Friend Function&Classes in C++

A Detailed Overview

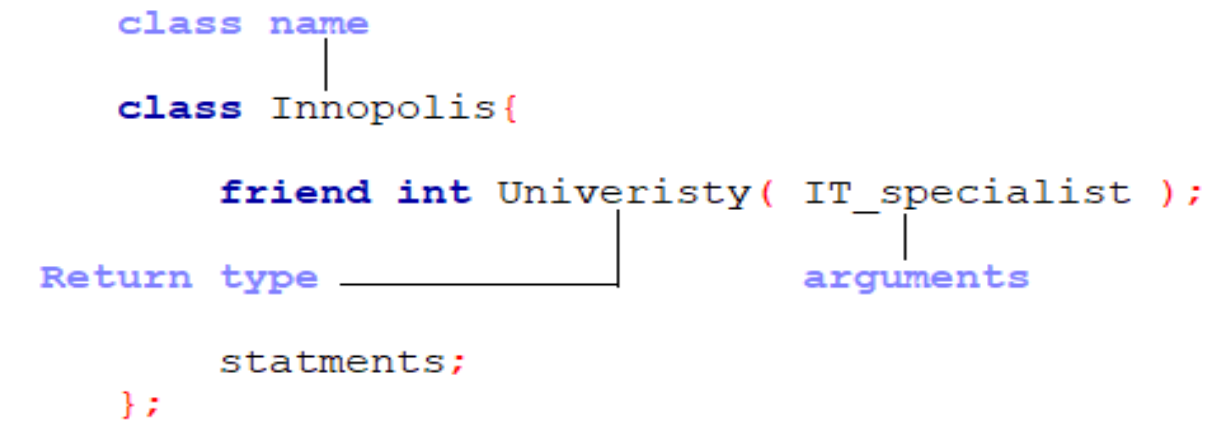
Assoev Habibullo & Yoqub Davlatov

Introduction

- The friend function allows non-member functions or other classes to access private/protected members.

Features

- - Declared with 'friend' keyword
- - Not a member of the class
- - Can be a global function or another class member



Purpose and Use Cases

- - Allows access to private/protected members
- - Helps in operator overloading
- - Useful for external function access

Friend Function Examples

```cpp
#include <iostream>
using namespace std;

class Box {
private:
    int width;

public:
    Box(int w) : width(w) {}

    // Declaring a friend function
    friend void printWidth(const Box& b);
};

// Friend function definition
void printWidth(const Box& b) {
    cout << "Width of box: " << b.width << endl;  // Can access private member
}

int main() {
    Box b(10);
    printWidth(b);
    return 0;
}
```

// Width of box: 10

```cpp
#include <iostream>
using namespace std;

class Box {
private:
    int width;

public:
    Box(int w) : width(w) {}

    // Declaring a friend function
    void printWidth(const Box& b);
};

// Friend function definition
void printWidth(const Box& b) {
    cout << "Width of box: " << b.width << endl;  // Can access private member
}

int main() {
    Box b(10);
    printWidth(b);
    return 0;
}
```

// width is private in this scope

Friend function with multiple functions

```cpp
#include <iostream>
using namespace std;

class ClassA;

class ClassB {
public:
    void show(ClassA& a);
};

class ClassA {
private:
    int value;

public:
    ClassA(int v) : value(v) {}

    friend void ClassB::show(ClassA& a);  // Friend function declaration in ClassB
};

void ClassB::show(ClassA& a) {
    cout << "Value from ClassA: " << a.value << endl;
}

int main() {
    ClassA a(10);
    ClassB b;
    b.show(a);  // Accesses private member of ClassA
    return 0;
}
```

// Value from ClassA: 10

Friend function in derived class

```cpp
#include <iostream>
using namespace std;

class Base {
private:
    int value;

public:
    Base(int v) : value(v) {}

    friend void showValue(Base& b);  // Friend function
};

class Derived : public Base {
public:
    Derived(int v) : Base(v) {}
};

void showValue(Base& b) {
    cout << "Base value: " << b.value << endl;  // Accessing private member of Base
}

int main() {
    Derived d(20);
    showValue(d);  // This works because showValue is a friend of Base
    return 0;
}
```

// Base value: 20

```cpp
#include <iostream>
using namespace std;

class Base {
private:
    int value;

public:
    Base(int v) : value(v) {}

    friend void showValue(Base& b);  // Friend function
};

class Derived : public Base {
public:
    Derived(int v) : Base(v) {}
};

void showValue(Derived& b) {
    cout << "Base value: " << b.value << endl;  // Accessing private member of Base
}

int main() {
    Derived d(20);
    showValue(d);  // This works because showValue is a friend of Base
    return 0;
}
```

// int Base::value is private within this scope

Friend functions with Templates

```cpp
#include <iostream>

template <typename T>
class MyClass {
private:
  T value;

public:
  MyClass(T v) : value(v) {}

  template <typename U> friend void printValue(const MyClass<U> &obj);
};

// The definition of the friend function
template <typename T> void printValue(const MyClass<T> &obj) {
  std::cout << "Value: " << obj.value << std::endl;
}

int main() {
  MyClass<int> obj(10);
  printValue(obj); // Works for template class
  return 0;
}
```

// Value: 10

```
class Student;

class Person {
private:
  std::string name;
  int age;

public:
  Person(const std::string &name, const int age)
      : name(name), age(age) {}

protected:
  friend class Student;
};
```

A **friend class** can access the private and protected members of another class.

Friend class declaration

```
class Student {
private:
  std::string name;
  int age;

  friend std::ostream
  &operator<<(std::ostream &out, const Student &student);

public:
  Student(const Person &person) {
    this->name = person.name;
    this->age = person.age;
  }
};
```

Friend function declaration

Student has access to Person private fields

Main Function

```
std::ostream &operator<<(std::ostream &out, const Student &student) {
    return out << "Student{\"name\":\"" << student.name
               << "\", \"age\":" << student.age << "}";
}

int main() {
    Person person(name: "Vasya", age: 20);
    Student student(person);
    std::cout << student << std::endl;
}
```

```
// Student{"name":"Vasya", "age":20}
```

A friend of your friend is not your friend

```
class A {
private:
  void printA() const { std::cout << "Print A" << std::endl; }

  friend class B;
};

class B {
private:
  void printB() const { std::cout << "Print B" << std::endl; }

  friend class C;
};

class C {
public:
  C(const B &b) { b.printB(); }
  C(const A &a) { a.printA(); } // 'printA' is a private member of 'A'
};
```

Important Takeaways

- **Advantages of Friend Functions/Classes:**
- Allows non-member functions to access private/protected members directly.
- Avoids the need for getters/setters in some cases, making the code cleaner.
- Commonly used for overloading operators like +, <<, etc.

- **Disadvantages of Friend Functions:**
- Violates the principle of encapsulation.
- Creates strong dependencies between the function and the class.
- Overuse can lead to complex, hard-to-maintain code.