Лекція 7

Тема лекції: Дружні функції та дружні класи. Неповне оголошення класу. Взаємодружні класи. Перевантаження операторів. Бінарні та унарні операції. Функціональна семантика. Перетворення типів

Дружні класи та дружні функції

В C++ можна обминути правила інкапсуляції за допомогою друзів. Класи в C++ дозволяють оголосити два види друзів: дружні функції і дружні класи.

Якщо один клас оголошеийі дружнім іншому класу, то він одержує доступ до всіх його закритих і захищених членів. Такі класи не пов'язані родинними стосунками. Для того, щоб зробити клас дружнім іншому класу, потрібно зазначити ключове слово *friend* у класі, до якого необхідно одержати доступ в іншому класі.

Оголосимо клас *A*:

```
class A
{
 private:
  double value;
 public:
  a()
   {
    value = 1.7;
   }
};
class B
{
 private:
  A anObject;
 public:
  void ShowValue(void)
  {
   cout<<anObject.value;
```

```
  }
};
```

Член класу *A anObject* закритий у класі *B. Ф*ункція-член *ShowValue()* спробує відобразити *value* з *anObject*. Але це неможливо, оскільки *value* закритий у класі *A*. Зробимо клас *B* дружнім класу *A* й проблема буде вирішена. Для цього додамо до оголошення класу *A* рядок:

```
friend class B;
```

Правила використання дружніх класів:

- Друзями можна оголосити будь-яке число класів.
- Ключове слово *friend* повинно бути всередині оголошення класу.
- У класі повинні бути перераховані усі його друзі.
- Клас ніколи не може оголосити сам себе другом іншого класу.
- Дружній клас може оголошуватись до або після класу, якому він друг.

Зазвичай дружні класи оголошуються останніми, щоб вбудовані функції класу могли звертатися до закритих або захищених частин іншого класу.

Похідні від дружніх класів не успадковують спеціального доступу до закритих і захищених членів базового класу.

Два класи можуть оголосити себе дружніми:

```
class BClass; //неповне оголошення класу.
class AClass
{
 friend class BClass;
};
class BClass
{
 friend class AClass;
};
```

Всі функції-члени в кожному з класів тепер мають доступ до закритих і захищених членів об'єктів іншого класу. Якщо в першому класі є посилання за ім'ям на другий клас, то може знадобитися попереднє неповне оголошення класу. У цьому випадку таке оголошення зроблене для класу *BClass*. Це дозволяє в *AClass* оголосити дані-члени і параметри функцій-членів типу *BClass* незважаючи на те, що повне оголошення *BClass* випливає пізніше.

Дружні функції можуть бути звичайними функціями C++ або членами класу. Зазвичай дружня функція оголошується з параметрами-класами, з якими вона дружить. Всередині дружньої функції оператори мають доступ до прихованих членів об’єкту класу, переданого аргументом функції. Наприклад:

```
#include <iostream.h>
class Two;// Неповне ооголошення класу
class One
{
 friend void Show(0ne &c1, Two &c2);
  private:
   char *s1;   // Доступен классу One и функции Show()
  public:
   One()
    {
     s1 = "Testing ";
    }
};
class Two
{
friend void Show(One &c1, Two &c2);
 private:
  char *s2;    // Доступний класу Two та функції 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';
}
```

Тут оголошені два класи *One* і *Two*. Неповне оголошення класу *Two* дозволяє посилатися в *One* на *Two*. Функція *Show()* оголошена дружньою в кожному класі. Вона має 2 параметри *C1* і *C2* двох типів класів, на які посилається. Оператори всередині *Show()* мають доступ до захищених і закритих членів об'єктів класів-аргументів, переданих функції.

Дружня функція також може бути членом класу. Зазвичай в класі оголошується дружньою функція-член іншого класу. Вона має доступ до закритої і захищеної частини класу, в якому вона оголошена дружньою. Клас, в якому утримується прототип функції-члена, повинен оголошуватись до класу, що вказує функцію-член дружньою.

Перевантаження операторів

Перевантаження операторів дозволяє визначати дії для об'єктів класів у виразах, що використовують звичайні оператори. Оголосимо клас *TAnyClass*, а потім - декілька об'єктів цього класу:

```
TAnyClass C1, C2, C3;
```

Перевизначивши оператор +, ми одержимо можливість використовувати ці об'єкти у виразах вигляду:

```
C3=C1+C2;
```

Перевантаження операторів дає можливість додавати до вбудованих типів даних нові типи. Перевантаження операторів оголошуєтьсяється так само, як і звичайна дружня функція або функція-член класу:

```
class ZZ
{
 public:
  friend ZZ operator+(ZZ a, ZZ b);
};
```

Для перевантаження операторів ім'я функції повинно складатися зі слова *operator* і символу бінарної або унарної операції. Оператори . , :: , *sizeof*, .* не

можуть перевантажуватися.

В оголошенні

```
friend ZZ operator+(ZZ a, ZZ b);
```

функція *operator+()* оголошується дружньою класу, тобто вона одержує доступ до закритих і захищених членів цього класу. Функція повертає об'єкти типу *ZZ* і цей результат можна привласнити іншому об'єкту *ZZ*. Ім'я функції *operator+()* ідентифікує функцію як метод, за допомогою якого вирази, що використовують оператор додавання, можуть оперувати об’єктами класу. Передбачається, що функція *operator+()* підсумовує два об'єкти засобом, що підходить для класу *ZZ*.

Вирази *a+b* та *operator+(a, b)* виконують однакову роботу і генерують однаковий код.

Перевантажені оператори можуть бути членами класу. Визначимо клас, спроможний запам'ятовувати цілі значення у вигляді рядків. За допомогою перевантажених операторів програма може обчислювати вирази, в яких додаються рядки у вигляді еквівалентних їм числових значень.

```
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[11]=0;
}
long operator+(TStr a, TStr b)
{
 return(atol(a.val)+atol(b.val));
}
```

Тут дружня функція перевантажує оператор додавання пари об'єктів класу TStr. Функція повертає довге ціле, хоча зазвичай функції перевантаження операторів повертають тип класу (або посилки на об'єкт класу), об’єктами якого вони оперують.

У випадку, якщо перевантажені функції - члени класу, лістинг зміниться. В оголошенні класу - рядок:

```
 friend long operator+(TStr a, TStr b);
```

зміниться на наступний рядок:

```
 long operator+(TStr b);
```

І реалізація цієї функції буде виглядати так:

```
long TStr::operator+(TStr b)
 {
  return(atol(val)+atol(b.val));
 }
```

Ця функція-член одержує покажчик *this*, що посилається на об'єкт, для якого вона викликана. Отже функції необхідний один параметр, а не два. Для додавання двох рядкових значень *operator+()* додає довгі цілі значення, еквівалентні виразам: *this -> val* та *b.val*. У вигляді

```
long operator+(TStr a,TStr b);
```

оператор не скомпілюється, тому що функція *operator+()* є членом класу *TStr* і як функція-член одержує покажчик *this*, що посилається на об'єкт класу. Таким чином, функція має три параметри: два явних та один прихований, а перевантаженому оператору необхідні тільки два.

Бінарні та унарні операції

Бінарна операція може бути визначеною або як функція-член, що одержує один параметр, або як глобальна функція, що одержує 2 параметри. Тобто, для будь-якої бінарної операції @ вираз aa@bb може інтепретуватися як *aa.operator@(bb)* або як *operator@(aa,bb).*

Унарна операція, префіксна або постфіксна, може бути визначеною або як функція-член, що не одержує параметрів, або як глобальна функція, що одержує один параметр. Для будь-якої префіксної унарної операції @ вираз *@aa* може інтепретуватися як *aa.operator@()* або як *operator@(aa).*

Для будь-якої постфіксної унарної операції *aa@* - *aa.operator@(int)* або *operator@(aa,int).*

Фіктивний параметр *int* ніколи не використовується, а лише служить ознакою того, що функція викликається для виконання операцій у постфіксному варіанті. Приклади:

```
class x
{
  x* operator&(); //унарне &(адреса)
  x operator&(x); //Бінарне &(і)
  x operator -(x); //Префіксний унарний мінус
  x operator -(x,x);//Бінарний мінус
};
```

Перетворення типів

Можна визначати свої власні правила перетворення типів даних для автоматичного перетворення об'єктів на інші типи. Наприклад, нехай

оголошено та ініціалізовано об'єкт класу *TStr*:

```
TStr myVal("9876");
```

Клас *TStr* запам'ятовує рядок, що може використовуватися як еквівалент довгого цілого значення в операції додавання. Оператор

```
long x = myVal;
```

не скомпілюється, тому що в класі *TStr* не передбачене перетворення рядка на довге ціле значення. Потрібно перевантажити операцію перетворення типів для перетворення об’єкту класу *TStr* на інший тип даних:

```
long operator long()
{
  return atol(val);
}
```

Ці рядки треба вставити до відкритої секції класу *TStr*. Подібне використання ключового слова *operator* спеціально забезпечується в C++ для створення нових правил перетворення типу. Для перетворення класу *A* на клас *B*, потрібно помістити таку функцію-член до класу *A*:

```
operator B()
{
...
}
```