Member Functions and Friend Functions

Member functions and Data member

The variables declared inside the class are known as data members and the functions are known as member functions. The public members can be accessed from outside the class, but the private members can be accessed from inside the member functions of the same class.

Defining Member Functions

Member functions can be defined in two places:

1. Within the same class
2. Outside the class

Member Function definition outside the Class

Member functions that are declared inside a class have to be defined separately outside the class. The general form of a member function definition is

```
return-type class-name::function-name(argument declaration)
 {
        Function body
 }
```

The membership label class-name:: tells the compiler that the function function-name belongs to the class class-name.

Eg.:

```
#include<iostream>
using namespace std;
class Account
   {
     private:
       int number;
      double balance;
     public:
      void setData(int no, double bal);
      void display();
  };
  void Account::setData(int no, double bal)
       {
         number=no;
         balance=bal;
       }
  void Account::display()
       {
         cout<<"Account Number:"<<number;
         cout<<"Balance:"<<balance;
       }
```

```
int main()
    {
      Account a1;
      a1.setData(10001,5000.0);
      a1.display();
      return 0;
    }
```

The characteristics of member functions are

- Several different classes can use the same function name.
- Member functions can access the private data of the class.
- A member function can call another member function directly, without using the dot operator.

Member Function definition within the Class – inline Function

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

Eg.:

```
#include<iostream>
using namespace std;
class Account
   {
    private:
      int number;
     double balance;
    public:
     void setData(int no, double bal)
       {
         number=no;
         balance=bal;
       }
     void display()
       {
         cout<<"Account Number:"<<number;
         cout<<"Balance:"<<balance;
       }
  };

int main()
    {
      Account a1;
      a1.setData(10001,5000.0);
      a1.display();
      return 0;
    }
```

When a function is defined inside a class, it is treated as an inline function.

inline Functions

A member function of a class can be defined outside the class definition and still make it inline by just using the qualifier inline in the header line of function definition.

```
#include<iostream>
using namespace std;
class Account
   {
     private:
       int number;
      double balance;
     public:
      void setData(int no, double bal);
      void display();
  };

  inline void Account::setData(int no, double bal)
       {
         number=no;
         balance=bal;
       }
  inline void Account::display()
       {
         cout<<"Account Number:"<<number;
         cout<<"Balance:"<<balance;
       }

int main()
    {
      Account a1;
      a1.setData(10001,5000.0);
      a1.display();
      return 0;
    }
```

Scope Resolution Operator

The symbol of scope resolution operator is ::. This operator is used to indicate the scope of the function.

The scope resolution operator can also be used to access the global variable, if there is a local variable with the same name.

Eg.:

```
#include<iostream>
using namespace std;
int x=10;
int main()
 {
    int x=20;
```

```
    cout<<"\nx="<<x;
    cout<<"\n::x="<<::x;
    return 0;
 }
```

Static or Class-wide Members

Static Data Members

A data member of a class can be qualified as static. A static member variable has certain special characteristics. These are:

- It is initialized to zero when the first object of its class is created.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class.
- It is visible only within the class, but its lifetime is the entire program.

Static variables are normally used to maintain values common to the entire class.

A static data member can be used as a counter that records the occurrences of all the objects.

```
#include<iostream>
using namespace std;
class Account
 {
   private:
    static int c;
    int number;
    double balance;

   public:
    void setData(double bal)
      {
       c++;
       number=c;
       balance=bal;
      }
    void display()
      {
       cout<<"\nAccount Number:"<<number;
       cout<<"\nBalance:"<<balance;
      }
      void totalAccounts()
      {
       cout<<"\nTotal number of Accounts "<<c;
      }
};

int Account::c;
```

```
int main()
 {
   Account a1,a2,a3;
   a1.setData(5000.0);
   a1.display();
   a2.setData(10000.0);
   a2.display();
   a3.setData(2000.0);
   a3.display();
   a1.totalAccounts();
   return 0;
 }
```

```
Account Number: 1
Balance: 5000
Account Number: 2
Balance: 10000
Account Number: 3
Balance: 2000
Total number of Accounts 3
```

Static Member Functions

Like static member variable, we can also have static member functions. A member function that is declared static has the following properties:

- A static function can have access to only other static members declared in the same class
- A static member function can be called using the class name as follows:

  class-name :: function-name;

The following program illustrates the implementation of these characteristics. The static function totalAccounts() displays the number of objects created till that moment. A count of number of objects created is maintained by the static variable c. The function display() displays the account number and balance of each object.

```
#include<iostream>
using namespace std;
class Account
 {
  private:
   static int c;
   int number;
   double balance;
  public:
   void setData(double bal)
     {
      c++;
      number=c;
      balance=bal;
     }
```

```
    void display()
     {
      cout<<"\nAccount Number:"<<number;
      cout<<"\nBalance:"<<balance;
     }
    static void totalAccounts()
     {
      cout<<"\nTotal number of Accounts "<<(c-1000);
     }
};
int Account::c=1000;
int main()
 {
   Account a1,a2,a3;
   a1.setData(3000.0);
   a1.display();
   a2.setData(5000.0);
   a2.display();
   a3.setData(7000.0);
   a3.display();
   Account::totalAccounts();
   return 0;
 }
```

```
Account Number: 1
Balance: 3000
Account Number: 2
Balance: 5000
Account Number: 3
Balance: 7000
Total number of Accounts 3
```

Constant member Functions

We can declare a member function as a constant member function. The constant member function does not alter the value of the instance variables.

```
#include<iostream>
using namespace std;
class Account
 {
  private:
   int number;
   double balance;
  public:
   void setData(int no,double bal)
    {
     number=no;
     balance=bal;
    }
```

```
  void display() const
   {
     cout<<"\nAccount Number:"<<number;
     cout<<"\nBalance:"<<balance;
   }
};

int main()
 {
   Account a1,a2;
   a1.setData(1001,3000.0);
   a1.display();
   a2.setData(1002,5000.0);
   a2.display();
   return 0;
 }
```

Constant Member Function Arguments

The individual arguments of the member functions can also be declared as constants. The constant arguments can not be modified by the respective functions.

```
#include<iostream>
using namespace std;
class Account
 {
  private:
   int number;
   double balance;
  public:
   void setData(const int no, const double bal)
    {
     number=no;
     balance=bal;
    }
   void display() const
    {
     cout<<"\nAccount Number:"<<number;
     cout<<"\nBalance:"<<balance;
    }
};

int main()
 {
   Account a1,a2,a3;
   a1.setData(1001,3000.0);
   a1.display();
   a2.setData(1002,5000.0);
   a2.display();
   return 0;
 }
```

friend Function

Friendly function is a function used to access private data members of a class without using object of that class. To make an outside function "friendly" to a class, we have to simply declare this function as a friend of the class as shown below:

```
class ABC
 {
        …..
        …..
    public:
        …..
        …..
        friend void xyz(void); //friendly function 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 functions.

The special characteristics of a friendly function are

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

Eg.:

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

class Account
 {
  private:
   int number;
   double balance;
  public:
   void setData(int no,double bal)
     {
      number=no;
      balance=bal;
     }
  friend void display(Account);
 };
```

```
void display(Account a)
     {
       cout<<"\nAccount Number:"<<a.number;
       cout<<"\nBalance:"<<a.balance;
     }
int main()
  {
    Account a1,a2,a3;
    a1.setData(1001,3000.0);
    display(a1);
    a2.setData(1002,5000.0);
    display(a2);
    return 0;
  }
```

We can also define a friend function, which will be used by more than one class.

```
#include<iostream>
using namespace std;
class FDAccount;
class SBAccount
 {
   private:
    int number;
    double balance;
   public:
    void setData(int no,double bal)
      {
        number=no;
        balance=bal;
      }
   friend void display(SBAccount,FDAccount);
};

class FDAccount
 {
   private:
    int number;
    double balance;
   public:
    void setData(int no,double bal)
      {
        number=no;
        balance=bal;
      }
   friend void display(SBAccount,FDAccount);
};
```

```
void display(SBAccount sb,FDAccount fd)
     {
      if (sb.number==fd.number)
      {
        cout<<"\nAccount Number:"<<sb.number;
        cout<<"\nSB Balance:"<<sb.balance;
        cout<<"\nFD Balance:"<<fd.balance;
        cout<<"\nTotal Balance:"<<sb.balance+fd.balance;
      }
      else
      {
         cout<<"Different Accounts";
      }
     }
int main()
  {
    SBAccount sa1;
    FDAccount fa1;
    sa1.setData(1001,3000.0);
    fa1.setData(1001,5000.0);
    display(sa1,fa1);
    return 0;
  }
```

```
Account Number:1001
SB Balance:3000
FD Balance:5000
Total Balance:8000
```

friend Class

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:

```
class Z
  {
        …..
        friend class X;  //all member functions of X are friends to Z
        …..
  };
```

Example:

```
#include<iostream>
using namespace std;
class First
 {
   int no;
  public:
```

```
    void setData(int n)
      {
        no=n;
      }
  friend class Second;
};

class Second
 {
   public:
   void display(First f)
    {
      cout<<"\nNo: "<<f.no;
    }
};

int main()
  {
     First f1;
     Second s1;
     f1.setData(100);
     s1.display(f1);
     return 0;
  }
```

Output:

No: 100