```
//Call by value
#include <iostream>
using namespace std;
void swap(int, int);// function declaration
int main ()
{
  int a = 100;     // local variable declaration
  int b = 200;
  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl<<endl;
  swap(a, b);    // calling a function to swap values
  cout << "After swap, value of a (in main()):" << a << endl;
  cout << "After swap, value of b (in main()):" << b << endl;
  return 0;
}
void swap(int x, int y)
{
  int temp;
  temp = x;   // save the value of x
  x = y;     // put y into x
  y = temp; // put x into y
  cout << "After swap, value of x(a) (in definition):" << x << endl;
  cout << "After swap, value of y(b) (in definition):" << y << endl<<endl;
}
```

Advantages:

1) Original values are safe, because entire work is done on the duplicate copies

2) We can pass variables,constants and expressions as actual arguments

Disadvantages:

It will consume more memory and will take more time as well

```
//Call by address/or call by pointer
#include <iostream>
using namespace std;
void swap(int*, int*);// function declaration
int main ()
{
  int a = 100;     // local variable declaration
  int b = 200;
  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl<<endl;
  swap(&a,&b);    // calling a function to swap values
  cout << "After swap, value of a (in main()):" << a << endl;
  cout << "After swap, value of b (in main()):" << b << endl;
  return 0;
}
void swap(int *px, int *py)
{
  int temp;
  temp = *px;
  *px = *py;
```

```
   *py = temp;
   cout << "(in definition):" << *px << endl;
   cout << "(in definition):" << *py << endl<<endl;
}
```

Advantage:

It will take less time and space as compared to call by value, because duplicate copies are not created

Disadvantage:

1) Original values are not safe..

2) We cannot pass direct variables, constants,expressions as actual arguments, as we can only pass addresses as actual arguments(Reason: Formal arguments are pointers)

```
   //Reference variable
#include <iostream>
using namespace std;
int main ()
{
   int a=10;
   int &b=a;
   b=b+2;
   cout<<a;
   b=50;
   cout<<endl<<a;
   return 0;
}
//Call by reference
#include <iostream>
using namespace std;
void swap(int &, int &);// function declaration
int main ()
{
   int a = 100;     // local variable declaration
   int b = 200;
   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl<<endl;
   swap(a,b);    // calling a function to swap values
   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
   return 0;
}
void swap(int &x, int &y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
   cout << "After swap, value of x :" << x << endl;
   cout << "After swap, value of y :" << y << endl<<endl;
}
//Single friend function for a class
```

```
#include<iostream>
using namespace std;
class sample
{
	int a;
	int b;
	public:
	friend float mean(sample);
		void setvalue()
		{
			a=45;
			b=40;
		}
};
float mean(sample s)
{
	return float(s.a+s.b)/2.0;
}
int main()
{
	sample X;
	X.setvalue();
	cout<<"Mean value for object X="<<mean(X)<<"\n";
	return 0;

}
//Single friend function for a class(Passing object as an argument(Call by value))
#include<iostream>
using namespace std;
class sample
{
	int a;
	int b;
	public:
	friend void modify(sample);
		void setvalue()
		{
			a=45;
			b=40;
		}
		void display()
		{
		   cout<<endl<<a<<" "<<b;
		}
};
void modify(sample s)
{
	s.a=100;
	s.b=200;
```

```
}
int main()
{
	sample X;
	X.setvalue();
	X.display();
	modify(X);
	X.display();
	return 0;
}
//Single friend function for a class(Passing object as an argument(Call by reference))
#include<iostream>
using namespace std;
class sample
{
	int a;
	int b;
	public:
	friend void modify(sample&);
		void setvalue()
		{
			a=45;
			b=40;
		}
		void display()
		{
		   cout<<endl<<a<<" "<<b;
		}
};
void modify(sample &s)//sample &s=X
{
	s.a=100;
	s.b=200;
}
int main()
{
	sample X;
	X.setvalue();
	X.display();
	modify(X);
	X.display();
	return 0;
}
//Single friend function for a class
#include<iostream>
using namespace std;
class sample
{
	int a,b,c;
```

```
        public:
        friend int largest(sample);
                void input()
                {
                   cout<<"\nEnter values of a,b and c:";
                   cin>>a>>b>>c;
                }
};
int largest(sample obj)
{
        if(obj.a>obj.b && obj.a>obj.c)
        return obj.a;
        else if(obj.b>obj.a && obj.b>obj.c)
        return obj.b;
        else
        return obj.c;
}
int main()
{
        sample s;
        s.input();
        cout<<largest(s);
        return 0;
}
//More than one friend functions for the same class
#include<iostream>
using namespace std;
class sample
{
        int a,b;
        public:
        friend int add(sample);
        friend int product(sample);
                void input()
                {
                   cout<<"\nEnter values of a,b:";
                   cin>>a>>b;
                }
};
int add(sample obj)
{
        return obj.a+obj.b;
}
int product(sample obj)
{
   return obj.a*obj.b;
}
int main()
{
```

```
    sample s;
    s.input();
    cout<<add(s);
    cout<<endl<<product(s);
    return 0;
}
```