Friend functions

A function which is a friend with a class has access to fields of a class (even private fields).

While writing a class we can specify, which fields are private – only methods of the class can access these fields. In some situations it can be interesting to allow functions outside of a class to access its fields. To allow such a relation between a class and a function we have to define inside a class definition friendship with a given function by defining it with the protected word **`friend.`**

The key observation is that it is a class and not a function which claims a friendship.

Example of friend functions

The class definition:

```
class Piece {
int x, y ;

    friend void report(Piece &);

};
```

The function declaration:

```
void report (Piece & p)
{
    cout << " The piece coordinates are " << p.x << ”, ” << p.y << endl;
}
```

Why do we need friend functions?

- A function can be a friend of many classes
- A friend function can convert arguments – we will talk about this later on
- Thanks to friend functions we can allow functions which cannot be methods to access fields of the class. Such a function can be, for example, written in another programming language.

C++

```cpp
#include <iostream>

extern"C" {
void fortfunc_(int *ii, float *ff);
}

main()
{

  int ii=5;
  float ff=5.5;

  fortfunc_(&ii, &ff);

  return 0;
}
```

Fortran

```fortran
      subroutine fortfunc(ii,ff)
      integer ii
      real*4  ff

      write(6,100) ii, ff
 100  format('ii=',i2,' ff=',f6.3)

      return
      end
```

```
gfortran -c testF.f
g++ -c testC.cpp
g++ -o test testF.o testC.o -lg2c
```

Friend functions in details

- A friend function is not a member of a class therefore it has no access to **`this`** pointer. To access fields we have to use `"."` or `"->"` operators
- We can declare a friendship in any place inside a class: private, protected, public has no importance
- We can declare and define a friend function inside a class declaration:
    - such a function is *inline*,
    - a friend function defined in this way is not a member of the class,
    - such a defined function is inside a lexical range of the class:
        - such function can use types defined inside a class – **`typedef`**,
        - it can use enumerations defined with **`enum`**

Friend functions and overloading

In the case of overload function only the function of the signature declared as a friend has access to fields of a class. For example,

```
class K
{
  friend void alarm(K obj, int k);
};

void alarm(float*, K obj);
void alarm(void) ;
void alarm(K obj, int i) ;
```

A method can be a friend function too

We can define a friendship between a class and a method of another class.

Class friendship

A class can declare friendship with another class.

```
class K {

friend class M ;
// ...
};
```

The friendship between class K and M allows class M to access all fields of class K. Note that the friendship declaration is one direction only – class M did not declare the friendship with class K.

Mutual class friendship

Two classes can declare a mutual friendship. The only way to obtain such a friendship is to declare friendship from one class to another and *vice versa*.

When we declare a friendship between classes then our compiler needs to know a given class---declared as a friend---before the friendship declaration. Normally, one class will be known to a compiler before another. How we can solve this problem?

Mutual class friendship

```
class Second;  // <- predclaration of a class

class First {
   friend class Second;
// ...
};

class Second {
   friend class First;
// ...
};
```

Friendship is not transitive

A friend of your friend is not your friend

Friendship is not inherited

Your friend's children are not your friends