знаете?

Что такое оператор

top КОЛЛЕДЖ

- Итак, в языке C++ предусмотрено большое количество арифметических, логических, побитовых (и их составных версий), операторов сравнения и специальных операторов, которые вы часто используете в коде:
  - Арифметические: `+`, `-` (унарные и бинарные), `*`, `/`, `%`, `++`, `--`
  - Логические: `&&`, `||`, `!`
  - Побитовые: `&`, `|`, `^`, `~`, `<<`, `>>`
  - Сравнения: `==`, `!=`, `>`, `<`, `>=`, `<=`
  - Специальные: `=`, `new`, `delete`, круглые скобки `()`, квадратные скобки `[]`, запятая, точка, звездочка `*` (разыменование указателя), стрелка `->`, а также операторы преобразования типов.



Вопрос

top КОЛЛЕДЖ

Как вы думаете, какие операторы можно перегружать?

Специальные методы

top КОЛЛЕДЖ

- Все перечисленные операторы можно ***перегружать***, за исключением оператора точка, — то есть создавать для них свой собственные методы при использовании на пользовательском типе данных.
- Для перегрузки оператора необходимо определить в структуре метод, название которого начинается с ключевого слова `operator`.

```
struct Foo
{
    ...
    {возвращаемый тип} operator{оператор}({аргументы}) { … }
}
```

Специальные методы

top КОЛЛЕДЖ

- Исходя из того, к какому типу принадлежит оператор, можно понять его возвращаемый тип. Например, если пользователь создает структуру дроби `Fraction`, то умножение двух дробей, очевидно, также должно давать в результате дробь.
- Заметим, что если оператор бинарный, то для него указывается один аргумент, если унарный — то аргументы не указываются вовсе. По умолчанию левый операнд выражения является экземпляром структуры, над которой выполняется оператор.

```
struct Fraction
{
    int num;
    unsigned denom;

    // Вызывается с помощью a * b, где a и b это
    // переменные типа Fraction, a - данный
    // экземпляр (this), b эквивалентно other
    Fraction operator*(Fraction other)
    {
        Fraction result{};
        result.num = num * other.num;
        result.denom = denom * other.denom;
        return result;
    }
};

int main()
{
    Fraction a{ 1, 2 };
    Fraction b{ 3, 4 };

    // Вызывается метод operator*
    Fraction c = a * b;

    // Вот так это выглядит для компилятора
    Fraction c = a.operator*(b);
}
```

Унарные и бинарные операторы

top КОЛЛЕДЖ

- На прошлом слайде был представлен пример перегрузки бинарного оператора, для унарного же оператора при объявлении метода не нужно перечислять аргументы.
- Из этого правила есть одно исключение, связанное с постфиксными операторами инкремента и декременты.
- Для бинарных операторов правая часть может иметь значение другого типа.

```
// Унарный оператор +
// Пример: +value
Fraction operator+();

// Бинарный оператор +
// Пример: a + b
Fraction operator+(Fraction other);

// Унарный оператор -
// Пример: -value
Fraction operator-();

// Бинарный оператор -
// Пример: a - b
Fraction operator-(Fraction other);

// Бинарный оператор - (правый операнд целое число)
// Пример: a - 10
Fraction operator-(int other);
```

Инкремент и декремент

top КОЛЛЕДЖ

- Существует две формы операторов инкремента и декремента:

Префиксная

`++a` `--a`

Сначала выполняет операцию инкремента/декремента, затем возвращает полученное значение

Постфиксная

`a++` `a--`

Сначала возвращает текущее значение переменной, после этого выполняет операцию инкремента/декремента

Инкремент и декремент

top КОЛЛЕДЖ

- Для данных видов предусмотрено разное объявление оператора. Префиксный оператор возвращает ссылку на тип, а постфиксный — сам тип, и при этом имеет фиктивный аргумент типа `int` в скобках.
- *Примечание*: в циклах ради производительности предпочтительнее использовать префиксный оператор, так как из-за использования ссылки он не выполняет копирование.

```
// Префиксный оператор ++
// Пример: ++a
Fraction& operator++()
{
    num += denom;
    return *this;
}

// Постфиксный оператор ++
// Пример: a++
Fraction operator++(int)
{
    Fraction a{ num, denom };
    num += denom;
    return a;
}
```

Вопрос

top КОЛЛЕДЖ

Как определить бинарный оператор в случае, если левый операнд представляет собой другой тип?

Пример: 10 - a

Перегрузка для левого операнда

top КОЛЛЕДЖ

- Для левого операнда необходимо определять функцию с именем, начинающимся с ключевого слова operator, за пределами структуры. В таком случае, аргументы функции идут в порядке выполнения самой операции (сначала аргумент типа левого операнда, затем аргумент типа правого операнда).

```
}; // struct Fraction

// Бинарный оператор - (левый операнд целое число)
// Пример: 10 - a
Fraction operator-(int a, Fraction b);

int main()
{
    Fraction a{ 1, 2 };
    Fraction b{ 3, 4 };

    // Вызывается функция operator-
    Fraction c = 10 - b;

    c++;
}
```

Перегрузка оператора вывода

top КОЛЛЕДЖ

- С помощью функций, выполняющих перегрузку операторов для некоторой структуры, но не являющихся членами структуры, можно определить собственное поведение операторов для произвольных типов.
- Один из примеров — перегрузка оператора вывода. `std::cout`, как вы, наверное, уже догадались, является локальной переменной некоторого типа данных, и ее типа существует множество перегруженных операторов побитового сдвига влево <<. Данный оператор визуально похож на стрелку, направляющую в std::cout данные (то есть отправляет их на вывод), а также возвращает значение такого же типа, так как является бинарным.

Перегрузка оператора вывода

top КОЛЛЕДЖ

- `std::cout` имеет тип `std::ostream`, который нельзя копировать. То есть возвращаемое значение должно представлять собой ссылку на `std::ostream`. А входящее значение может иметь любой тип.
- Для перегрузки оператора вывода для типа Type можно определить следующую функцию:

```
std::ostream& operator<<(std::ostream& s, Type t)
{
    // Здесь осуществляем вывод в s для полей Type с
    // уже определенным оператором вывода
    …
    return s;
}
```

Пример

top КОЛЛЕДЖ

```
// Перегрузка оператора вывода для типа Fraction
std::ostream& operator<<(std::ostream& s, Fraction f)
{
    s << f.num << "/" << f.denom;
    return s;
}

int main()
{
    Fraction a{ 1, 2 };

    // Вывод дроби
    std::cout << "Value: " << a << std::endl;
}
```

```
Value: 1/2
```

Перегрузка преобразования типов

top КОЛЛЕДЖ

- Одним из перегружаемых операторов является оператор перегрузки преобразования типов, то есть приведения переменной одного типа к переменной другого с помощью преобразования в стиле C или `static_cast`.
- Для определения преобразования типа Foo к типу Bar необходимо в структуре Foo объявить метод следующего вида:

```
operator Bar()
{
    Bar b;
    // здесь преобразование полей Foo к полям Bar
    ...
    return b;
}
```

Пример

top КОЛЛЕДЖ

```cpp
struct Person
{
    std::string name;
    unsigned age;
};

struct Worker
{
    std::string workerName;
    std::string jobPost;
    unsigned workerAge;

    operator Person()
    {
        Person p{};
        p.age = workerAge;
        p.name = workerName;
        return p;
    }
};
```

```cpp
int main()
{
    Worker me{};
    me.workerName = "Emil";
    me.jobPost = "Teacher";
    me.workerAge = 23;

    // Вызывается оператор преобразования типа
    Person me2 = static_cast<Person>(me);

    std::cout << me2.name << ", " << me2.age;
}
```

```
Emil, 23
```

Функторы

top КОЛЛЕДЖ

- ***Функтор*** — это объект, который можно использовать как функцию, то есть вызывать. Для реализации такой возможности необходимо в структуре перегрузить оператор круглые скобки ( ), причем для данного оператора могут быть произвольные количество аргументов и возвращаемый тип, так как по сути он является той же самой функцией:

```
{возвращаемый тип} operator()({аргументы}) { … }
```

Пример

top КОЛЛЕДЖ

```
struct TryFunctor
{
    bool (*func)();

    // Вызвать функцию func attempts раз
    bool operator()(unsigned attempts)
    {
        for (unsigned i = 0; i < attempts; ++i)
        {
            std::cout << "Attempt #" << i << "...\n";
            if (func())
            {
                std::cout << "Success!\n";
                return true;
            }
        }
        std::cout << "Failure.\n";
        return false;
    }
};
```

```
// Проверка, что рандомное число от 0 до 49 - это 42
bool check42()
{
    unsigned value = rand() % 50;
    return value == 42;
}
```

```
int main()
{
    TryFunctor tryfunc{ check42 };

    // Используем экземпляр как функцию
    bool result = tryfunc(attempts: 5);

    if (result)
    {
        std::cout << "Let's go!" << std::endl;
        return 0;
    }

    std::cout << "No luck today..." << std::endl;
    return -1;
}
```

```
Attempt #0...
Attempt #1...
Attempt #2...
Attempt #3...
Attempt #4...
Failure.
No luck today...
```

Пользовательские литералы

top КОЛЛЕДЖ

- Существует также возможность создать пользовательский литерал с помощью синтаксиса перегрузки операторов. Некоторые литералы вы уже знаете, например литерал u, записанный после некоторого целого числа, преобразует его в беззнаковое число типа `unsigned int`.
- Для определения пользовательских литералов используется следующий синтаксис:

```
{возвращаемый тип} operator""{имя литерала}({изначальный тип}) { … }
```

- Стоит сказать, что литералы могут применятся только к символу, строке и числу, поэтому изначальным типом могут являться только:

```
const char *,  char,  unsigned long long,  long double.
```

*для типа `const char *` нужен еще второй аргумент типа `size_t`

Пример

top КОЛЛЕДЖ

```
// Литерал TO_LIST преобразует строку с запятыми в список строк
std::list<std::string> operator""TO_LIST(const char * v, size_t size)
{
    std::list<std::string> l;
    int prev = 0;
    for (int i = 0; i < size; ++i)
    {
        if (v[i] == ',')
        {
            l.push_back(_Val: std::string(v + prev, v + i));
            prev = i + 1;
        }
    }
    l.push_back(_Val: std::string(v + prev, v + size));
    return l;
}
```

```
one
two
three
four
five
```

```
int main()
{
    std::list<std::string> l = "one,two,three,four,five"TO_LIST;

    for (auto& std::string & v : l)
    {
        std::cout << v << "\n";
    }

    return 0;
}
```

Мини-задание

top КОЛЛЕДЖ

- Напишите перегрузку оператора / для типов `std::string`, так, чтобы в результате получалась строка, представляющая путь в файловой системе.
- Напишите перегрузку оператора вывода для типа `std::list<double>` так, чтобы вывод в консоль имел следующий формат:

```
[1.0 3.5 -2.2 …]
```

- Напишите перегрузку оператора преобразования типа `Fraction` в тип `double`.
- Напишите пользовательский литерал `oz`, который будет преобразовывать жидкие унции в миллилитры (в 1 унции — 28,41 мл жидкости).