**лекция 8**

**Тема лекции**: Перегрузка оператора индексирования массива, оператора вызова функции и оператора доступа к члену класса. Перегрузка унарных операторов инкремента и декремента. Правила перегрузки операторов

**Оператор индексирования массива**

 Возможна перегрузка оператора индексирования массива [] для реализации доступа к данным-членам класса, вроде доступа к элементам массива. Эти данные-члены могут иметь вид отдельных членов или списка. Пример перегрузки оператора [] для класса, запоминает четыре целых значения в виде отдельных данных членов.

class PseudoArray

{

private:

int val0, val11, val2, val3;

public:

PseudoArray (int v0, int v1, int v2, int v3)

{

val0 = v0;

val1 = v1;

val2 = v2;

val3 = v3;

}

int GetInt (unsigned i)

int operator [] (unsigned i)

};

main ()

{

PseudoArray pa (10,20,30,40)

for (int i = 0; i <= 3; i ++) cout << pa [i] << '\ n';

return 0;

}

int PseudoArray :: GetInt (unsigned i)

{

switch (i)

{

    case 0: return val0;

    case 1: return val1;

    case 2: return val2;

    case 3: return val3;

    default: return val0;

}

}

int PseudoArray :: operator [] (unsigned i)

{

return GetInt (i)

}

Перегрузка оператора индексирования позволяет использовать в массиве индексацию вместо вызова функции-члена для объекта класса. В этом примере вместо

cout << pa [i] << '\ n';

можно было бы написать:

cout << pa.GetInt (i)

Выражение pa [i] обеспечивает доступ к массиву, несмотря на то, что pa в действительности ссылается на объект класса.

Индексы перегруженных массивов могут быть произвольными типами данных. Например, можно использовать функцию:

int operator [] (char c)

и затем оперировать символьными индексами для индексации массива с начальным индексом "A". В обычных массивах С и С ++ используют целые индексные значения, начинающиеся с нуля. Перегружая [] и используя символы в качестве индексов, мы устанавливаем соответствие между множеством символов и множеством значений, создавая ассоциативный массив.

Возможна такая реализация функции-члена перегрузки оператора []:

int PseudoArray :: operator [] (char c)

{

return GetInt (c-'A ');

}

После этого можно использовать символьную переменную в качестве индекса массива:

for (char c = 'A'; c <= 'D'; c ++)

cout << pa [c] << '\ n';

**Оператор вызова функции**

Перегрузка оператора вызова функции operator () делает объект класса похожим на функцию, которую можно вызвать. Перегружена функция () может возвращать значение заданных типов или совсем ничего не возвращать. В ней также могут объявляться параметры. Она не может быть статичным членом класса.

Пример: класс с перегруженным оператором вызова функции, возвращающей целое значение:

class TAnyClass

{

int x;

public:

int operator () (void);

TAnyClass (int n)

{

x = n;

}

};

int TAnyClass :: operator () (void)

{

return x;

}

main ()

{

TAnyClass object (100)

int g = objеct ();

cout << g;

return 0;

}

В классе TAnyClass перегружается оператор вызова функции с помощью объявления

int operator () (void);

Можно заменить void списком параметров. Эта функция возвращает целое значение члена х объекта TAnyClass. В программе оператор

int g = object ();

похож на вызов функции с именем object (), но на самом деле выполняется оператор:

object.operator () ();

**Оператор доступа к члену класса**

Унарный оператор доступа к члену класса перегружается как operator> (). Он не может быть статичной функцией-членом класса.

class TAnyClass

{

int x, y;

public:

TAnyClass (int xx, int yy)

{

x = xx;

y = yy;

}

TAnyClass \* operator -> ();

int GetХ (void)

{

return x;

}

int GetУ (void)

{

return y;

}

};

TAnyClass \* TAnyClass :: operator -> ()

{

cout << "Доступный член:";

return this;

}

main ()

{

TAnyClass t (100,200)

cout << t-> GetХ () << '\ n';

cout << t-> GetУ () << '\ n';

return 0;

}

Функция-член operator> () должна возвращать объект, посылку или указатель на объект класса. В данном случае она возвращает указатель на класс TAnyClass. В этом примере используется перегрузка оператора> для отладки программы.

Перегружен оператор выводит сообщение перед тем, как вернуть указатель this, ссылающийся на объект, для которого была вызвана функция-член. В программе в двух операторах вывода в поток используется перегруженный оператор> для доступа к функциям членов GetХ () и GetУ () объекта t класса TAnyClass. На самом деле эти операторы выполняются так:

cout << (t.operator -> ()) -> GetХ () << '\ n';

out << (t.operator -> ()) -> GetУ () << '\ n';

Программа отобразит пометки перед значениями, возвращаемые функциями GetХ () и GetУ ():

Доступен член 100

Доступен член 200

**Операторы инкремента и декремента**

Пример перегрузки операторов инкремента и декремента:

class TAnyClass

{

int x;

public:

TAnyClass (int xx)

{

x = xx;

}

int operator ++ ()

{

return ++ x;

}

int operator ++ (int)

{

return x ++;

}

int GetХ (void)

{

return x;

}

};

Функция-член operator ++ () определяет префиксный оператор инкремента для объекта типа TAnyClass. Эта функция не имеет параметров. Функция-член operator ++ (int) определяет постфиксный оператор инкремента для объекта типа TAnyClass. С ++ присваивает 0 едином целом параметру. Для объекта v типа TAnyClass выражение ++ v вызовет перегружен префиксный оператор ++. На самом деле выполняется оператор

v.operator ++ ().

Выражение v ++ вызывает перегружен постфиксный оператор v ++ выполняется как

v.operator ++ (int)

Операторы декремента работают аналогично.

**Правила перегрузки операторов**

• Необходимо обеспечить обдуманные функции перегрузки операторов, так как С ++ не понимает семантики перегруженного оператора.

• С ++ не может выводить сложные операторы из простых. В классе, где объявлены функции перегрузки операторов operator \* () и operator = (), С ++ не сможет вычислить выражение a \* = b, пока не будет перегружен оператор operator \* = ().

• Нельзя изменить синтаксис перегруженных операторов. Бинарные операторы должны оставаться бинарными, унарные - унарными. Например, можно создать унарное деления, так как такой встроенной возможности в С ++ не существует.

• Нельзя изобретать новые операторы.

• Нельзя перегружать символы препроцессора # и ##.