Friend Functions

The private member data of a class can be accessed only by member functions of that class.

Well, there is one exception. A **friend function** will be friendly with a class even though it is not a member of that class and can access the private members of the class.

#include <iostream>

using namespace std;

class Rectangle {

int width, height;

public:

Rectangle(int w = 1, int h = 1):width(w),height(h){}

friend void display(Rectangle &);

};

void display(Rectangle &r) {

cout << r.width \* r.height << endl;

}

int main () {

Rectangle rect(5,10);

display(rect);

return 0;

}

We make a function a friend to a class by declaring a prototype of this external function within the class, and preceding it with the keyword **friend**

friend void display(Rectangle &);

The friend function **display(rect)** has an access to the private member of the Rectangle class object though it's not a member function. It gets the width and height using dot: **r.width** and **r.height**. If we do this inside main, we get an error because they are private members and we can't access them outside of the class. But friend function to the class can access the private members.

But what's the point of friend functions. In the above example, we could have made "display" as a member function of the class instead of declaring it as a friend function to the class.

Why do we need friend functions?

A friend function can be friendly to 2 or more classes. The friend function does not belong to any class, so it can be used to access private data of two or more classes as in the following example.

#include <iostream>

using namespace std;

class Square; // forward declaration

class Rectangle {

int width, height;

public:

Rectangle(int w = 1, int h = 1):width(w),height(h){}

friend void display(Rectangle &, Square &);

};

class Square {

int side;

public:

Square(int s = 1):side(s){}

friend void display(Rectangle &, Square &);

};

void display(Rectangle &r, Square &s) {

cout << "Rectangle: " << r.width \* r.height << endl;

cout << "Square: " << s.side \* s.side << endl;

}

int main () {

Rectangle rec(5,10);

Square sq(5);

display(rec,sq);

return 0;

}

Output is:

Rectangle: 50

Square: 25

The friend functions can serve, for example, to conduct operations between two different classes. Generally, the use of friend functions is out of an object-oriented programming methodology, so whenever possible it is better to use members of the same class to perform operations with them as in the following example getting exactly same output.

#include <iostream>

using namespace std;

class Rectangle {

int width, height;

public:

Rectangle(int w = 1, int h = 1):width(w),height(h){}

void display() {

cout << "Rectangle: " << width \* height << endl;

};

};

class Square {

int side;

public:

Square(int s = 1):side(s){}

void display() {

cout << "Square: " << side \* side << endl;

};

};

int main () {

Rectangle rec(5,10);

Square sq(5);

rec.display();

sq.display();

return 0;

}

Summary:

1. Friend functions are not members of any class but they can access private data of the class to which they are a friend.
2. Because they are not members of any class, you should not call them using the dot operator.

Friend Classes

Just like functions are made friends of classes, we can also make one class to be a friend of another class. Then, the friend class will have access to all the private members of the other class.

#include <iostream>

using namespace std;

class Square;

class Rectangle {

int width, height;

public:

Rectangle(int w = 1, int h = 1):width(w),height(h){}

void display() {

cout << "Rectangle: " << width \* height << endl;

};

void morph(Square &);

};

class Square {

int side;

public:

Square(int s = 1):side(s){}

void display() {

cout << "Square: " << side \* side << endl;

};

friend class Rectangle;

};

void Rectangle::morph(Square &s) {

width = s.side;

height = s.side;

}

int main () {

Rectangle rec(5,10);

Square sq(5);

cout << "Before:" << endl;

rec.display();

sq.display();

rec.morph(sq);

cout << "\nAfter:" << endl;

rec.display();

sq.display();

return 0;

}

We declared **Rectangle** as a friend of **Square** so that **Rectangle** member functions could have access to the private member, **Square::side**

In our example, **Rectangle** is considered as a friend class by **Square** but **Rectangle** does not consider **Square** to be a friend, so **Rectangle** can access the private members of **Square** but not the other way around.

// Friend class demonstration

#include <iostream>

using namespace std;

class NumberOne;

class NumberTwo{

int num;

friend class NumberOne;

public:

NumberTwo(int k):num(k){}

void display(){ cout<<"Display from NumberTwo - NumberTwo.num = "<<num<<endl; }

};

class NumberOne{

int num;

public:

NumberOne(int k):num(k){}

void displayAndSet(NumberTwo &two){

cout<<"displayAndSet from NumberOne - ";

cout<<"NumberOne.num = "<<this->num<<" "<<"NumberTwo.num = "<<two.num<<endl;

// Changing value of private member of class NumberTwo within friend class NumberOne of class NumberTwo.

cout<<"Changing value of private member of class NumberTwo within friend class NumberOne who is friend of class NumberTwo."<<endl;

two.num = 100;

}

};

int main() {

NumberTwo two(10);

NumberOne one(15);

two.display();

one.displayAndSet(two);

two.display();

cout<<"exit!!!";

}

// Friend function demonstration

#include <iostream>

using namespace std;

class NumberOne{

int num;

public:

NumberOne(){}

NumberOne(int k):num(k){}

void display(){

cout<<"num = "<<num<<endl;

}

friend void friendDisplayNumberOne(NumberOne &one);

//friend void friendSetNumberOne(NumberOne &one);

friend NumberOne friendDuplicateclass(const NumberOne &one, int multiplier);

};

void friendDisplayNumberOne(NumberOne &one)

{

cout<<"friendDisplayNumberOne: num = "<<one.num<<endl;

}

/\*

void friendSetNumberOne(NumberOne &one, int k)

{

cout<<"Setting NumberOne.num"<<endl;

one.num = k;

}\*/

NumberOne friendDuplicateclass(const NumberOne &one, int multiplier){

NumberOne two;

two.num = one.num \* multiplier;

return two;

}

int main() {

NumberOne one(15);

NumberOne two;

one.display();

friendDisplayNumberOne(one);

//Throws error : int NumberOne::num is private;

//friendSetNumberOne(one, 100);

two.display();

two=friendDuplicateclass(one, 2);

friendDisplayNumberOne(two);

return 0;

}