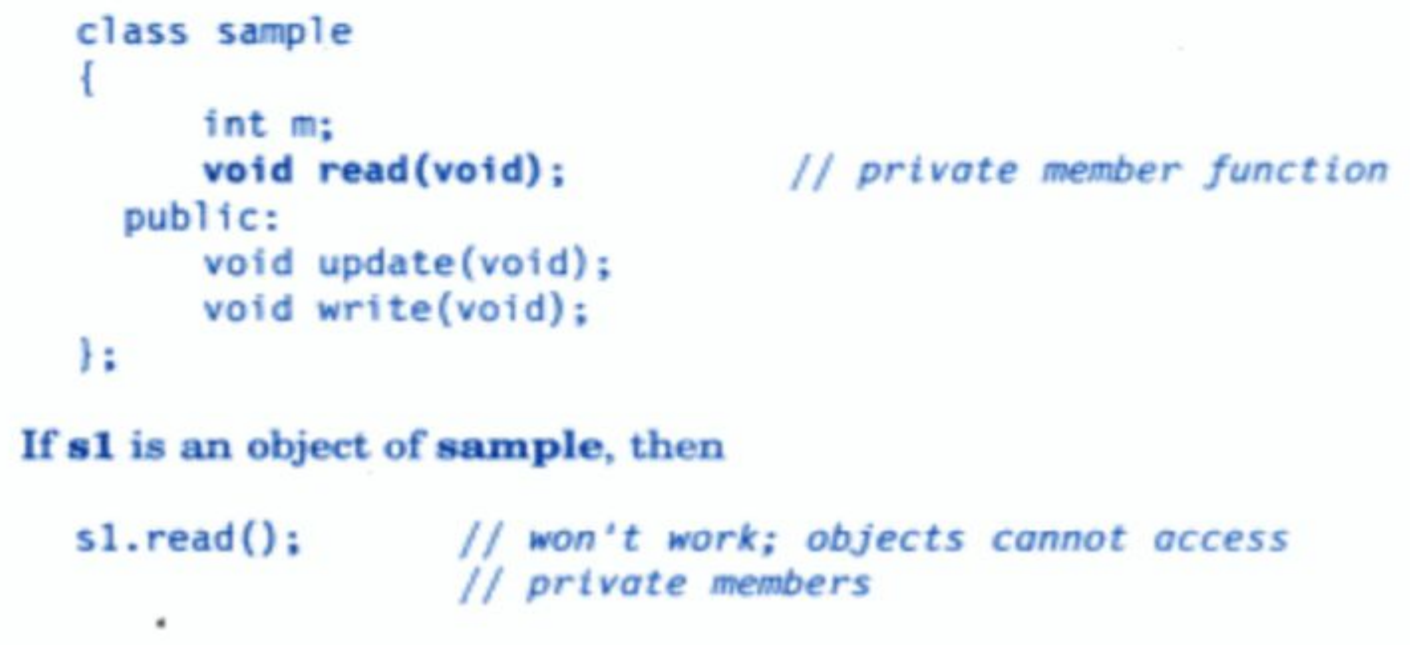
Classes and Objects

- Private Member Functions
- Memory Allocation for Objects
- Static Data Members
- Static Member Functions
- Array of Objects
- Objects as a Function Arguments
- Friendly Function

Private Member Functions

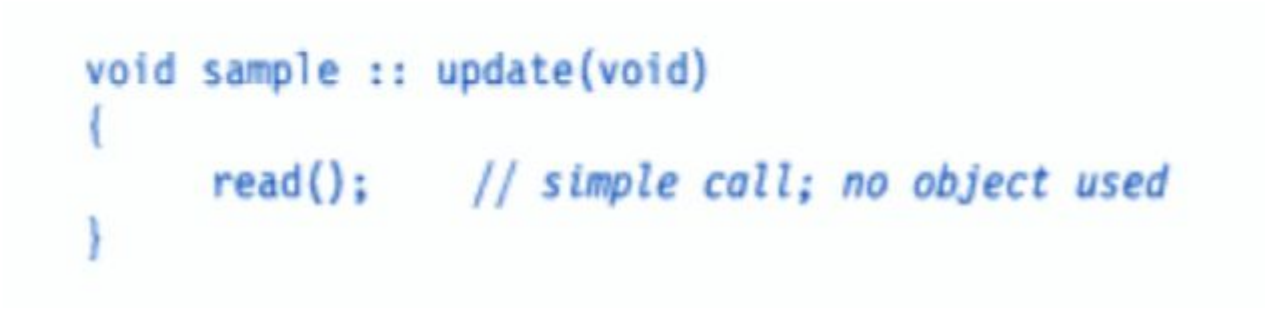


```
class sample
{
        int m;
        void read(void);              // private member function
  public:
        void update(void);
        void write(void);
};
```

If **s1** is an object of **sample**, then

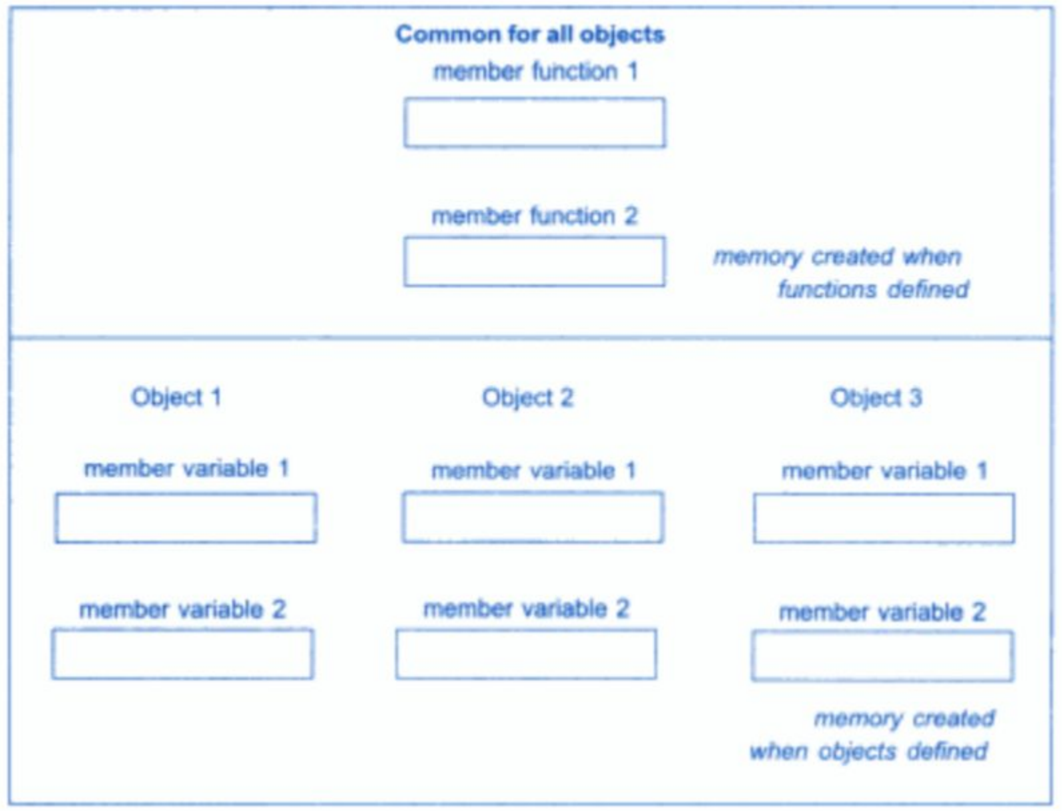
```
s1.read();          // won't work; objects cannot access
                    // private members
```

Private Member Functions



```
void sample :: update(void)
{
        read();     // simple call; no object used
}
```

Memory Allocation for Objects



Static Data Members

A data members of a class can be qualified as static. A static member variables has certain special characteristics;

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all objects of that class, no matter how many objects are created.
- 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, e.g count. It can be defined as

```
int item::count;
```

Note that, the type and scope of each static member variable must be defined outside the class.

Static Data Members

```
#include<iostream>
using namespace std;
class item
{
        static int count;
        int number;
    public:
        void getdata(int a)
        {
            number=a;
            count++;
        }
        void display_count()
        {
            cout<<"Count = "<<count<<endl;
        }
};

int item::count;    //count definition
```

```
int main()
{
    item x,y,z;
    x.display_count();
    y.display_count();
    z.display_count();

    x.getdata(10);
    y.getdata(20);
    z.getdata(30);

    cout<<"\nCount values after
reading data\n";
    x.display_count();
    y.display_count();
    z.display_count();
}
```

Static Data Members

```
#include<iostream>
using namespace std;
class item
{
        static int count;
        int number;
    public:
        void getdata(int a)
        {
            number=a;
            count++;
        }
        void display_count()
        {
            cout<<"Count = "<<count<<endl;
        }
};

int item::count;    //count definition
```

Output:

```
Count=0
Count=0
Count=0

After reading Data
Count=3
Count=3
Count=3
```

```
int main()
{
    item x,y,z;
    x.display_count();
    y.display_count();
    z.display_count();

    x.getdata(10);
    y.getdata(20);
    z.getdata(30);

    cout<<"After reading data\n";
    x.display_count();
    y.display_count();
    z.display_count();
}
```

Static Member Functions

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

- A static function can have access to only other static members(functions or variables) declared in the same class.
- A static member function can be called using the class name(instead of its objects) as follows:

```
class_name::function_name;
```

Static Member Functions

```
#include<iostream>
using namespace std;
class item
{       int code;
        static int count;
    public:
        void setcode()
        {
            code=++count;
        }
        void display_code()
        {
            cout<<"Object Member: "<<code<<"\n";
        }
        static void display_count()
        {
            cout<<"Count : "<<count<<endl;
        }
};
```

```
int item::count;
int main()
{
    item x,y;
    x.setcode();
    y.setcode();

item::display_count();

    item z;
    z.setcode();


item::display_count();

    x.display_code();
    y.display_code();
    z.display_code();
}
```

Static Member Functions

```
#include<iostream>
using namespace std;
class item
{       int code;
        static int count;
    public:
        void setcode()
        {
            code=++count;
        }
        void display_code()
        {
            cout<<"Object Member: "<<code<<"\n";
        }
        static void display_count()
        {
            cout<<"Count : "<<count<<endl;
        }
};
```

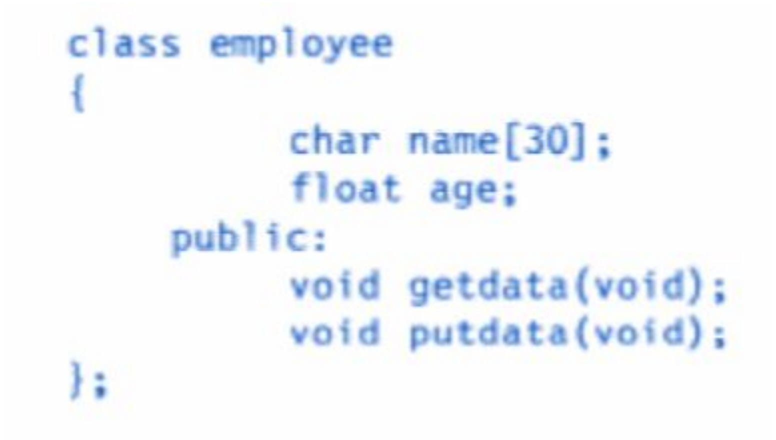
Output:

```
Count=2
Count=3

Object Member=1
Object Member=2
Object Member=3
```

```
int item::count;
int main()
{
    item x,y;
    x.setcode();
    y.setcode();

item::display_count();

    item z;
    z.setcode();


item::display_count();

    x.display_code();
    y.display_code();
    z.display_code();
}
```

Array of Object

Consider the following class definition



```
class employee
{
            char name[30];
            float age;
      public:
            void getdata(void);
            void putdata(void);
};
```

Consider following objects to above class employee

```
employee manager[3];            // array of manager
employee foreman[15];           // array of foreman
employee worker[75];            // array of worker
```

Array of Object

```
#include<iostream>
using namespace std;
class employee
{
        char name[20];
        float age;
    public:
        void getdata();
        void putdata();
};
void employee::getdata()
{
    cout<<"Enter Name : ";
    cin>>name;
    cout<<"Enter Age : ";
    cin>>age;
}
```

```
void employee::putdata()
{   cout<<"Name : "<<name<<"\n";
    cout<<"Age :"<<age<<endl;
};
int main()
{
    employee manager[3];
    for(int i=0;i<3;i++)
    {   cout<<"Enter Details\n";
        manager[i].getdata();
    }
    for(int i=0;i<3;i++)
    {
        cout<<"\n Manager Details
are :\n";
        manager[i].putdata();
    }
}
```

Objects as a Function Arguments

```
#include<iostream>
using namespace std;
class time
{
        int hours;
        int minutes;
    public:
        void gettime(int h, int m)
        {   hours=h;
            minutes=m;
        }
        void puttime()
        {
            cout<<hours<<":"<<minutes<<endl;
        }
        void sum(time t1,time t2);
};
```

```
void time::sum(time t1,time t2)
{   minutes=t1.minutes+t2.minutes;
    hours=minutes/60;
    minutes=minutes%60;
    hours=hours+t1.hours+t2.hours;
}

int main()
{   time T1,T2,T3;
    T1.gettime(2,45);
    T2.gettime(3,30);

    T3.sum(T1,T2);

    T1.puttime();
    T2.puttime();
    T3.puttime();
}
```

Objects as a Function Arguments

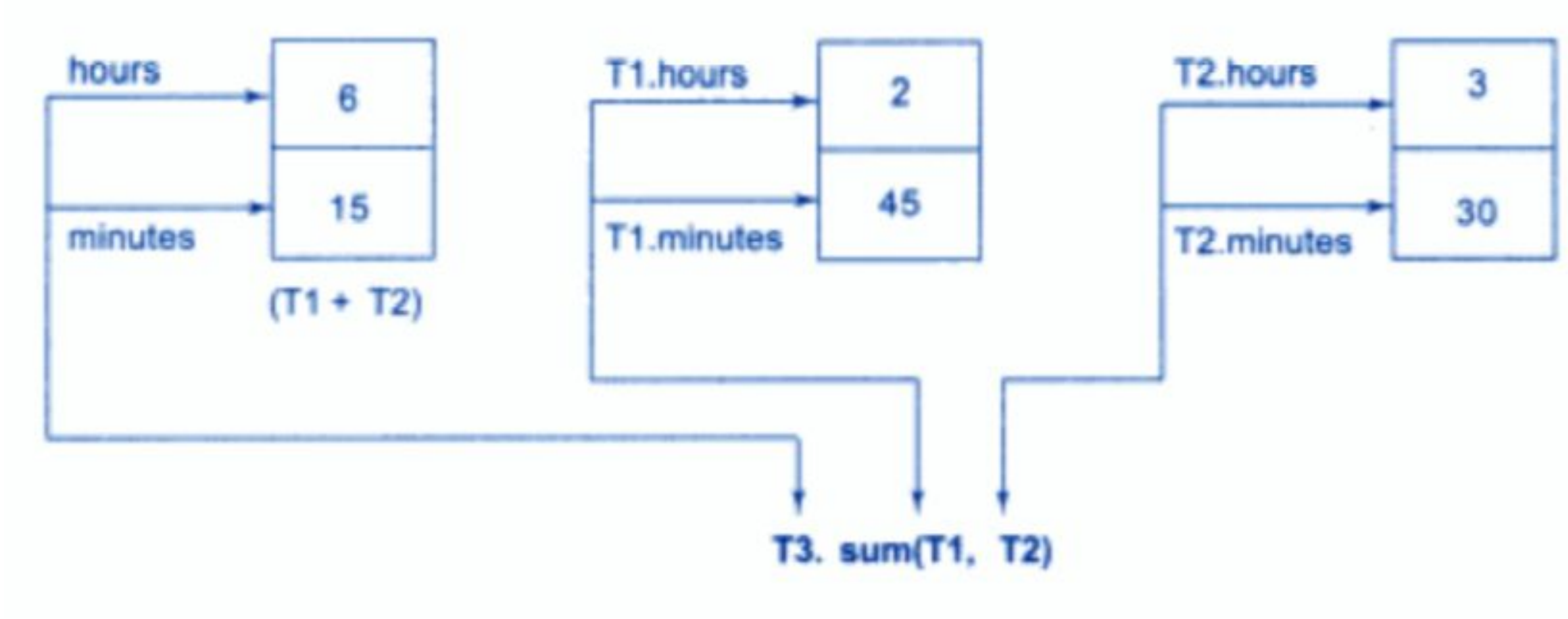
```
#include<iostream>
using namespace std;
class time
{
        int hours;
        int minutes;
    public:
        void gettime(int h, int m)
        {   hours=h;
            minutes=m;
        }
        void puttime()
        {
            cout<<hours<<":"<<minutes<<endl;
        }
        void sum(time t1,time t2);
};
```

```
void time::sum(time t1,time t2)
{   minutes=t1.minutes+t2.minutes;
    hours=minutes/60;
    minutes=minutes%60;
    hours=hours+t1.hours+t2.hours;
}

int main()
{   time T1,T2,T3;
    T1.gettime(2,45);
    T2.gettime(3,30);

    T3.sum(T1,T2);

    T1.puttime();
    T2.puttime();
    T3.puttime();
}
```

Output:

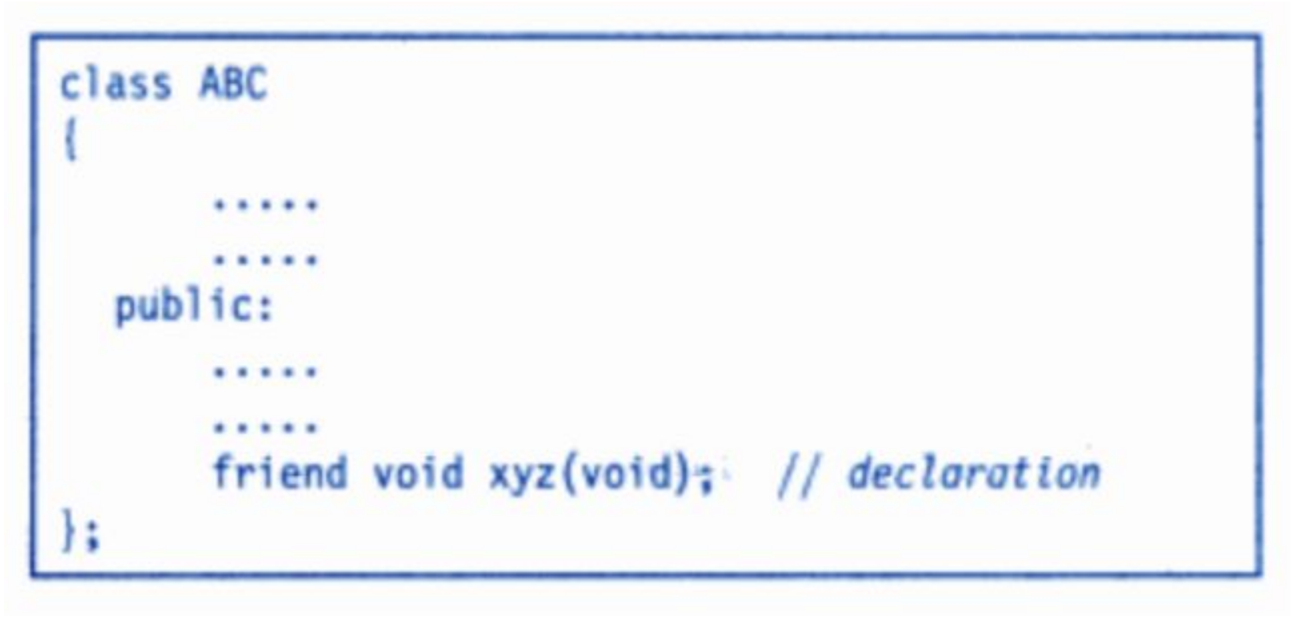
```
2:45
3:30
6:15
```

Objects as a Function Arguments



Friendly Function

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);   // declaration
};
```

Friendly Function

A friend function possesses certain special characteristics:

- It is not in the scope of the class to which it has been declared as friend.
- Since it is not in the scope of the class, it can not be called using the object of that class. It can be invoked like a normal function without the help of any object.
- Unlike member functions, it can not access the member names directly and has to use 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 in the private part of a class without affecting its meaning.
- Usually, it has the object as a arguments.

Friendly Function

```
#include<iostream>
using namespace std;
class sample
{
        int a;
        int b;
    public:
        void getvalue()
        {
            a=25; b=40;
        }
        friend float mean(sample s);
};

float mean(sample s)
{
    return float(s.a+s.b)/2.0;
}
```

```
int main()
{
    sample s;
    s.getvalue();
    cout<<" Mean Value is : "<<mean(s);
}
```

```
Output:

Mean Value is : 32.5
```