Friendly Functions
C++ allows the common function to be made friendly with more than one class, thereby allowing the function to have access to the private data of these classes. Such a function need not be a member of any of these classes.
To make an outside function "friendly" to class, we have to simply declare this function as a friend of the class as shown below:

```cpp
class ABC
{
    .......
    .......
    public:
    .......
    .......
    friend void xyz(void) //declaration
};
```
The function declaration should be preceded by the keyword `friend`. The function is defined elsewhere in the program like a normal C++ function. The function definition does not use either the keyword `friend` or the scope operator `::`. The functions that are declared with the keyword `friend` are known as `friend` function. A function can be declared as a friend in any number of classes. A friend function, although not a member function, has a full access rights to the private members of the class.

A friend function possesses certain special characteristics:
- It is not in the scope of the class to which it has been declared as friend.
- It cannot be called using the objects of that class. it can be invoked like a normal function without the help of any object.
- Unlike member function, it cannot access the member names directly and has to use an object name and dot membership operator with each member name. (e.g. A.x)
- It can be declared either in the public or private part of a class without affecting its meaning.
- Usually, it has the objects as arguments.

The friend functions are often used in operator overloading.

Program below illustrate the use of `friend` function.

```cpp
#include <iostream.h>

class sample
{
    int a;
    int b;
public:
    void setvalue() { a= 25; b = 40; }
    friend float mean(sample s);  // Friend declaration
};
```
return float( s.a + s.b) / 2.0;
}

int main(void)
{
    sample X; // object X
    X.setvalue( );
    cout << " mean value = " << mean(x) << "\n";
}

The output of program 5.8 would be:
Mean value: 32.5

Note that the friend function accesses the class variables \texttt{a} and \texttt{b} by using the dot operator and the object passed to it. The function calls \texttt{mean(X)} passes the object \texttt{X} by value to the friend function.

Member functions of one class can be friend functions of another class. In such cases, they are defined using the scope resolution operator as shown below:

```cpp
class X
{
    .......
    int fun( ); // member function of X
    .......
};

class Y
{
    .......
    ...
    friend int X::fun( ); // fun1 of X
    .......
     // is friend of Y
};
```

The function \texttt{fun1( )} is a number of class \texttt{X} and a friend of class \texttt{Y}.

We can also declare all the member functions of one class as the friend functions of another class. In such cases, the class is called a friend class. This can be specified as follows:

```cpp
class Z
{
    .......
    friend class X; // all member functions of X are friend to \texttt{Z}
};
```

Program below demonstrates how friend functions work as a bridge between the classes. Note that the function \texttt{max( )} has arguments from both \texttt{XYZ} and \texttt{ABC}. When the function \texttt{max( )} is declared as a friend in \texttt{XYZ}, for the first time, the compiler will not acknowledge the presence of \texttt{ABC} under its name is declared in the beginning as

```cpp
class ABC;
```

This is known as "forward" declaration.
As pointed out earlier, a friend function can be called by reference. In this case, local copies of the objects are not made. Instead, a pointer to the address of the object is passed and the called function directly works on the actual object used in the call.
This method can be used to alter the values of the private members of a class. Remember, altering the values of private member is against the basic principles of data hiding. It should be used only when absolutely necessary.
Program below shows how to use a common friend function to exchange the private values of two classes. The function is called by reference.
```cpp
#include <iostream.h>

class class_2;

class class_1
{
  int value1;
  public:
    void indata(int a) { value1 = a;}
    void display(void) { cout << value1 << "\n";}
  friend void exchange ( class_1 &_, class_2 &)
};

class class_2
{
  int value2;
  public:
    void indata( int a) { value2 = a;}
    void display(void) { cout << value2 << "\n";}
  friend void exchange ( class_1 &, class_2 &);
};

void exchange( class_1 x, class_2 & y)
{
  int temp = x.value1;
  x.value1 = y.value2;
  y.value2 = temp;
}

int main (void)
{
  class_1 C1;
  class_2 C2;
  C1.indata(100);
  C2.indata(200);
  cout << "values before exchange" << "\n";
  C1.dispaly();
  C2.dispaly();
  exchange(C1,C2); // swapping
  cout << " Values after exchange"<<"\n";
  C1.display();
  C2.display();
}
```

//program 5.6>