# лекция 7

**Тема лекции**: Дружный функции и дружественные классы. Неполное объявление класса. Взаимно-дружные классы. Перегрузка операторов. Бинарные и унарные операции. Функциональная семантика. преобразование типов

**Дружественных класса и дружественные функции**

В С++ можно обойти правила инкапсуляции с помощью друзей.

Классы в С++ позволяют объявить два вида друзей: дружественные функции и дружественные классы.

Если один класс объявленные дружественным иному классу, то он получает доступ ко всем его закрытых и защищенных членов. Такие классы не связаны родственными отношениями. Для того, чтобы сделать класс дружным иному классу, нужно отметить ключевое слово friend в классе, к которому необходимо получить доступ в другом классе.

//Объявим класс А:

class A

{

private:

double value;

public:

a (){value = 1.7;}

};

class B

{

private:

A anObject;

public:

void ShowValue (void)

{cout << anObject.value;}

};

Член класса А anObject закрыт в классе В. Функция-член ShowValue () попытается отразить value с anObject. Но это невозможно, поскольку value закрыт в классе А. Сделаем класс В дружественный класс А и проблема будет решена. Для этого добавим к объявлению класса А строка:

friend class B;

**Правила использования дружественных классов:**

• Друзьями можно сказать любое число классов.

• Ключевое слово friend должно быть внутри объявления класса.

• В классе должны быть перечислены все его друзья.

• Класс никогда не может объявить сам себя другом другого класса.

• Дружественный класс может объявляться до или после класса, которому он друг.

Обычно дружественных класса объявляются последними в встроенные функции класса могли обращаться к закрытым или защищенным частей другого класса.

Производные от дружественных класс наследуют специального доступа к закрытым и защищенным членам базового класса.

Два класса могут объявить себя дружественными:

class BClass; // Неполное объявление класса.

class AClass

{

friend class BClass;

};

class BClass

{

friend class AClass;

};

Все функции-члены в каждом из классов теперь имеют доступ к закрытым и защищенным членам объектов другого класса. Если в первом классе есть ссылки с именем на второй класс, то может потребоваться предварительное неполное объявление класса. В этом случае такое объявление сделано для класса BClass. Это позволяет в AClass объявить данные-члены и параметры функций-членов типа BClass несмотря на то, что полное объявление BClass следует позже.

**Дружественные функции**

Дружественные функции могут быть обычными функциями С ++ или членами класса. Обычно дружественная функция объявляется с параметрами-классами, с которыми она дружит. Внутри дружественной функции операторы имеют доступ к скрытым членов объекта класса, передаваемого аргументом функции. К примеру:

#include <iostream.h>

class Two; // Неполное ооголошення класса

class One

{

friend void Show (0ne & c1, Two & c2)

private:

char \* s1; // Доступен класса One и функции Show ()

public:

Оne ()

{

 s1 = "Testing";

}

};

class Two

{

friend void Show (Оne & c1, Two & c2)

private:

char \* s2; // Доступный класса Two и функции Show ()

public:

Two ()

{

s2 = "one, two, three";

}

};

main ()

{

One c1;

Two c2;

Show (c1, c2)

return 0;

}

void Show (One & c1, Two & c2)

{

cout << c1.s1 << c2.s2 << '\ n';

}

Здесь объявлены два класса One и Two. Неполное объявление класса Two позволяет ссылаться в One на Two. Функция Show () объявлена ​​дружественной в каждом классе. Она имеет 2 параметра C1 и C2 двух типов классов, на которые ссылается. Операторы внутри Show () имеют доступ к защищенным и закрытым членам объектов классов-аргументов, переданных функции.

Дружеская функция также может быть членом класса. Обычно в классе объявляется дружественной функция-член другого класса. Имеет доступ к закрытой и защищенной части класса, в котором она объявлена ​​дружественной. Класс, в котором содержится прототип функции-члена, должен объявляться до класса, указывает функцию-член дружественной.

**Перегрузка операторов**

1. Перегрузка операторов позволяет определять действия для объектов классов в выражениях, использующих обычные операторы. Объявим класс TAnyClass, а затем - несколько объектов этого класса:

TAnyClass C1, C2, C3;

Переопределив оператор +, мы получим возможность использовать эти объекты в выражениях вида:

C3 = C1 + C2;

1. Перегрузка операторов дает возможность добавлять к встроенным типам данных новые типы.
2. Перегрузка операторов объявляется так же, как и обычная дружеская функция или функция-член класса:

class ZZ

{

public:

friend ZZ operator + (ZZ a, ZZ b)

};

Для перегрузки операторов имя функции должно состоять из слова operator и символа бинарной или унарный операции. Операторы. , ::, Sizeof,. \* Не могут перегружаться.

В объявлении

friend ZZ operator + (ZZ a, ZZ b)

функция operator + () объявляется дружественной класса, то есть она получает доступ к закрытым и защищенным членам этого класса. Функция возвращает объекты типа ZZ и этот результат можно присвоить другому объекту ZZ. Имя функции operator + () идентифицирует функцию как метод, с помощью которого выражения, которые используют оператор сложения, могут оперировать объектами класса. Предполагается, что функция operator + () подытоживает два объекта средством, подходит для класса ZZ.

Выражения a + b и operator + (a, b) выполняют одинаковую работу и генерируют одинаковый код.

Перегружены операторы могут быть членами класса. Определим класс, способный запоминать целые значения в виде строк. С помощью перегруженных операторов программа может вычислять выражения, в которых добавляются строки в виде эквивалентных им числовых значений.

class TStr

{

private:

char val[12];

public:

TStr () { val [0] = 0; }

TStr (const char \* s);

long GetVal (void) {return atol(val) }

friend long operator + (TStr a, TStr b)

};

main ()

{

TStr a ( «1234»);

TStr b ( "4321");

cout << a.GetVal ();

cout << b.GetVal ();

cout << a + b + 6;

return 0;

}

TStr :: TStr (const char \* s)

{

strncpy (val, s, 11);

val [12] = 0;

}

long operator + (TStr a, TStr b)

{

return (atol (a.val) + atol (b.val))

}

Здесь дружественная функция перегружает оператор сложения пары объектов класса TStr. Функция возвращает длинное целое, хотя обычно функции перегрузки операторов возвращают тип класса (или посылки на объект класса), объектами которого они оперируют.

В случае, если перегружены функции - члены класса, листинг изменится. В объявлении класса - строка:

friend long operator + (TStr a, TStr b)

изменится на следующую строку:

long operator + (TStr b)

И реализация этой функции будет выглядеть так:

long TStr :: operator + (TStr b)

{

return (atol (val) + atol (b.val))

}

Эта функция-член получает указатель this, ссылающийся на объект, для которого она вызвана. Итак функции необходим один параметр, а не два. Для добавления двух строчных значений operator + () добавляет длинные целые значения, эквивалентные выражениям: this> val и b.val. В виде

long operator + (TStr a, TStr b)

оператор не скомпилируется, так как функция operator + () является членом класса TStr и как функция-член получает указатель this, ссылающийся на объект класса. Таким образом, функция имеет три параметра: два явных и один скрытый, а перегруженном оператору необходимы только два.

**Бинарные и унарные операции**

Бинарная операция может быть определенной или как функция-член, получает один параметр, или как глобальная функция, получает 2 параметра. То есть, для любой бинарной операции @ выражение aa @ bb может интепретуватися как aa.operator @ (bb) или как operator @ (aa, bb).

Унарная операция, префиксная или Постфиксная, может быть определенной или как функция-член, не получает параметров, или как глобальная функция, получает один параметр. Для любой префиксной унарными операции @ выражение @aa может интепретуватися как aa.operator @ () или как operator @ (aa).

Для любой Постфиксная унарными операции aa @ - aa.operator @ (int) или operator @ (aa, int).

Фиктивный параметр int никогда не используется, а только служит признаком того, что функция вызывается для выполнения операций в Постфиксная варианте. примеры:

class x

{

x \* operator & (); // Унарное & (адрес)

x operator & (x) // Бинарное & (и)

x operator - (x) // Префиксный унарный минус

x operator - (x, x); // Бинарный минус

};

**Преобразование типов**

Можно определять свои собственные правила преобразования типов данных для автоматического преобразования объектов на другие типы. Например, пусть объявлено и инициализирован объект класса TStr:

TStr myVal ( "9876")

Класс TStr запоминает строку, которая может использоваться как эквивалент долгого целого значения в операции сложения. оператор

long x = myVal;

НЕ скомпилируется, потому что в классе TStr не предусмотрено преобразование строки в длинное целое значение. Нужно перегрузить операцию преобразования типов для преобразования объекта класса TStr на другой тип данных:

long operator long ()

{

return atol (val)

}

Эти строки надо вставить в открытой секции класса TStr. Подобное использование ключевого слова operator специально обеспечивается в C ++ для создания новых правил преобразования типа. Для преобразования класса А на класс В, нужно поместить такую ​​функцию-член к классу А:

operator B ()

{

...

}