**Лекція 14 Дружні функції та класи**

**Дружні friend функції**

Іноді бажано мати безпосередній доступ ззовні до прихованих полів класу, тобто розширити інтерфейс класу. Для цього служать дружні функції і дружні класи. С++ надає можливість обійти одну із концепцій ООП – концепцію інкапсуляції - за допомогою друзів. С++ дозволяє об’являти два види друзів класу: дружню функцію та дружній клас.

Звичайний доступ до закритих членів класу – використання відкритої функції-члена. Але С++ підтримує інший спосіб одержання доступу до закритих членів класу – за допомогою дружніх функцій. Дружні функції не є членами класу, але тим не менше мають доступ до його закритих членів. Одна така функція може мати доступ до закритих членів декількох класів.

Функція не може стати другом класу “без його згоди”. Для одержання прав друга функція повинна бути описана в тілі класу.

Щоб оголосити функцію дружньою деякому класу, у визначення цього класу включають її прототип, перед яким ставиться ключове слово friend.

Дружні функції застосовуються для доступу до прихованих членів класу і є альтернативою методам. Метод, як правило, використовується для реалізації властивостей об'єкту, а у вигляді дружніх функцій оформляються дії, що не представляють властивості класу, але що концептуально входять в його інтерфейс і потребують доступу до його прихованих полів.

***Правила опису і особливості дружніх функцій***.

1. Дружня функція оголошується *усередині класу,* до елементів якого їй потрібен доступ, з ключовим словом **friend**. Як параметр їй повинні передаватися об'єкт або посилання на об'єкт класу, оскільки вказівка **this** їй не передається. Тому, викликаючи дружню функцію, не потрібно вказувати ім’я об’єкта або вказівник на об’єкт і операцію доступу до члена класу (крапку або стрілку).
2. Дружня функція може бути звичайною функцією або методом іншого раніше створеного класу. На неї не розповсюджується дія специфікаторів доступу, місце розміщення її оголошення в класі байдуже.
3. Одна функція може бути дружньою відразу з декількома класами.
4. Обмеження на кількість дружніх функцій відсутні.
5. Дружня функція не успадковується, тобто вона не є такою для похідних класів.
6. Відношення дружності не є транзитивним: Х дружній Y, Y дружній Z, але не обов‘язково Х дружній Z.

***Загальна форма оголошення “дружньої” функції до деякого класу***

Оголошення “дружньої” функції до класу починається з ключового слова friend. Загальна форма оголошення “дружньої” функції до класу має вигляд:

**friend** *type* **FunName**(*parameters*);

де

* *FunName* – ім’я “дружньої” функції;
* *type* – тип, що повертається функцією FunName();
* *parameters* – параметри “дружньої” функції. Щоб отримати об’єкт потрібного класу у функції FunName() доцільно передати у цю функцію посилання (або покажчик) на об’єкт цього класу.

Якщо потрібно оголосити “дружню” функцію у деякому класі, то загальний вигляд такого оголошення наступний:

**class CClass**

**{**

**// ...**

**friend *type* FunName(*parameters*);**

**// ...**

**};**

де

* *FunName* – ім’я “дружньої” функції;
* *type* – тип, що повертається функцією FunName();
* *parameters* – параметри “дружньої” функції.

**Приклад :**

**class Person**

**{**

int secret; // закрита частина класу

friend void Spouse(Person &); // не член класу

public:

Person(){ secret = 10;}

**}**;

# Void Spouse(Person &p)

**{**

++p.secret;

cout << p.secret;

**}**

## Void main()

**{**

Person ff;

Spouse(ff);

**}**

У даному прикладі функція **Spouse()** є звичайною функцією і не належить жодному класу. Цій функції передається посилання на об'єкт класу **Person**.

**Специфікатор friend:**

* + Оголошує в класі дружню-зовнішню функцію, що має такий же доступ до членів-даних і функцій (приватних та публічних), як власні методи класу
  + Вона не має доступу до this тому має на 1 параметр більше ніж аналогічний метод
  + Вона реалізовується (без вживання friend) і викликається як зовнішня функція

**Приклад:**

class monstr; // *Попереднє оголошення класу*

**class hero**

**{**

**public:**

void kill(monstr &);

…

**}**

**class monstr**

**{**

…

**friend int steal\_ammo(monstr &);**

**friend void hero:: kill(monstr &);**

// *Клас* hero *має бути визначений раніше*

**};**

…

int steal\_ammo(monstr &M){return --M.ammo;}

void hero::kill(monstr &M){M.health = 0; M.ammo = 0;}

У даному прикладі приведений опис двох функцій, дружніх класу **monstr**. Функція **kill** є методом класу **hero**, а функція **steal\_ammo** не належить жодному класу.

**Приклад** оголошення дружньої функцій для двох класів.

**#include <iostream.h>**

**class myclass1; //неповне оголошення класу**

**class myclass2**

**{**

**int d;**

**public:**

**myclass2 (int d1) { d=d1;}**

**// оголошення дружньої фукції**

**friend bool isfactor(myclass1 ob1, myclass2 ob2);**

**};**

**class myclass1**

**{**

**int n;**

**public:**

**myclass1 ( int n1) {n=n1;}**

**// оголошення дружньої фукції**

**friend bool isfactor(myclass1 ob1, myclass2 ob2);**

**};**

**// визначення дружньої фукції**

**bool isfactor (myclass1 ob1, myclass2 ob2)**

**{**

**if (ob1.n % ob2.d=0) return true;**

**else return false;**

**}**

**void main()**

**{**

**myclass1 ob1(12);**

**myclass2 ob2(3);**

**if (isfactor (ob1, ob2))**

**cout << “ 12 ділиться без остачі на 3 \n”;**

**else**

**cout << “ 12 не ділиться без остачі на 3 \n”;**

**}**

  Ця програма демонструє також важливий випадок застосування неповного оголошення класу: без застосування цієї конструкції в даному випадку було б неможливо оголосити дружню функцію для двох класів. Неповне оголошення класу myclass1 дає можливість використовувати його ім’я в оголошенні дружньої функції ще до його визначення.

Функція може бути членом одного класу і дружньою іншому.

Наприклад.

**#include <iostream.h>**

**class myclass1; //неповне оголошення класу**

**class myclass2**

**{**

**int d;**

**public:**

**myclass2 (int d1) { d=d1;}**

**bool isfactor(myclass1 ob1);**

**};**

**class myclass1**

**{**

**int n;**

**public:**

**myclass1 ( int n1) {n=n1;}**

**// оголошення дружньої фукції**

**friend bool myclass2 :: isfactor(myclass1 ob1);**

**};**

**// визначення дружньої фукції**

**bool myclass2 :: isfactor (myclass1 ob1)**

**{**

**if (ob1.n % d=0) return true;**

**else return false;**

**}**

**void main()**

**{**

**myclass1 ob1(12);**

**myclass2 ob2(3);**

**//isfactor () викликається як функція-член класу myclass2**

**if (ob2.isfactor (ob1))**

**cout << “ 12 ділиться без остачі на 3 \n”;**

**else**

**cout << “ 12 не ділиться без остачі на 3 \n”;**

**}**

  Використання дружніх функцій потрібно по можливості уникати, оскільки вони порушують принцип інкапсуляції і, таким чином, ускладнюють відлагодження та модифікацію програми.

**Friend порушує інкапсуляцію даних !**

[**Правило 23: Предпочитайте функциям-членам функции, не являющиеся ни членами, ни друзьями класса**](http://cpp.com.ru/meyers_1/ch4.html)  (**Скот Маэрс. Эффективное использование С++.**)

## Дружній клас

Бувають випадки, коли для заданого класу потрібно оголосити інший клас або функцію, які повинні мати необмежений доступ до внутрішніх змінних та методів класу. Така необхідність виникає з суті задачі, що розв’язується.

Якщо клас A оголошується “дружнім” до класу B, то об’єкти класу A мають доступ до усіх членів даних і методів класу B. Якщо функція оголошується “дружньою” до деякого класу, то у цій функції також є необмежений доступ до членів даних та методів цього класу.

Щоб оголосити “дружній” клас до даного класу, використовується ключове слово friend. Загальна форма оголошення “дружнього” класу до даного має вигляд:

**class CClass**

**{**

**// ...**

**friend class CFriendClass;**

**// ...**

**};**

**class CFriendClass**

**{**

**// ...**

**};**

де

* CClass – клас, в якому оголошується “дружній” клас CFriendClass. Усі змінні (навіть і private) та методи цього класу є доступними для об’єктів класу CFriendClass;
* CFriendClass – клас, який є “дружнім” до класу CClass. Оголошення “дружнього” класу CFriendClass до класу CClass може бути в будь-якому місці тіла класу – у межах оголошення класу (між фігурними дужками { }).

***Особливості дружніх функцій***.

1. Оголошення **friend** не є специфікатором доступу і не успадковується.
2. Оголошувати “дружній” клас до заданого класу можна у будь-якому місці чи розділі класу в межах його оголошення (між фігурними дужками { } ).
3. Обмеження на кількість дружніх класів відсутні.
4. · Дружність не є взаємною властивістю: якщо А друг В, то не означає, що В – друг А;
5. · Дружність не успадковується: якщо В – друг А, то класи, що походять від В, не є дружніми із А. Це означає, що коли клас оголошений дружнім, всі його методи так само стають дружніми, до того класу в якому він оголошений. При цьому методи класу, який дозволив дружбу не мають доступу до елементів дружнього класу;
6. · Дружність не переходить на нащадки базового класу: якщо В – друг А, то В не є другом для класів, похідних від А.
7. Якщо визначення дружнього класу розташовується нижче визначення класу, надає дружбу, то оголосити дружній клас треба вище. Це допоможе уникнути помилок при компіляції.

**Приклад оголошення класу, що є дружнім до іншого класу**

Нехай задано клас Number, що містить цілочисельну величину. Також задано клас RangeNum, який містить величину Number, але в межах заданого діапазону.

Щоб з класу RangeNum можна було мати доступ до приватної змінну num класу Number, клас RangeNum оголошується “дружнім” до класу Number.

Лістинг класів Number та RangeNum має вигляд:

**// клас, що реалізує ціле число**

**class Number**

**{ // оголошення дружнього класу RangeNum до класу Number**

**friend class RangeNum;**

**int num;**

**public:**

**// конструктори**

**Number() { num = 0; }**

**Number(int num) { this->num = num; }**

**};**

**// оголошення класу RangeNum, який тримає число Number в // заданих межах**

**class RangeNum**

**{**

**Number num; // об'єкт класу Number -просте ціле число**

**int min; // нижня межа числа num**

**int max; // верхня межа числа num**

**public:**

**// конструктор класу**

**RangeNum()**

**{// доступ до private-члена класу Number, тому що**

**// RangeNum є дружнім до Number**

**num.num = 0;**

**// встановлення діапазону 0..99 за замовчуванням**

**min = 0;**

**max = 99;**

**}**

**// методи доступу**

**int GetNum(void)**

**{**

**return num.num; // доступ до приватного члена з**

**// "дружнього" класу Range**

**}**

**void SetNum(int nnum)**

**{**

**num.num = nnum; // доступ до приватного члена з**

**// "дружнього" класу Range**

**if (num.num>max) num.num = max-1;**

**if (num.num<min) num.num = min;**

**}**

**// встановлення діапазону для num в заданих межах**

**void SetRange(int min, int max)**

**{**

**this->min = min;**

**this->max = max;**

**// знову доступ через дружній клас:**

**if (num.num>max) num.num = max-1;**

**if (num.num<min) num.num = min;**

**}**

**};**

Якщо у класі Number в оголошенні “дружнього” класу RangeNum

friend class RangeNum;

забрати ключове слово friend, то у конструкторі та усіх методах класу при доступі до num.num компілятор видасть помилку:

Number::num: cannot access private member declared in class 'Number'

Клас не може оголосити сам себе другом деякого іншого класу. Для того, щоб механізм дружності запрaцював, він повинен бути оголошеним дружнім в цьому іншому класі.

Два класа могуть оголосити один одного другами. З практичної точки зору така ситуація свідчить про погано продуману ієрархію класів, хоча мова С++ допускає таку можливість. В цьому випадку оголошення класів буде мати вигляд:

Class B; //неповне оголошення класу

Class A

{

friend class B;

//…

};

class B

{

friend class A;

//…

};

Неповне оголошення класу, яке наведене в даному фрагменті, може знадобитись, лише якщо в класі А є посилання на клас В, наприклад, в параметрі функції-члена.

Іноді зручніше не оголошувати дружні функції в тілі іншого класу, а оголосити замість них дружній клас. Тоді методи цього дружнього класу, автоматично стануть дружніми класу, який надає дружбу.

Наприклад, якщо дружній клас містить 5 – 10 методів і кожному з них необхідний доступ до приватних елементів іншого класу. Тоді, оголосивши дружній клас, сам код буде виглядати компактніше. Але якщо доступ до елементів іншого класу необхідний тільки кільком методам дружнього класу, краще скористатися оголошенням дружніх функцій. Так наші приватні елементи будуть більш захищені від випадкового внесення в них помилкових даних.

**Класифікація методів**

По відношенню до класу

* Внутрішні (методи)
* Зовнішні
* Дружні

Прив’язування до об’єкту/класу

* Динамічне
* Статичне

**Спеціальні методи**

* Спеціальні методи
* Оператори
* Методи, визначенні програмістом

Сигнатура спеціального методу частково або повністю визначається компілятором

Деякі спеціальні методи генеруються компілятором автоматично, якщо його не визначив програміст. До спеціальних методів відносяться:

* Конструктор з параметрами
* Конструктор по замовчуванню
* Конструктор копії
* Оператор присвоєння
* Оператори виділення і очищення пам‘яті
* Оператори приведення
* Деструктор

**Призначення методів**

* **Породжуючі методи**
* **Ініціалізуючі методи**
* **Завершуючі методи**
* **Методи доступу до властивостей**
* **Перетворючі методи**

***Контрольні запитання*.**

[1. Для чого використовуються так звані “друзі” класу?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/#q01)

[2. Яка загальна форма оголошення “дружнього” класу?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/#q02)

[3. Яка загальна форма оголошення “дружньої” функції до деякого класу?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/#q03)

[4. В якому місці (розділі) оголошення класу можна оголошувати дружній клас або функцію?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/#q04)

[5. Скільки “дружніх” функцій та “дружніх” класів можна оголошувати у тілі класу?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/#q05)

6. [Як отримати об’єкт потрібного класу у “дружній” функції, щоб мати доступ до усіх його членів?](https://www.bestprog.net/uk/2018/07/21/friends-of-the-class-friendly-classes-and-functions-the-keyword-friend-examples_ua/" \l "q06)

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи.

**Рекомендована література**

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
2. Страуструп Б. Язык программирования С++. — СПб.; М.: Невский диалект — ЗАО “Изд-во “Бином”, 1999.
3. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>

6. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ . [2-е изд.] / Буч Г. - СПб.: Невский Диалект, 1998. - 560 с.

7. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997. URL: <http://khizha.dp.ua/library/Timothy_Budd_-_Introduction_to_OOP_(ru).pdf>

8. *Скотт Мейерс.* Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. *"ДМК", 2000; 240 с.*