Лекция 3. Перегрузка операций

**Перегрузка операций** - это средство языка, позволяющее вводить операции для работы с пользовательскими типами.

*Перегрузка операций - 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.*

тип\_возвращаемого\_значения

operator знак\_операции(список\_параметров)

{

тело\_функции

}

На этапе синтаксического анализа компилятор по значению символа знака операции выделяет типы и имена операндов для выполнения соответствующей операции.

Функцию для перегрузки можно определить тремя способами, при которых она должна быть:

1) Методом класса

2) Дружественной функцией класса

3) Обычной функцией, если имеется хотя бы один аргумент с типом класса или являющийся указателем или ссылкой на класс

При перегрузке операций

1. Количество аргументов, приоритеты и правила ассоциации операций аналогичны операциям со стандартными типами данных.

2. Нельзя переопределить операции для стандартных типов данных.

3. Нельзя использовать аргументы по умолчанию.

4. Функция операций (за исключением операций присваивания) наследуются.

5. Нельзя вводить новые операций

6. Можно перегружать любые операции, за исключением:

.

.\*

?:

::

#

##

sizeof

**Перегрузка унарных операций**

Пример. Написать программу с перегруженным оператором "инкремент".

#include <iostream>

using namespace std;

class Per

{

double s;

int p;

public:

Per():s(1), p(10)

{}

void get\_sp()

{

cout<<"s="<<s<<"p="<<p<<endl;

}

void operator++()

{

s++, p++;

}

};

void main()

{

Per x,y;

cout<<"x:"; x.get\_sp();//x: s=1 p=10

cout<<"y:"; y.get\_sp();//y: s=1 p=10

++x; ++y; ++y;

cout<<"x:"; x.get\_sp();//x: s=2 p=11

cout<<"y:"; y.get\_sp();//y: s=3 p=12

}

**Перегрузка бинарных операций**

Пример. Написать класс, позволяющий складывать нуль-терминальные строки.

#include <iostream>

#include <string.h>

using namespace std;

class CStr

{

char st[20];

public:

CStr()

{strcpy(st,"");}

CStr(char s[])

{strcpy(st,s);}

void out\_st()

{cout<<st<<endl;}

CStr operator+(CStr st2)

{

CStr tmp;

strcpy(tmp.st, st);

strcat(tmp.st, st2.st);

return tmp;

}

void operator+=(CStr st2)

{

strcat(st, st2, st);

}

};

int main()

{

CStr sr;

CStr s1="01";

CStr s2="23";

CStr s3="45";

sr=s1+s2;

sr.out\_st();//Выводит 0123

sr=s1+s2+s3;

sr.out\_st();//Выводит 012345

s1+="00";

s1.out\_st();//Выводит 0100

return 0;

}

При вычислении строки "sr=s1+s2;" оператор работает следующим образом: если аргументы в арифметическом выражении имеют тип class, то левый аргумент выражения s1 вызывает метод operator+. Метод имеет прямой доступ к полям левого аргумента. Правый объект передаётся в метод в качестве аргумента функции. Результат передаётся в место вызова метода operator+.

Пример. Написать класс, содержащий оператор для сравнения нуль-терминальных строк.

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

class CStr

{

char st[20];//private необязательно

public:

CStr() {strcpy(st," ");}

CStr(char s[]) {strcpy(st, s);}

bool operator>=(CStr st2)

{

int k=strcmp(st,st2.st);

if(k>=0) return true;

else return false;

}

};//Здесь можно объявить объекты класса

int main()

{

CStr s1="table";

CStr s1="book";

if(st>=s2) cout<<"Str1>=Str2";

else cout<<"Str1<Str2";

return 0;

}

Пример. Написать класс, использующий перегруженный оператор вложений, описанный вне класса. Получить строку путём сложения строки и числа.

#include <iostream>

#include <string.h>

using namespace std;

const int n=20;

class Cls01

{

int k;

public:

Cls01(int m):k(m) {}

int rd()

{

return k;

}

};

class Cls02

{

char st[n];

public:

Cls02(char s[])

{strcpy(st, s);}

char\* rd()

{

return st;

}

};

char\* operator+(Cls01 a, Cls02 b)

{

char s[n];

itoa(a.rd(), s, 10);//Переводит число в строку

strcat(s, b.rd());

return s;//Возвращаем указатель на строку

}

char\* operator+(Cls02 b, Cls01 a)

{

char s1[n], s2[n];

strcpy(s1, b.rd());

itoa(a.rd(), s2, 10);//Переводит число в строку

strcat(s1, s2);

return s1;

}

int main()

{

Cls01 x(5);

Cls02 y("abc");

char \*str=new char [n];

str=x+y; cout<<str<<endl;//"5abc"

str=y+x; cout<<str<<endl;//"abc5"

delete []str;

return 0;

}

Функции, описанные вне класса, имеют доступ только к открытым элементам класса, поэтому при описании класса необходимо объявить метод, который позволяет получить доступ к полям класса.

**Перегрузка операции индексации**

Пример. Написать программу для подсчёта суммы элементов, используя переопределение операции(), и контролирующую выход данных за пределы массива, используя переопределение операции[].

#include <iostream>

#include <string.h>

using namespace std;

const int n=5;

int s=0;

class Cmas

{

int a[n];

public:

int &operator[] (int k)

{

if(k>=n||k<0){

cout<<endl<<"Going beyond the array!"<<endl;

exit(1);

}

return a[k];}

int &operator() (int k)

{

int sum=0;

for (int i=0; i<k; i++) sum+=a[i];

return sum;

}

};

int main()

{

CMas mas;

for(int i=0; i<n; i++) mas[i]=i\*10;

cout<<"Sum of elements ="<<mas(n)<<endl;

cout<<"ARRAY: ";

int i=0;

while(true) cout<<mas[i++}<<" ";

}

Т. к. операция может располагаться слева от операции присваивания, то перегруженный метод возвращает значение по ссылке.

**Преобразование типов**

operator имя\_нового\_типа () {};

Тип возвращаемого значения и параметры указывать не требуется.

Пример.

Cls03::operator int() {return x;}

…

Cls03 a;

…

cout<<int(a);

Пример. Написать программу, использующую преобразование типов.

const int n=20;

class PStr

{

char str[n];

public:

PStr(){

str[0]='/0';

}

PStr(char s[]){

strcpy(str, s);

}

operator int(){

return atoi(str);

}

friend ostream &operator<<(ostream&, const PStr&);//Функция возвращает указатель на поток

};

ostream &operator<<(ostream &os, const PStr&s)

{

os<<s.st<<endl;

return os;

}

int main()

{

PStr st1="Slovo";

cout<<st1;

PStr st2="12";

int sum=10+st2;

cout<<sum<<endl;

return 0;

}

Операция помещения в поток является бинарной операцией (имеет 2 операнда: данные и поток). Если определить функцию operator внутри класса, то у функции фактически будет не 2, а 3 параметра: this и 2 объявленных параметра, что вызывает ошибку.

Оператор можно перегрузить:

1. Функцией членов класса с одним явным параметром (единственный явный параметр будет выступать в качестве второго операнда, а первым операндом будет служить неявный оператор this). В данном случае для переопределения операции помещения в поток этот вариант не подходит, так как первым параметром должен быть указатель на поток. Поэтому операция перегружается за пределами описания класса в виде дружественной функции.

2. Функцией, объявленной вне класса, с двумя явными параметрами.