OOPS

Friend Functions

A ***friend function*** is a function that can ***access the private members*** of a class as though it were a member of that class. To declare a friend function, simply use the ***friend keyword*** in front of the prototype of the function you wish to be a friend of the class. It does not matter whether you declare the friend function in the private or public section of the class.

```
#include <iostream>
using namespace std;
class Adder {
  private:
   int val;

  public:
   Adder() { val = 0; }
   void add(int value) { val += value; }
   // Make the reset() function a friend of this class
   friend void reset(Adder &Adder);
};
// reset() is now a friend of the Adder class
void reset(Adder &Adder) {
   // And can access the private data of Adder objects
   Adder.val = 0;
}
int main() {
   Adder acc;
   acc.add(5);  // add 5 to the Adder
   reset(acc);  // reset the Adder to 0
   return 0;
}
```

Points to Note:

We've declared a function named reset() that takes an object of class Adder, and sets the value of ***val*** to 0.

Because reset() is not a member of the Adder class, normally reset() would not be able to access the private members of Adder.

However, because Adder has specifically declared this reset() function to be a friend of the class, the reset() function is given access to the private members of Adder.

Note that we have to pass an Adder object to reset(). This is ***because reset() is not a member function***. It does not have a *this pointer, nor does it have an Adder object to work with, unless given one.

Multiple friends

A function can be a friend of more than one class at the same time. For example, consider the following example:

```
#include <iostream>
using namespace std;
class Contest;
class Course {
  private:
   int progress_course;
  public:
   Course(int temp = 0) { progress_course = temp; }
   friend void printProgress(const Course &course, const Contest &contest);
};

class Contest {
  private:
   int progress_contest;
```

```
  public:
   Contest(int contest = 0) { progress_contest = contest; }
   friend void printProgress(const Course &course, const Contest &contest);
};

void printProgress(const Course &course, const Contest &contest) {
   cout << "Course Progress is: " << course.progress_course << "\n"
        << "Contest Progress is: " << contest.progress_contest << '\n';
}

int main() {
   Contest ct(60);
   Course co(80);
   printProgress(co, ct);
   return 0;
}
```

Output:

```
Course Progress is: 80
Contest Progress is: 60
```

Points to Note:

Because printProgress is a friend of both classes, it can access the private data from objects of both classes.

Note the following line at the top of the example:

```
class Contest;
```

This is a class prototype that tells the compiler that we are going to define a class called ***Contest*** in the future. Without this line, the compiler would tell us it doesn't know what a Contest is when parsing the prototype for ***printProgress()*** inside the ***Course*** class.

Class prototypes serve the same role as function prototypes - they tell the compiler what something looks like so it can be used now and defined later.

However, unlike functions, classes have no return types or parameters, so class prototypes are always simply ***class ClassName***, where ClassName is the name of the class.

Friend Classes

It is also possible to make an entire class a friend of another class. This gives all of the ***members of the friend class access to the private members of the other class***.

```
#include <iostream>
using namespace std;
class Contest;
class Course {
  private:
   int progress_course;
  public:
   Course(int temp = 0) { progress_course = temp; }
   // Make Contest Class Friend of Course Class
   friend class Contest;
};

class Contest {
  private:
   int progress_contest;
  public:
   Contest(int contest = 0) { progress_contest = contest; }
   void printProgress(const Course &course, const Contest &contest) {
      cout << "Course Progress is: " << course.progress_course << "\n"
           << "Contest Progress is: " << contest.progress_contest << '\n';
   }
```

```
};

int main() {
   Contest ct(60);
   Course co(80);
   ct.printProgress(co, ct);
   return 0;
}
```

Output:

```
Course Progress is: 80
Contest Progress is: 60
```

Points to Note:

Because the ***Contest*** class is a friend of ***Course***, any of Contest's members that use a Course class object can access the private members of Course directly.

Even though Contest is a friend of Course, Contest has no direct access to the *this pointer of Course objects.

Just because Contest is a friend of Course, that does not mean Course is also a friend of Contest.

If you want two classes to be friends of each other, both must declare the other as a friend.

Finally, if class A is a friend of B, and B is a friend of C, that does not mean A is a friend of C.

Be careful when using friend functions and classes, because it allows the friend function or class to violate encapsulation.

If the details of the class change, the details of the friend will also be forced to change. Consequently, limit your use of friend functions and classes to a minimum.