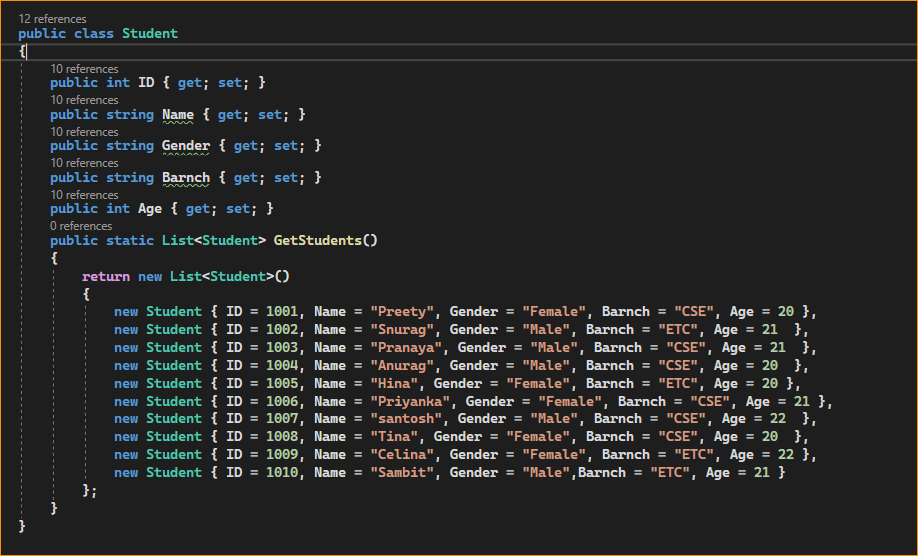
1. Why do we need to group the data based on Multiple Keys?

It is a common requirement in real-time applications to group the data based on multiple keys. In LINQ (Language-Integrated Query), you can use the GroupBy method to group elements in a collection based on one or more keys. When grouping by multiple keys, you can create compound keys with multiple properties or expressions. Now, we will see how to group the data based on multiple keys.

LINQ GroupBy Method in C# with Examples using Multiple Keys

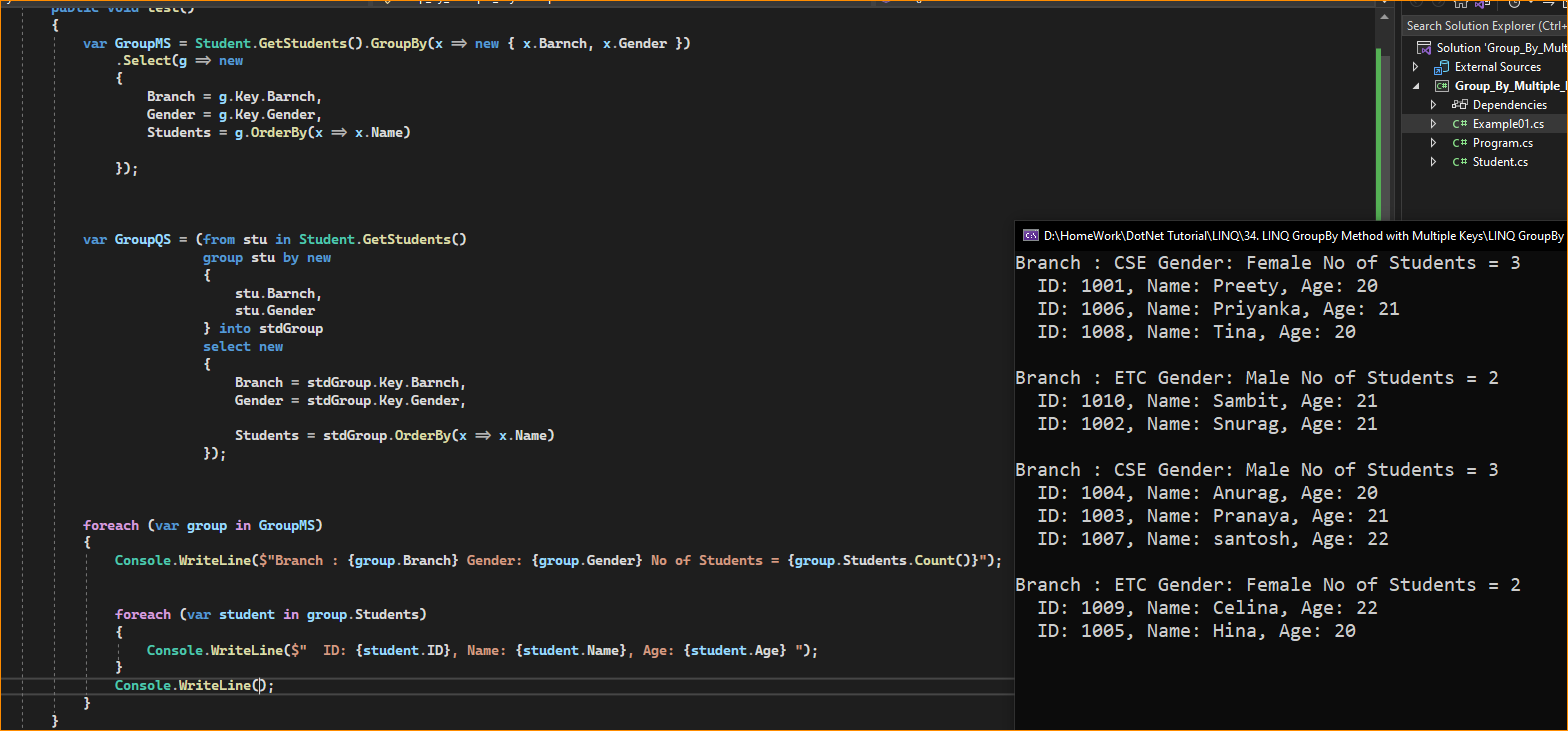
Let us understand how to use the LINQ GroupBy Method in C# with Multiple Keys. We will use the following Student class to understand the GroupBy Method with Multiple Keys. So, create a class file named Student.cs and copy and paste the following code. This is a very simple class having five properties such as ID, Name, Gender, Branch, and Age. This class also has one method called GetStudents(), which returns a list of all students, and this is going to be our data source.



1. Example to Understand Grouping Students Based on Multiple Keys in C#

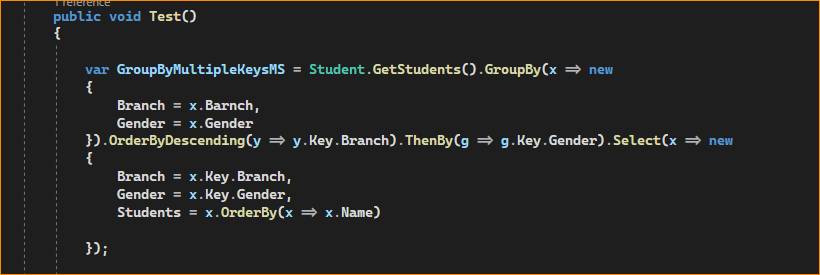
Now, our requirement is **to group the students based on Branch and gender**.

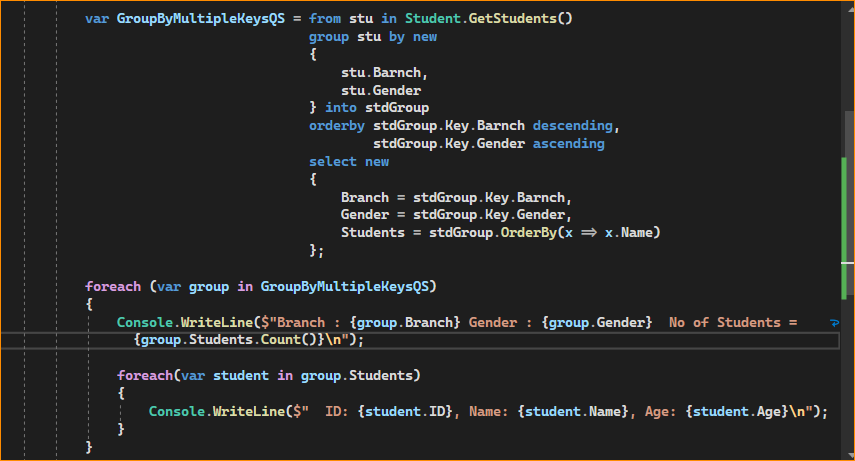
First, we need to group the students by Branch, and then we need to group the students by Gender. Also, we need to sort the students in each group by Name in Ascending order. For a better understanding, please look at the following example, which exactly does the same thing. The following example code is self-explained, so please go through the comment lines.



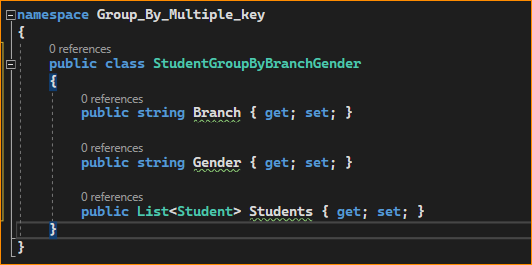
1. Grouping Students Based on the Branch and Gender along with OrderBy Method

Now, our requirement is to group the students based on Branch and gender. First, we need to group the students by Branch in Descending Order, and then we need to group the students by Gender in Ascending order on each branch group. Finally, the students in each group need to be sorted by their names in Ascending Order. For a better understanding, please look at the following example, which exactly does the same thing. The following example code is self-explained, so please go through the comment lines.

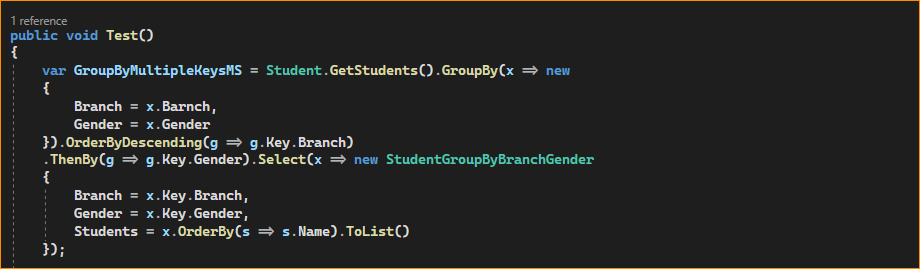


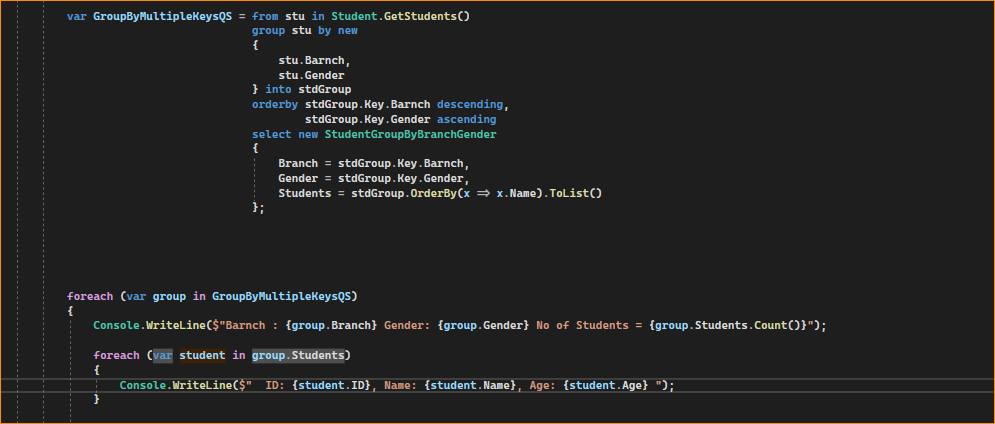


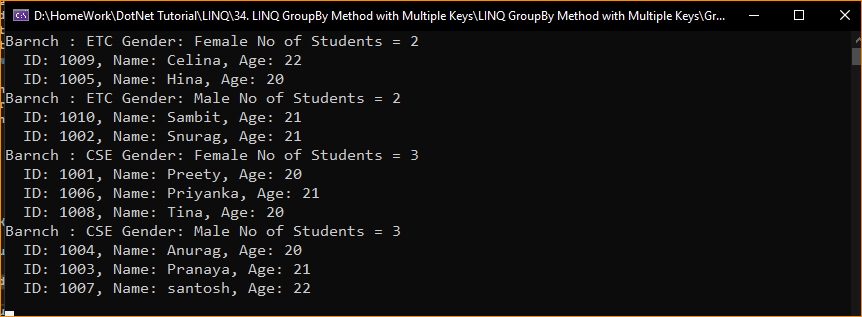
In the above example, we have projected the result to an anonymous type. If you want, you can also create a new type with the required properties and then project the result to that new type. Let us understand this. First, create a class file named StudentGroupByBranchGender.cs, and copy and paste the following code.



With the above changes in place, now modify the Main method of the Program class as follows. Here, you can see we are projecting the result into the newly created StudentGroupByBranchGender type instead of the anonymous type. The following example code is self-explained, so please go through the comment lines for a better understanding.







1. When to Use the LINQ GroupBy Method with Multiple Keys?

The GroupBy method in LINQ is a versatile tool for organizing data into groups based on some shared attributes. Here are scenarios when you might use GroupBy with multiple keys:

* **Complex Grouping Criteria**: When the logic for grouping elements is based on a combination of properties rather than a single attribute. For example, if you have a collection of orders and you want to group them by both CustomerId and OrderDate to see the orders from the same customer on the same date.
* **Hierarchical Data Representation**: To represent hierarchical data structures, such as categorizing products by both category and subcategory, which allows you to create a nested group structure for better data organization.
* **Multi-Dimensional Data Analysis**: In cases where you need a multi-dimensional analysis of data, similar to a pivot table in spreadsheets. For example, grouping sales data by Region and Quarter to understand sales trends.
* **Composite Key Relationships**: When you are dealing with data that has a composite key, meaning the uniqueness of an entry is determined by the combination of two or more fields, such as a composite primary key in a database table.
* **Detailed Aggregations**: If you need to perform detailed aggregations and operations on grouped data, such as counting, summing, or averaging, based on multiple criteria. This is useful in creating detailed statistical reports.
* **Data Segmentation**: For segmenting data into more granular buckets to perform operations like batch processing or detailed analysis on each segment individually.
* **Complex Data Shaping**: When preparing data for display in a UI, you want to group and structure data in a way that matches the UI components, like grids or lists that show grouped data.
* **Reducing Data Redundancy**: To eliminate redundant data in a collection and to organize it in a way that is easier to manage and display, especially when the same combinations of keys occur frequently.
* **Preparing for Joins**: Grouping by multiple keys can be a precursor to a join-like operation where you’re setting up your data to be aligned with another set of data on more than one matching field.
* **Data Summarization**: For summarizing complex data sets where you want to provide a summary based on a unique combination of fields (e.g., summarizing orders by customer and state).

What is LINQ ToLookup Method in C#?

The LINQ ToLookup Method in C# exactly does the same thing as done by the GroupBy Method. The only difference between these two methods is the GroupBy method uses Deferred Execution whereas the ToLookup method uses Immediate Execution. Before proceeding further, let us first understand what Deferred Execution and Immediate Execution are in LINQ.

What is LINQ Deferred Execution?

In the case of Deferred Execution, the LINQ Query is not executed at the point of its declaration. That means, when we write a LINQ query, it doesn’t execute by itself. It executes only when we access the query results. So, here the execution of the query is deferred until the query variable is iterated over using for-each loop. For example, Select, SelectMany, Where, Take, Skip, etc. belongs to the Deferred or Lazy Operators Category.

1. What is LINQ Immediate Execution?

In the case of Immediate Execution, the LINQ query is executed at the point of its declaration. So, it forces the query to execute and gets the result immediately. For example. Count, Average, Min, Max, First, Last, ToArray, ToList, etc. belongs to the Immediate or Greedy Operators category. Please read the following article to understand what Deferred and Immediate Execution are in Linq Queries in detail with examples.

Deferred Execution VS Immediate Execution in C#.

1. LINQ ToLookup Method in C# with Examples

Let us understand how to use the LINQ ToLookup Method in C# using both Method and Query Syntax with Examples. We are going to use the following Student class to understand the ToLookup Method. So, create a class file with the name Student.cs and then copy and paste the following code into it. The following Student class contains five properties such as ID, Name, Gender, Branch, and Age. This class also has one method called GetStudents() which returns a list of all students and this is going to be our data source.

