Тема 2.7 Операции над объектами

Перегрузка операций

В C++ существует возможность перегрузить не только функции, но и операции внутри класса, например, можно добиться того, чтобы операция * при работе с матрицами осуществляла умножение матриц, а при работе с комплексными числами — умножение комплексных чисел.

Для перегрузки операций внутри класса нужно написать специальную функцию — метод класса. При перегрузке операций следует помнить следующее:

- нельзя поменять приоритет операций;
- нельзя изменить тип операции (из унарной операции нельзя сделать бинарную или наоборот);
- перегруженная операция является членом класса и может использоваться только в выражениях с объектами своего класса;
- нельзя создавать новые операции;
- запрещено перегружать операции: . (доступ к членам класса), унарную операцию * (значение по адресу указателя), :: (расширение области видимости), ?: (операция if);
- допустима перегрузка следующих операций: +, —, *, /, %, =, <, >, +=, -=, *=, /=,&&, ||, ++, —, (), [], new, delete.

Для перегрузки бинарной операции внутри класса необходимо создать функцию-метод:

```
1 type operator symbols(type1 parametr)
2 {
3 операторы;
4 }
```

Здесь type — тип возвращаемого операцией значения, operator — служебное слово, symbols — перегруженная операция, type1 — тип второго операнда, первым операндом является экземпляр текущего класса, parametr — имя переменной второго операнда.

Перегрузка функций в C++

Перегрузка функций в C++ используется, когда нужно сделать одно и то же действие с разными типами данных. Для примера, создадим простую функцию max, которая будет определять максимальное из двух целых чисел:

```
/* Функция max для целых чисел */
int max(int num1, int num2)
{
    if (num1 > num2)
        return num1;
    return num2;
}
```

В эту функцию мы можем передавать только целочисленные параметры. Для того, чтобы сделать аналог этой функции для чисел с плавающей запятой, выполним перегрузку этой функции:

```
/* Функция max для чисел с плавающей запятой */
double max(double num1, double num2)
{
    if (num1 > num2)
        return num1;
    return num2;
}
```

Теперь, когда мы будет вызывать функцию max с целыми параметрами, то вызовется первая функция. А если с вещественными — то вторая. Например:

```
// Здесь будет использоваться первый вариант функции max
int imax = max(1, 10);
// А здесь - второй
double dmax = max(1.0, 20.0);
```

Перегруженные конструкторы

Хотя конструкторы и предназначены для решения особо важных задач, они мало чем отличаются от обычных функций и могут быть перегружены. Как будет показано далее, во многих случаях перегрузка конструкторов позволяет добиться определённых преимуществ.

```
class timer {
  int seconds;
public:
// секунды задаются строкой
  timer(char *t) { seconds = atoi(t); }
// секунды задаются целым числом
  timer (int t) { seconds = t; }
// время задается в минутах и секундах
  timer (int min, int sec) { seconds = min*60 + sec; }
}
main{
  timer a(10), b(20), c(1, 10);
  ...
  ...
  return 0;
}
```

Как можно видеть, при создании объектов a, b и c внутри функции main() они получают начальное значение с использованием трех различных методов,

поддерживаемых перегруженными конструкторами. Каждый из них позволяет провести инициализацию для соответствующих данных.

Перегрузка операторов

Перегрузка операторов — в программировании это один из способов реализации полиморфизма, заключающийся в возможности 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.

Синтаксис перегрузки операторов очень похож на определение функции с именем operator@, где @ — это идентификатор оператора (например +, -, <<, >>).

Перегрузка унарных операторов

```
//префиксная версия возвращает значение после инкремента
const Integer& operator++(Integer& i) {
    i.value++;
    return i;
}
```

Перегрузка бинарных операторов

```
const Integer operator+(const Integer& left, const Integer& right) {
    return Integer(left.value + right.value);
}
```

Неперегружаемые операторы

Некоторые операторы в C++ не перегружаются в принципе (по всей видимости, это сделано из соображений безопасности):

- Оператор выбора члена класса ".";
- Оператор разыменования указателя на член класса ".*";
- В C++ отсутствует оператор возведения в степень (как в Fortran) "**";
- Запрещено определять свои операторы (возможны проблемы с определением приоритетов);
- Нельзя изменять приоритеты операторов.

Рекомендации к форме определения операторов

Существует два способа операторов — в виде функции класса и в виде дружественной глобальной функции.Роб Мюррей, в своей книге C++ Strategies and Tactics определил следующие рекомендации по выбору формы оператора:

Оператор	**Рекомендуемая форма**
Все унарные операторы	Член класса
= () [] -> ->*	Обязательно член класса
+= -= /= *= ^= &= |= %= >>= <<=	Член класса
Остальные бинарные операторы	Не член класса

Правила перегрузки операторов

- Не допускается определять новые операторы, такие как **.

- Не допускается переопределение операторов применительно ко встроенным типам данных.
- Перегруженные операторы должны быть нестатической функцией-членом класса или глобальной функцией.
- Операторы подчиняются правилам приоритета, группирования и числа операндов, определяемым их типичным использованием со встроенными типами.
- Операторы подчиняются правилам приоритета, группирования и числа операндов, определяемым их типичным использованием со встроенными типами.
- Унарные операторы, объявленные как функции-члены, не принимают аргументов; при объявлении как глобальные функции они принимают один аргумент.
- Бинарные операторы, объявленные как функции-члены, принимают один аргумент; при объявлении как глобальные функции они принимают два аргумента.
- Если оператор можно использовать и как унарный, и как бинарный оператор (&, *, + и -), каждый способ применения можно перегружать отдельно.
- Перегруженные операторы не могут иметь аргументов по умолчанию.
- Все перегруженные операторы, кроме оператора присваивания (operator=), наследуются производными классами.
- Первым аргументов операторов, перегруженных в виде функций-членов, всегда является тип класса объекта, для которого вызывается этот оператор (класса, в котором объявлен оператор, или класса, производного от этого класса).Для первого аргумента никакие преобразования не предоставляются.

Приоритеты операторов

Приоритет	Оператор	Описание	Ассоциативность
1	::	Область видимости	Слева-направо
2	++ -- () [] . ->	Суффиксальный/постфиксный инкремент и декремент Вызов функции Обращение к массиву по индексу Выбор элемента по ссылке Выбор элемента по указателю	
3	++ -- + - ! ~ (type) * & sizeof new, new[] delete, delete[]	Префиксный инкремент и декремент Унарный плюс и минус Логическое НЕ и побитовое НЕ Приведение к типу type Indirection (разыменование) Адрес Размер Динамическое выделение памяти Динамическое освобождение памяти	Справа-налево
4	.* ->*	Указатель на член	Слева-направо
5	* / %	Умножение, деление и остаток	
6	+ -	Сложение и вычитание	
7	<< >>	Побитовый сдвиг влево и сдвиг вправо	
8	< <= > >=	Операторы сравнения < и ≤ Операторы сравнения > и ≥	
9	== !=	Операторы сравнения = и ≠	
10	&	Побитовое И	
11	^	Побитовый XOR (исключающее или)	
12	|	Побитовое ИЛИ (inclusive or)	
13	&&	Логическое И	
14	||	Логическое ИЛИ	
15	?: = += -= *= /= %= <<= >>= &= ^= |=	Тернарное условие Прямое присваивание (предоставляемое по умолчанию для C++ классов) Присвоение с суммированием и разностью Присвоение с умножением, делением и остатком от деления Присвоение с побитовым сдвигом слево и вправо Присвоение с побитовыми логическими операциями (И, XOR, ИЛИ)	Справа-налево
16	throw	Throw оператор (выброс исключений)	
17	,	Запятая	Слева-направо