(FRIEND FUNCTIONS)

(MADE BY ALI AKBER)

(BSCS 2ND SS1)

//SINGLE FRIEND FUNCTION.

```
#include<iostream>

using namespace std;

class MyClass{

    private:

    int no;

    public:

    MyClass():no(0){}    //no arg constructor.

    MyClass(int n):no(n){}    //one arg constructor.

    friend void show(MyClass);

};

void show(MyClass obj){

    cout<<"No is:"<<obj.no<<endl;

}

int main(){

    MyClass obj1(100);

    show(obj1);

    return 0;

}
```

//ADDING OBJECTS BY TWO FRIEND FUNCTION.

```
#include<iostream>

using namespace std;

class YourClass;
class MyClass{
        private:
        int no;
        public:
        MyClass():no(0){}    //no arg constructor.
        MyClass(int n):no(n){}    //one arg constructor.
        friend void show(MyClass,YourClass);
};
class YourClass{
        private:
        int no;
        public:
        YourClass():no(0){}    //no arg constructor.
        YourClass(int n):no(n){}    //one arg constructor.
        friend void show(MyClass,YourClass);
};
void show(MyClass obj,YourClass obj2){
        cout<<"Sum is:"<<obj.no+obj2.no<<endl;
}
int main(){
        MyClass obj1(100);
        YourClass obj2(50);
        show(obj1,obj2);
        return 0;
}
```

//DISTANCE EXAMPLE USING FRIEND FUNCTION.

```
#include <iostream>

using namespace std;

class Distance

{

    private:

        int feets;

        float inches;

    public:

        /// constructors

        Distance(): feets(0), inches(0.0f)    {}

        Distance(int f, float inc): feets(f), inches(inc) {}

        Distance(float tfeets)            /// conversion constructor

        {

            feets = tfeets;

            inches = (tfeets -feets) *12.0f;

        }

        void Showdist ()

        {

            cout<<"Distance is "<<feets<<'\''<<inches<<'\"'<<endl;

        }


        /// + operator overloading

        ///

        friend Distance operator + (Distance, Distance);
        ~Distance(){}
};
Distance operator + (Distance dd1, Distance dd2)
{
    Distance temp;
    temp.feets =    dd1.feets + dd2.feets;
    temp.inches = dd1.inches + dd2.inches;

    while(temp.inches >= 12.0f)
    {
        temp.inches -= 12.0f;
        temp.feets++;
    }
    return temp;
}


int main()
{
    Distance d1(1, 1.1f), d2(2,2.2f), d3, d4, d5, d6;

    d1.Showdist ();
    d2.Showdist ();
    d3    =    d1 + d2;
    cout<<"\nResult of (d3    =    d1 + d2) "<<endl;
    d3.Showdist ();
    d4    =    d1 + 10.1f;

    cout<<"\nResult of (d4    =    d1 + 10.1f) "<<endl;

    d4.Showdist ();


    d5    =    10.1f + d2;

    cout<<"\nResult of (d5    =    10.1f + d2) "<<endl;

     d5.Showdist ();


    d6    =    10.1f + 20.1f;

    cout<<"\nResult of (d6    =    10.1f + 20.1f) "<<endl;

     d6.Showdist ();



    return 0;

}
```

//DISTANCE EXAMPLE USING FRIEND FUNCTION.

//WITH INSERTION AND EXTRACTION OPERATOR OVERLOADING.

```
#include <iostream>

using namespace std;


class Distance

{

    private:

        int feets;

        float inches;
    public:

        /// constructors

        Distance(): feets(0), inches(0.0f)    {}

        Distance(int f, float inc): feets(f), inches(inc) {}

        Distance(float tfeets)            /// conversion constructor

        {

                feets = tfeets;

                inches = (tfeets -feets) *12.0f;

        }


        /// + operator overloading

        ///

        friend Distance operator + (Distance, Distance);

        friend istream& operator >> (istream&, Distance&);

        friend ostream& operator << (ostream&, Distance&);


        ~Distance(){}

};

Distance operator + (Distance dd1, Distance dd2)

{

    Distance temp;

    temp.feets =    dd1.feets + dd2.feets;

    temp.inches = dd1.inches + dd2.inches;


    while(temp.inches >= 12.0f)

    {
```

```
        temp.inches -= 12.0f;

        temp.feets++;

    }

    return temp;

}

istream& operator >> (istream& in, Distance& dd)

{

    cout << "Enter feet: ";

    in >> dd.feets;

    cout << "Enter inches: ";

    in >> dd.inches;

    return in;

}

ostream& operator << (ostream& out, Distance& dd)

{

    out<<"Distance is "<<dd.feets<<'\''<<dd.inches<<'\"'<<endl;

    return out;

}


int main()

{

    Distance d1(1, 1.1f), d2, d3, d4, d5, d6;


    cin >> d2;

    cout << d1 << d2;
```

```
        d3    =    d1 + d2;

        cout<<"\nResult of (d3    =    d1 + d2) "<<endl;

        cout << d3;


        d4    =    d1 + 10.1f;

        cout<<"\nResult of (d4    =    d1 + 10.1f) "<<endl;

        cout << d4;


        d5    =    10.1f + d2;

        cout<<"\nResult of (d5    =    10.1f + d2) "<<endl;

        cout << d5;


        d6    =    10.1f + 20.1f;

        cout<<"\nResult of (d6    =    10.1f + 20.1f) "<<endl;

        cout << d6;



        return 0;

}
```

//EMPOLYEE EXAMPLE USING FRIEND FUNCTION.

//WITH INSERTION AND EXTRACTION OPERATOR OVERLOADING.

```
#include <iostream>

#include <string.h>


using namespace std;

const int SIZE = 100;
```

```
class Employee

{

        private:

                char name[SIZE];

                int id;

        public:

                Employee():id(0)

                {

                        strcpy(name,"");

                }

                Employee(int i,char na[]):id(i)

                {

                        strcpy(name,na);

                }


                friend istream& operator >> (istream&, Employee&);

                friend ostream& operator << (ostream&, Employee&);


                ~Employee() {}

};

int main()

{

        Employee e1, e2;


        //istream >> Employee
        cin >> e1    >> e2;

        cout<<endl;

        //ostream << Employee

        cout << e1 << e2;


        return 0;

}


istream& operator >>(istream& in, Employee& e)

{

        cout<<"Enter ID : ";

        in >> e.id;

        cout<<"Enter Name : ";

        in.ignore();

        in.getline(e.name,SIZE);

        return in;

}

ostream& operator << (ostream& out, Employee& e)

{

        out<<"ID is : "<<e.id<<endl;

        out<<"Name is : "<<e.name<<endl;

        return out;

}
```

//COPY CONSTRUCTOR.

//ASSIGNMENT OPERATOR OVERLOADING.

```
#include <iostream>
```

```
#include <string.h>


using namespace std;

const int SIZE = 100;


class Employee

{

	private:

		char name[SIZE];

		int id;

	public:

		Employee():id(0)

		{

			strcpy(name,"");

		}

		Employee(int i, char na[]):id(i)

		{

			strcpy(name,na);

		}

		Employee (Employee& e)

		{

			cout<<"Copy Constructor"<<endl;

			id = e.id;

			strcpy(name, e.name);

		}
```

```
        Employee& operator = (Employee& e)

        {

            cout<<"= operator"<<endl;

            id = e.id;

            strcpy(name, e.name);

            Employee obj(id, name);

            return obj;

            /// or use below statements with mentioned function header

            /// Employee& operator = (Employee& e)

            /// return *this;

        }


        friend istream& operator >> (istream&, Employee&);

        friend ostream& operator << (ostream&, Employee&);


        ~Employee() {}

};

int main()

{

    Employee e1, e2;

    cin >> e1;

    cout << e1;


    cout<<endl;


    e2 = e1;
    cout<<e2;


    cout<<endl;

    Employee e3(e2);

    cout<<e3;


    return 0;

}


istream& operator >>(istream& in, Employee& e)

{

    cout<<"Enter ID : ";

    in >> e.id;

    cout<<"Enter Name : ";

    in.ignore();

    in.getline(e.name,SIZE);

    return in;

}

ostream& operator << (ostream& out, Employee& e)

{

    out<<"ID is : "<<e.id<<endl;

    out<<"Name is : "<<e.name<<endl;

    return out;

}
```

/// THE THIS POINTER.

```
#include <iostream>
```

```
using namespace std;


class Myclass

{

        private:

                int no;

        public:

                Myclass() : no(0)

                {

                        cout<<"i am no argument constructor "<<this<<endl;

                }

                Myclass(int no)

                {

                        this->no = no;

                        cout<<"i am one argument constructor "<<this<<endl;

                }

                Myclass(Myclass& m)

                {

                        cout<<"Copy Constructor"<<endl;

                        this->no = m.no;

                }

                /// obj3 = obj4 = obj1;

                Myclass& operator = (Myclass& m)

                {

                        cout<<" = operator"<<endl;

                        this->no = m.no;
                        return *this;
                }
                void get()
                {
                        cout<<"Enter No ";
                        cin>>no;
                }
                void show()
                {
                        cout<<"No is "<<this->no<<endl;
                }
                ~Myclass()
                {
                        cout<<"i am destructor "<<this<<endl;
                }
};
int main()
{
        Myclass obj1(11);
        cout<<"In main() "<<&obj1<<endl;

        Myclass obj2(obj1);
        obj2.show();

        Myclass obj3, obj4;
        obj3 = obj4 = obj1;
        return 0;
}
```

// ---> MEMORY EFFICIENT STRING .

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

class StrCount
{
private:
    int count;
    char *str;
    friend class String;

public:
    StrCount(char *s)
    { //        1-Argument constructor.
        int length = strlen(s);
        str = new char[length + 1];
        strcpy(str, s);
        count = 1;
    }
    ~StrCount()
    {
        delete[] str; // Neccesary to delete the memory that is stored in heap .
    }
};

class String
{
private:
    StrCount *psc; // Pointer to StrCount .
public:
    String()
    { //    No - Argument .
        psc = new StrCount("NULL");
    }
    String(char *s)
    { // 1 - Argument .
        psc = new StrCount(s);
    }
    String(String &s)
    { // Copy Constructor .
        psc = s.psc;
        (psc->count)++;
    }
    ~String()
    {
        if (psc->count == 1)
            delete psc;
        else
```

```
                (psc->count)--;
        }
        void Display()
        {
            cout << psc->str;
            cout << "( Addr = " << psc << " ) , Count is = " << psc->count << endl;
        }
        void operator=(String &s)
        {
            if (psc->count == 1)
                delete psc;
            else
                (psc->count)--;
            psc = s.psc;
            (psc->count)++;
        }
};

int main()
{
        String S3(" WHEN THE FOXES PREACHES , LOOK AT YOUR GEESE "); // Obj of the friend Class .
        cout << " \n S3 : ";
        S3.Display();
        cout << "\n";
        String S1;
        cout << "\n S1 : ";
```

```
S1.Display();

cout << "\n";

S1 = S3; // Assign it another string .

cout << "\n S1 : ";

S1.Display();

cout << "\n\n S3 : ";

S3.Display();

cout << endl;

String S4;

S1 = S4;

cout << " \n S1 : ";

S1.Display();

cout << " \n\n S3 : ";

S3.Display();

cout << " \n\n S4 : ";

S4.Display();


String S2(S3);

cout << " \n\n S1 : ";

S1.Display();

cout << " \n\n S2 : ";

S2.Display();

cout << " \n\n S3 : ";

S3.Display();

cout << " \n\n S4 : ";

S4.Display();
```

```
        return 0;
}
```