Association

In this lesson, we'll learn about the relationship between two unrelated objects that is the association in C++.

WE'LL COVER THE FOLLOWING

- Example

In object-oriented programming, `association` is the relationship between the unrelated objects of the classes. Objects lifespan are not directly tied to each other like in **composition** and **aggregation**. To clearly understand the concept of the **association** we'll take an example of student and teacher class.

Example

The *teacher* has a relationship with his *students*, but it's not a *part-of* relationship as in composition. A teacher can add many students to a class, and a student can be registered in many courses. Neither of the objects(teacher and student) lifespan is tied to the other as any student can leave the class of specific teacher and similarly any teacher can change the course which he's currently teaching. Let's look at the implementation:

```
class Teacher;  // Making this friend of a student class

class Student
{
private:
        string Std_name;
        vector<Teacher *> tr;
        void addTeacher(Teacher * teach);

public:
        Student(string name) {
                Std_name = name;
        }

        string getName() const {
    return Std_name;
  }

  friend ostream& operator<<(ostream &out, const Student &std);
```

```
  friend ostream& operator<<(ostream &out, const Student &std);

  // Making teacher friend of this class to access addTeacher function
        friend class Teacher;
};

class Teacher {
private:
        string tr_name;
        vector<Student *> stdnt;

public:
        Teacher (string name) {
                tr_name = name;
        }

        void addStudent(Student *st)
        {
                // Teacher will add this student to course
                stdnt.push_back(st);

                // Student will also add this teacher for connection
                st->addTeacher(this);
        }

  friend ostream& operator<<(ostream &out, const Teacher &tchr)
        {
                int length = tchr.stdnt.size();
                if (length == 0)
                {
                        out << tchr.tr_name << " is not teaching any class";
                        return out;
                }

                out << tchr.tr_name << " is teaching: ";
                for (int count = 0; count < length; ++count)
                         out << tchr.stdnt[count]->getName() << ' ';

                return out;
        }

        string getName() const {
    return tr_name;
  }
};

void Student::addTeacher(Teacher *teach){
        tr.push_back(teach);
}

ostream& operator<<(ostream &out, const Student &std) {
        int length = std.tr.size();
        if (length == 0)
        {
                out << std.getName() << " is not registered in any course";
                return out;
        }

        out << std.Std_name << " is taught by: ";
        for (int count = 0; count < length; ++count)
                out << std.tr[count]->getName() << ' ';

        return out;
```

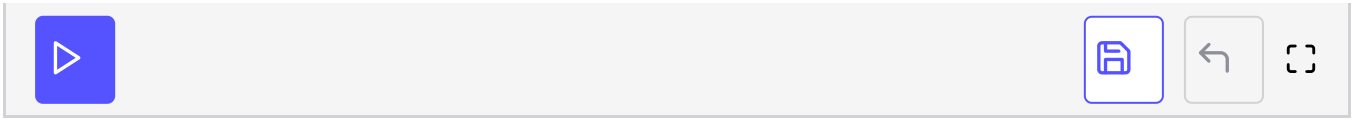
```
}

int main()
{
        // Creating a Students outside the scope of the Teacher
        Student *s1 = new Student("John");
        Student *s2 = new Student("Stacy");
        Student *s3 = new Student("Sarah");

        Teacher *t1 = new Teacher("Henry");
        Teacher *t2 = new Teacher("Neil");
  Teacher *t3 = new Teacher("Steve");

        t1->addStudent(s1);
        t2->addStudent(s1);
        t1->addStudent(s3);

        cout << *t1 << endl;
        cout << *t2 << endl;
  cout << *t3 << endl;
        cout << *s1 << endl;
        cout << *s2 << endl;
        cout << *s3 << endl;
}
```



In the above example, *student* and *teacher* objects are sharing a connection between each other. As clearly seen in the example that *teacher* and *student* can exist independently too.