"const" Member Function

```
#include <iostream>
using namespace std;
class Student
{
   int roll;
   char grade;
   float per;
public:
   void get();
   void show()const;
};
void Student::get()
{
   cout<<"Enter roll,grade and per:";
   cin>>roll>>grade>>per;
}
void Student::show()const
{
  cout<<roll<<","<<grade<<","<<per<<endl;
}
```

```
int main()
{
   Student S;
   S.get();
   S.show();
   return 0;
}
```

"const" object
==========

A constant object in C++ is an object whose values can never change once it has been initialized.

For example: Suppose we have a class call Date and we create multiple object of the Date class as shown below:

```
Date today(16,10,2022);
Date ind_day(15,8,1947);
```

Now suppose we have a member function in the Date class called as changeDate(). This member function changes the date of calling object. Now, if we call

```
ind_day.changeDate();
```

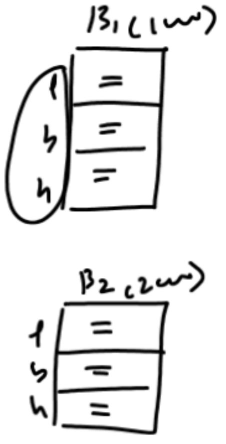
The above code will compile and run but it will accidently change independence date which is not acceptable behaviour. To stop the compiler from compiling the above code we must declare the object as **ind_day** as **const object** as shown below:

```
const Date ind_day(15,8,1947);
ind_day.changeDate(); // Syntax error
```

Passing Object As Argument To Member Functions

```
class Box
{
    int l, b, h;

public:
        void get( )
        {
            //input
        }
        void show( )
        { //display
        }
  { compare( )
 ;
```



```
int main( )
{
  Box B1, B2;
  B1.get( );
  B2.get( );
  B1.compare( );
~~B1 B2.compa( )~~;
```

Passing Object As Argument To Member Functions

In C++ just like we can pass variables as arguments to member functions, similarly we also can pass objects as argument to member function. This is required whenever we have a single member function working on multiple objects at the same time.

For example: suppose we have a class called **Box** and we have created two objects of the **Box** class called **b1** and **b2.** Now, suppose we want to compare volumes of **b1** and **b2** objects and find out whose volume is greater.

In this case we will require a member function called **compareVolume().** This member function will be called by **b1** and **b2** will be passed as argument and the call will look like:

b1.compareVolume(b2);

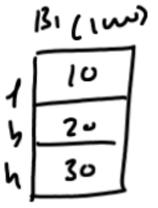
So in general we can say that if we want to define a member function which has to work upon **'n'** objects at the same time, then it will require **'n-1'** object to be passed as argument.

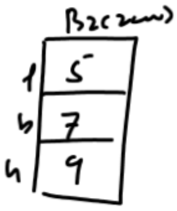
Just like we can pass variables in 3 ways as argument to member functions, similarly we also can pass objects also in 3 ways as argument and they are:

1. **Passing Object by Value**
2. **Passing Object by Address**
3. **Passing Object by Reference**

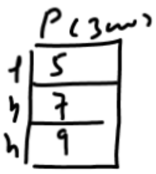
Passing Object By Value

```
#include <iostream>
using namespace std;
class Box
{
   int l,b,h;
public:
        void get()
        {
         cout<<"Enter l,b,h:";
         cin>>l>>b>>h;
        }
        void show()
        {
          cout<<l<<","<<b<<","<<h<<endl;}
        }
        int compareVol(Box);
};
```





```
int Box::compareVol(Box P)
{
  int x,y;
  x=l*b*h;
  y=P.l*P.b*P.h;
  if(x>y)
    return 1;
  else if(y>x)
    return -1;
  else
    return 0;
```

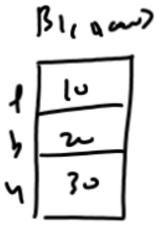


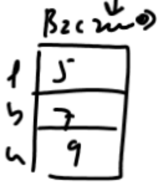
```
int main()
{
  Box B1; ✓
  Box B2; ✓
  B1.get(); ✓
  B2.get(); ✓
  B1.show();   10, 20, 30
  B2.show();   5, 7, 9
  int ans;
  ans=B1.compareVol(B2);
  if(ans==1)
       cout<<"Vol of B1 is gr";
  else if(ans==-1)
      cout<<"Vol of B2 is gr";
  else
      cout<<"Both are eq";
return 0;
}
```

STYLE	Fn Call	Memory	
① Pass By Value	11	36B	→ Inefficient
② Pass By Addr	9	32B	
③ Pass By Ref	9	32B	→ Best style

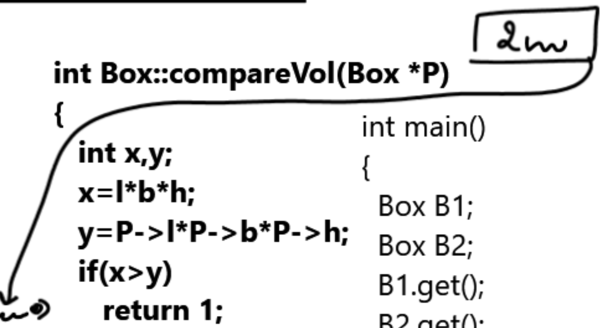
Passing Object By Address

```
#include <iostream>
using namespace std;
class Box
{
   int l,b,h;
public:
        void get()
        {
         cout<<"Enter l,b,h:";
         cin>>l>>b>>h;
        }
        void show()
        {
          cout<<l<<","<<b<<","<<h<<endl;}
        }
        int compareVol(Box *);
};
```





```
int Box::compareVol(Box *P)
{
  int x,y;
  x=l*b*h;
  y=P->l*P->b*P->h;
  if(x>y)
    return 1;
  else if(y>x)
    return -1;
  else
    return 0;
```

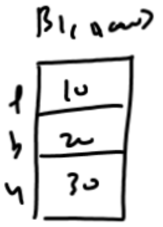


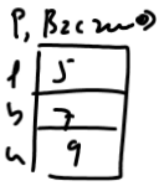
```
int main()
{
  Box B1;
  Box B2;
  B1.get();
  B2.get();
  B1.show();
  B2.show();
  int ans;
  ans=B1.compareVol(&B2);
  if(ans==1)
       cout<<"Vol of B1 is gr";
  else if(ans==-1)
      cout<<"Vol of B2 is gr";
  else
      cout<<"Both are eq";
return 0;
}
```

2000

Passing Object By Reference

```
#include <iostream>
using namespace std;
class Box
{
   int l,b,h;
public:
        void get()
        {
         cout<<"Enter l,b,h:";
         cin>>l>>b>>h;
        }
        void show()
        {
          cout<<l<<","<<b<<","<<h<<endl;}
        }
        int compareVol(Box &);
};
```





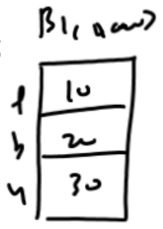
```
int Box::compareVol(Box &P)
{
  int x,y;
  x=l*b*h;
  y=P.l*P.b*P.h;
  if(x>y)
    return 1;
  else if(y>x)
    return -1;
  else
    return 0;
```

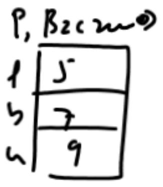
```
int main()
{
  Box B1;
  Box B2;
  B1.get();
  B2.get();
  B1.show();
  B2.show();
  int ans;
  ans=B1.compareVol(B2);
  if(ans==1)
       cout<<"Vol of B1 is gr";
  else if(ans==-1)
      cout<<"Vol of B2 is gr";
  else
      cout<<"Both are eq";
return 0;
}
```

2000

Passing Object By Reference

```
#include <iostream>
using namespace std;
class Box
{
   int l,b,h;
public:
          void get()
          {
           cout<<"Enter l,b,h:";
           cin>>l>>b>>h;
          }
          void show()
          {
            cout<<l<<","<<b<<","<<h<<endl}
          }
          int compareVol(const Box &);
};
```





```
int Box::compareVol(const Box &P)
{
   int x,y;
   x=l*b*h;
   y=P.l*P.b*P.h;
   if(x>y)
      return 1;
   else if(y>x)
      return -1;
   else
      return 0;
```

```
int main()
{
  Box B1;
  Box B2;
  B1.get();
  B2.get();
  B1.show();
  B2.show();
  int ans;                                  2000
  ans=B1.compareVol(B2);
  if(ans==1)
       cout<<"Vol of B1 is gr"
  else if(ans==-1)
      cout<<"Vol of B2 is gr";
  else
      cout<<"Both are eq";
return 0;
}
```