

```

/* This C++ program illustrates the general concept of FRIEND Function*/

/* Whenever a class declared a function which is not the member
 * function as friend, then that function can access the private content
 * of class. */

/*NAME: Sagar Giri, Roll No. 205, Section: A*/

#include <iostream>
using namespace std;
class beta; //declare a class that we are going to use in the program
class alpha
{
    private:
        int a;
    public:
        alpha()
        {
            a = 5;
        }
        friend void sum(alpha, beta);
};
class beta
{
    private:
        int b;
    public:
        beta()
        {
            b = 10;
        }
        friend void sum(alpha, beta); //declaring friend function
};

//this friend function can access the private content of the class

void sum(alpha x, beta y)
{
    cout<<"sum ="<<x.a+y.b;
}

int main()
{
    alpha A;
    beta B;
    sum(A,B);
    return 0;
}

```

OUTPUT:

```

sum =15
-----
(program exited with code: 0)
Press return to continue

```

```

/* C++ Program to overload insertion and extraction operator
 * using friend function concept for multiple complex number.
 * */

/*Name: Sagar Giri, Roll No. 205, Section: A*/

#include <iostream>
using namespace std;
class complex
{
    private:
        int real , imag;
    public:
        complex()
        { real= imag = 0; }

        complex(int r, int i)
        {
            real = r;
            imag = i;
        }

        friend istream & operator>>(istream &, complex &);
        friend ostream & operator<<(ostream &, complex &);
};

istream & operator >>(istream & input, complex & cc1) //for extraction
{
    cout<<"Enter real and imaginary part of complex"<<endl;
    input>>cc1.real>>cc1.imag;
    return input;
}

ostream & operator <<(ostream & output, complex & cc1) //for insertion
{
    cout<<endl<<"The complex is:: ";
    cout<<cc1.real<<"+"<<cc1.imag<<"i";
    return output;
}

int main()
{
    complex c1, c2;
    cin>>c1>>c2;
    cout<<c1<<c2;
    return 0;
}

```

OUTPUT:

```

Enter real and imaginary part of complex
5 6
Enter real and imaginary part of complex
8 9

The complex is:: 5+6i
The complex is:: 8+9i

```

```
/*C++ PROGRAM TO DEMONSTRATE TO OVERLOAD THE RELATIONAL OPERATOR != */
/*NAME: SAGAR GIRI, ROLL: 205 , SEC: A*/
```

```
#include <iostream>
using namespace std;
class Distance
{
    private:
        float feet;
    public:
        Distance()
        { feet = 0.0; }
        Distance(float ft)
        {
            feet = ft;
        }
        bool operator != (Distance dd2)
        {
            float ddd1 = feet;
            float ddd2 = dd2.feet;
            if(ddd1 != ddd2)
                return (true);
            else
                return (false);
        }
};

int main()
{
    float data1,data2;
    cout<<"Enter two floating point data"<<endl;
    cin>>data1>>data2;
    Distance d1(data1), d2(data2);
    if(d1.operator != (d2))
        cout<<"Not equal"<<endl;
    else
        cout<<"Both Equal"<<endl;
    return 0;
}
```

OUTPUT:

```
Enter two floating point data
5.6 8.9
Not equal
-----
```

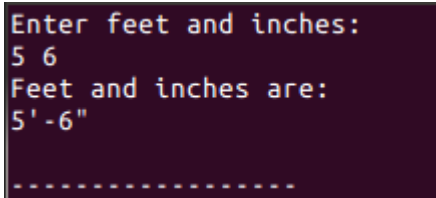
```
Enter two floating point data
5.5 5.5
Both Equal
-----
```

```
/* C++ Program to overload insertion and extraction operator using friend
* function for one distance at a time.
* */
```

```
/*Name: Sagar Giri, Roll No. 205, Section: A*/
```

```
#include <iostream>
using namespace std;
class Distance
{
    private:
        int feet;
        float inches;
    public:
        Distance()
        { feet = 0; inches= 0.0; }
        Distance (int ft, float in)
        {
            feet = ft;
            inches = in;
        }
        void display()
        {
            cout<<feet<<"' - "<<inches<<"\"";
        }
        friend void operator>>(istream & input, Distance & dd1);
        friend void operator<<(ostream & output, Distance & dd1);
};
void operator>>(istream & input, Distance & dd1)
{
    cout<<"Enter feet and inches:"<<endl;
    input>>dd1.feet>>dd1.inches;
}
void operator<<(ostream & output, Distance & dd1)
{
    cout<<"Feet and inches are:"<<endl;
    output<<dd1.feet<<"' - "<<dd1.inches<<"\"";
}
int main()
{
    Distance d1;
    cin>>d1;
    cout<<d1;
    return 0;
}
```

OUTPUT:



```
Enter feet and inches:
5 6
Feet and inches are:
5'-6"
.....
```

```

/*C++ program to illustrate the conversion between c strings and string objects
*/
/*Name: Sagar Giri, Roll No. 205, Section: A*/

#include<iostream>
#include<string.h>
using namespace std;

class String
{
    private:
        enum{SZ = 80};           //size of all String objects
        char str[SZ];           //holds a C-string
    public:
        String()                 //default constructor
        {   str[0]=' '; }

        String(char s[])         //one argument constructor
        {   strcpy(str,s); } //convert C-string to String

        void display()
        {   cout<<str; }

        operator char *()       //conversion operator
        {
            return str;         //convert String to C-string
        }
};

int main()
{
    String S1;                 //use default constructor
    char xstr[]="Hello\n"; //create and initialize C-string

    //use 1-arg constructor to convert C-string to String
    S1 = xstr;
    S1.display();

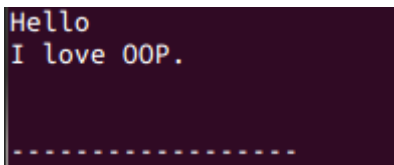
    //uses 1-arg constructor to initialize string
    String S2("I love OOP.");

    //use conversion operator to convert String to C-string
    cout<<static_cast<char *>(S2);

    cout<<endl;
    return 0;
}

```

OUTPUT:



```

Hello
I love OOP.

```

```

/* C++ Program to illustrate the concept of Basic to UserDefined
 * type-conversion using one argument constructor*/

/*Name :- Sagar Giri, Roll No. :205 , Section: A*/

#include <iostream>
using namespace std;

class Distance
{
    private :
        int feet; float inches;
    public :
        Distance()
        {
            feet = 0; inches = 0.0;
        }

        Distance(float mtr)
        {
            float fltfeet = mtr * 3.28;    //convert meter into feet

            //takes only stream of digit before decimal
            feet = (int)fltfeet; //explicit basic to basic Type Conversion

            // takes stream after decimal and converts them into inches
            inches = (fltfeet - feet) * 12.0;
        }

        void display()
        {
            cout<<feet<<"' - "<<inches<<"'";
        }
};

int main()
{
    Distance d1 = 5.3; // d1 calls one argument constructor
    cout << "Distance one =";
    d1.display();
    return 0;
}

```

OUTPUT:

```

Distance one =17'-4.60801'
-----
(program exited with code: 0)
Press return to continue

```