**СЕМИНАР 5. Перегрузка операций. Alpha. Часть 1**

**Теоретический материал**

Механизм перегрузки операций позволяет реализовать распространение действия стандартных операций языка на переменные (операнды), которые являются объектами абстрактных (пользовательских) типов данных (классов и структур). Это дает возможность применять к объектам классов и структур арифметические операции, операции сравнения и пр. как к обычным переменным стандартных типов данных языка.

Не могут быть перегружены следующие операции:

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| ?: | Условная операция |
| . | Операция точка (операция доступа к элементу) |
| sizeof | Операция определения размера |
| :: | Операция области видимости |
| .\* ->\* | Операции разрешения контекста |

При перегрузки может быть изменен смысл самой операции. То есть допустимо сделать так, что запись вида ***x+y*** физически будет выполнять вычитание или деление (***x*** и ***y*** – объекты класса). Естественно, не рекомендуется делать это на практике, это усложнит чтение и понимание кода программы.

По числу операндов все операции (за исключением тернарной операции :? ) могут быть разделены на *унарные* (с одним операндом) и *бинарные* (с двумя операндами).

Перегружаемые операции:

|  |  |  |
| --- | --- | --- |
| **Оператор** | **name** | **Тип** |
| **,** | запятая (разделитель элементов) | Бинарные |
| **!** | Логическое НЕ | Унарные |
| **!=** | Неравенство | Бинарные |
| **%** | Остаток от деления | Бинарные |
| **%=** | Назначение остатка от деления | Бинарные |
| **&** | Побитовое И | Бинарные |
| **&** | Взятие адреса | Унарные |
| **&&** | Логическое И | Бинарные |
| **&=** | Назначение побитового И | Бинарные |
| **( )** | Вызов функции | — |
| **( )** | Оператор приведения типа | Унарные |
| **\*** | Умножение | Бинарные |
| **\*** | Разыменование указателя | Унарные |
| **\*=** | Присваивание умножения | Бинарные |
| **+** | Сложение | Бинарные |
| **+** | Унарный плюс | Унарные |
| **++** | Инкремент1 | Унарные |
| **+=** | Присваивание сложения | Бинарные |
| **–** | Вычитание | Бинарные |
| **–** | Унарное отрицание | Унарные |
| **––** | Декремент | Унарные |
| **–=** | Присваивание вычитания | Бинарные |
| **–>** | Выбор члена | Бинарные |
| **–>\*** | Выбор указателя на член | Бинарные |
| **/** | Деление | Бинарные |
| **/=** | Присваивание деления | Бинарные |
| **<** | Меньше | Бинарные |
| **<<** | Сдвиг влево | Бинарные |
| **<<=** | Сдвиг влево с присваиванием | Бинарные |
| **<=** | Меньше или равно | Бинарные |
| **=** | Присваивание | Бинарные |
| **==** | Равенство | Бинарные |
| **>** | Больше | Бинарные |
| **>=** | Больше или равно | Бинарные |
| **>>** | Сдвиг вправо | Бинарные |
| **>>=** | Сдвиг вправо с присваиванием | Бинарные |
| **[ ]** | Операция взятия индекса | — |
| **^** | Исключающее ИЛИ | Бинарные |
| **^=** | Исключающее ИЛИ/присваивание | Бинарные |
| **|** | Побитовое включающее ИЛИ | Бинарные |
| **|=** | Назначение побитового включающего ИЛИ | Бинарные |
| **||** | Логическое ИЛИ | Бинарные |
| **~** | Дополнение до единицы | Унарные |
| **delete** | **Delete** | — |
| **new** | **New** | — |
| conversion operators  (type) | операторы преобразования | Унарные |

При перегрузке операций НЕВОЗМОЖНО изменить приоритет операций и её синтаксис (число операндов – сделать из унарной операции бинарную и наоборот). Также ЗАПРЕЩЕНО создавать и перегружать собственные операции (+?, \*\* и пр.). При перегрузке операций по крайней мере один из операндов должен иметь абстрактный тип – перегрузка операций для стандартных типов данных (int, float и пр.) НЕДОПУСТИМА.

Реализация перегрузки операций производится путем создания специальной, так называемой операторной функции с именем ***operator@***, где ***@*** - символ перегружаемого оператора.

<код возврата> ***operator@***(<аргументы>)

Для обеспечения связи операторной функции и класса, эта функция должна или являться членом класса, или быть внешней (для эффективности работы программы – дружественной) функцией с аргументами типа «объект класса» или «ссылка на объект класса».

Запись вида

***x@y***

интерпретируется компилятором как вызов функции

***operator@(x,y)*** – если перегрузка бинарной операции @ реализована в виде дружественной функции (у функции два аргумента);

***x.operator@(y)*** – если перегрузка бинарной операции @ реализована в виде компонентной функции класса (у функции один аргумент).

Вызов **бинарной** операции всегда реализует операнд, стоящий слева от операции. В случае реализации перегрузки компонентным методом он передается в операторную функцию через указатель ***this***. Таким образом, число аргументов такой операторной функции на 1 меньше, чем дружественной операторной функции (см. пример выше).

Перегруженная операция (бинарная или унарная), левым операндом которой является переменная стандартного типа (int, float и пр.), а не объект класса, НЕ МОЖЕТ быть компонентной функцией класса.

Например, запись вида x+1 (x – объект некоторого класса) интерпретируется компилятором как вызов операторной функции

***x.operator+(1)*** или ***operator+(x,1)***

Если поменять операнды местами, то при интерпретации записи 1+х (х – объект некоторого класса) возникнет ошибка:

***1.operator+(x)*** – ошибка (попытка вызова перегруженного оператора переменной стандартного типа).

***operator+(1,x)*** – допустимый вариант.

Из указанного выше примера следует важное правило: перегрузка бинарной операции, первым операндом которой является переменная стандартного типа

***<переменная\_стандартного\_типа> @ <объект класса>***

допустима только с помощью дружественных (внешних) операторных функций.

Перегрузка **унарной** операции $ также может быть реализована при помощи дружественной или компонентной функций. В первом случае функция будет иметь один аргумент, во втором – ноль (операнд будет передан в функцию через указатель this).

Так запись вида $x (x – объект некоторого класса, $ - унарная операция) будет интерпретироваться как

***operator$(x)*** – если перегрузка унарной операции $ реализована в виде дружественной функции (у функции один аргумент);

***x.operator$()*** – если перегрузка унарной операции $ реализована в виде компонентной функции класса (у функции нет аргументов).

Перегрузка постфиксных и префиксных форм унарных операций **++** и **--** имеют существенное отличие. В языке С++ принято правило, согласно которому перегрузка префиксных операций **++** и **--** не отличается от перегрузки других унарных операций. Операторная функция перегрузки постфиксных форм этих операций включает в себя дополнительный аргумент типа int. В случае перегрузки дружественной функцией, операторная функция имеет два аргумента (первый – «класс» или «ссылка на класс», а второй - int); у компонентной операторной функции будет только один аргумент типа int. Введенный в состав функции дополнительный вспомогательный целочисленный аргумент будет иметь произвольное значение, неоказывающее влияние на итог операции.

**friend clname& operator++(clname&);** // префиксный ++

**friend clname& operator--(clname&, int);** // постфиксный –-

**clname& operator--();** // префиксный –-

**clname& operator++(int);** // постфиксный ++

Следует отметить, что операторная функция может быть вызвана в явном виде, по имени, как и любая другая функция класса.

class clname {…}

int main()

{

clname x(10), y(20), z1, z2;

z1=x+y; // вызов перегруженной операции +

z2=x.operator+(y); // явный вызов операторной функции operator+

…

}

**Особенности перегрузки некоторых операций**

Операторы = (присваивание) и & (получение адреса) по умолчанию включаются в спецификацию класса, если они явно не определены пользователем.

Некоторые операции, такие как = (операция присваивания), -> (операция косвенного выбора компонента класса), [] (операция индексирования массива), () (операция вызова функции), можно переопределить только как методы класса.

***Перегрузка операции присваивания =***

Перегружающая операцию присваивания операторная функция может быть только нестатической функцией класса. Это позволяет гарантировать, что слева от оператора присваивания будет стоять реально существующий в памяти объект.

Создаваемый в классе по умолчанию перегруженный оператор присваивания должен иметь в качестве кода возврата ссылку на объект класса и в качестве аргумента константную ссылку на объект класса. Код возврата (return) в коде оператора присваивания должен возвращать \*this.

Перегруженный по умолчанию оператор присваивания реализует прямое последовательное присваивание одноименных полей одного объекта, другому объекту того же класса. Если класс содержит поля-указатели на динамически выделяемую память, то для корректной работы он должен быть явно переопределен. В противном случае, после присваивания оба объекта будут ссылаться на один и тот же участок памяти.

Иногда может возникнуть ситуация, когда требуется запретить присваивание объектов класса во внешних функциях (создаваемый по умолчанию перегруженный оператор присваивания позволяет это сделать). Для этого следует явно переопределить перегружающую оператор присваивания операторную функцию в приватной области определения класса.

Операция присваивания имеет правую ассоциативность. Это означает, что запись а=b=c , будет интерпретироваться как a=(b=с)). Правая ассоциативность этой операции обеспечивается возвратом ссылки на объект класса, от имени которого вызывается оператор-функция operator= и который указан с левой стороны от него.

***Перегрузка операции вызова функции ()***

Операция () также должна быть нестатической функцией класса.

Переопределение операции () позволяет использовать синтаксис вызова функции применительно к объекту класса (имя объекта с круглыми скобками). Количество операндов в скобках может быть любым.

Аргументы и возвращаемое значение операционной функции могут иметь любые типы, допустимые для функций. Функция operator() вызывается путем применения списка аргументов к объекту того класса, в котором он определен.

***Перегрузка операции индексирования массива [ ]***

Эта операция всегда определяется как нестатический член класса и, так как подразумевается поведение индексируемого объекта как массива, то ей следует возвращать ссылку. Это позволяет использовать эту операцию не только в правой, но и в левой части операции присваивания, облегчая доступ к необходимым элементам данных.

Переопределение операции [] позволяет использовать синтаксис доступа к элементам массива (имя объекта с квадратными скобками).

***Перегрузка операции запятая ,***

В число операций с необычным синтаксисов перегрузки входит также операция запятая. Она вызывается для объектов, между которыми поставлена запятая, но он не вызывается в списках аргументов функций. При перегрузке следует понимать и различать эти случаи.

Операция «запятая» используется для связки нескольких выражений. Она вызывает выполнение последовательности действий. Левая сторона операции «запятая» всегда вычисляется как void (то есть не имеет значения). В результате значение операнда, находящегося с правой стороны от знака операции, станет значением разделенного запятыми выражения. Так в выражении  х = (у=3, у+1);  в первую очередь переменной y присваивается значение 3, а затем значение 4 присваивается переменной х. Скобки необходимы, поскольку операция «запятая» имеет самый низкий приоритет.

Об операции «запятая» следует думать как об обычном слове «и» в нормальном русском языке, когда оно используется в выражении «сделай это, и это, и это».

**Оператор преобразования типа**

Для преобразования типа из некоторого класса в какой-либо другой тип должна использоваться перегруженная операция преобразования типа. Она обязательно принадлежит классу и является операторной функцией без аргументов и возвращаемого значения (не возвращает даже void).

Оператор преобразования типа реализует создание объекта типа данных, указанном в его названии, на основании объекта класса, в котором он определен. Каким образом будет выполняться преобразование задает код тела оператора.

Синтаксис определения операции преобразования имеет вид

***operator <тип данных>() {<тело операции>}***

Преобразование типа необходимо, когда требуется по объекту пользовательского типа получить объект стандартного типа (например из class clname - int).

Помимо приведения пользовательского типа данных (класса) к стандартному, возможно реализовать приведение объекта одного класса к объекту другого класса. Для этого необходимо:

1. В каждом из классов объявить конструктор, аргументом которого является ссылка на объект или значение объекта другого класса.
2. В каждом классе определить оператор преобразования типа, имя которого является названием того класса, к объекту которого приводится объект искомого класса.

**Разбор программы**

Разработать ООП для объединения 2-х любых подмножеств латинских букв, которые передаются ей через аргументы командной строки. Полученный результат должен отображаться строкой стандартного вывода, где все буквы перечислены в алфавитном порядке. Программная реализация вычислений должна быть основана на разработке класса множества латинских букв. При этом состав множества кодируется двоичными разрядами целого числа без знака в приватном поле данных этого класса, а публичный компонентный метод предназначается для перегрузки оператора ‘**+’** в операции объединения 2-х множеств. Конструктор класса должен обеспечивать преобразование символьной записи множества в двоичный целочисленный формат. Необходимо также предусмотреть дружественную перегрузку оператора ‘**<<’** потока стандартного вывода или компонентный оператор преобразования типа множества в символьную строку для отображения объектов класса множества латинских букв.

1: #include <cctype>

2: #include <iostream>

3: using namespace std;

4:

5: class Alpha

6: {

7: private:

8: unsigned bin;

9: public:

10: Alpha() {bin=0;};

11: Alpha(char\*);

12: Alpha operator +(Alpha&);

13: operator char\*();

14: friend ostream& operator << (ostream&, Alpha&);

15: };

16:

17: Alpha::Alpha(char\* s)

18: {

19: bin=0;

20: while(\*s)

21: {

22: bin |= (1 << (tolower(\*s) - 'a'));

23: s++;

24: }

25: }

26:

27: Alpha Alpha::operator + (Alpha& y)

28: {

29: Alpha z;

30: z.bin = bin | y.bin;

31: return(z);

32: }

33:

34: ostream& operator << (ostream& out, Alpha& z)

35: {

36: unsigned bit=1;

37: int i;

38: for(i=0; i<26; i++)

39: {

40: if((z.bin & bit)>0)

41: out<<(char)('a'+i);

42: bit=bit<<1;

43: }

44: return out;

45: }

46:

47: Alpha::operator char\*()

48: {

49: static char s[32];

50: unsigned w=bin;

51: int i=0, j=0;

52: while(w > 0)

53: {

54: if(w & 1)

55: s[j++]='a'+i;

56: i++;

57: w = w >> 1;

58: }

59: s[j++]='\n';

60: s[j]='\0';

61: return (s);

62: }

63:

64: int main (int argc, char\* argv[])

65: {

66: Alpha x(argv[1]);

67: Alpha y(argv[2]);

68: Alpha z;

69: z=x+y;

70: cout<<z<<endl; // или cout<<(char\*)z;

71: return (0);

72: }