|  |
| --- |
| 윈도우즈 프로그래밍 |

-과제-

14주차

LINQ를 사용한 문제 해결

20144638

이한범

# 문제 1번

int[] numbers = {1,3,4,6,5,9,8,12,15,18,17,11,22}; 이 배열에서 10보다 작은 데이터만 뽑아내라. 10과 같거나 그보다 큰 데이터들은 걸러낼 것.

- foreach 문을 통하여 결과를 출력할 것

1. 작성한 코드내용

#Program class (Main)

|  |
| --- |
| namespace test  {  class Program  {  static void Main(string[] args)  {  int[] number = { 1, 3, 4, 6, 5, 9, 8, 12, 15, 18, 17, 11, 22 };  var numQuery =  from num in number  where (num < 10)  select num;  foreach (int num in numQuery)  Console.Write(num);  }  }  } |

▲ LINQ 사용. 쿼리문을 이용하여 number 배열내의 값을 where을 통해 10 미만인 값들만 추려내 주었다.

2) 실행결과

|  |
| --- |
|  |

# 문제 2번

1번에서 데이터 정렬을 추가할 것. 오름차순 혹은 내림차순으로 정렬된 결과를 콘솔창에 보여라.

- orderby를 사용하여 정렬할 것

1. 작성한 코드내용

# Program class (Main)

|  |
| --- |
| namespace test  {  class Program  {  static void Main(string[] args)  {  int[] number = { 1, 3, 4, 6, 5, 9, 8, 12, 15, 18, 17, 11, 22 };  var numQuery =  from num in number  where (num < 10)  //2번  //orderby num ascending : 오름차순  orderby num descending  select num;  foreach (int num in numQuery)  Console.Write(num);  }  }  } |

▲ 1번문제에서 order by 문을 추가시켜주었다. 디폴트 값은 오름차순이며, descending 옵션을 주게되면 내림차순으로 출력된다.

2) 실행결과

|  |
| --- |
|  |

# 문제 3번

int[] numbers = {1,3,4,6,5,9,8,12,15,18,17,11,22}; 이 배열에서 10 미만이며 짝수인 요소만 뽑아낸 결과를 콘솔창에 보여라.

- where을 통해 조건을 작성하고, and 키워드를 사용할 것.

1. 작성한 코드내용

# Program class (Main)

|  |
| --- |
| namespace test  {  class Program  {  static void Main(string[] args)  {  int[] number = { 1, 3, 4, 6, 5, 9, 8, 12, 15, 18, 17, 11, 22 };  var numQuery =  from num in number  //3번  where (num < 10 && (num % 2) == 0)  //2번  orderby num descending  select num;  foreach (int num in numQuery)  Console.Write(num);  }  }  } |

▲ 2번 코드에서 where 조건을 추가시켜주었다. 조건문에서 쓰듯이 and를 추가하여 짝수인 것들만 (2로 나눴을 때 나머지가 없는 것들) 추려내었다.

2) 실행결과

|  |
| --- |
|  |

# 문제 4번

리스트인 listStudent의 요소를 group 절의 분류 기준을 통해 분류하여 그룹화 한다. 여기서 분류 기준은 요소의 Average가 80 미만인지 이상인지에 따라 분류하여 두개의 그룹으로 나누어 출력한다.

- group by를 사용할 것, Student 클래스 정의

1. 작성한 코드내용

# Program class (Main)

|  |
| --- |
| namespace test  {  class Program  {  static void Main(string[] args)  {  List<Student> listStudent = new List<Student>  {  new Student() { Name = "김철수", Average = 78.5 },  new Student() { Name = "김영희", Average = 91.2 },  new Student() { Name = "홍길동", Average = 77.3 },  new Student() { Name = "김길수", Average = 80.8 }  };  var stuQuery =  from s in listStudent  group s by s.Average < 80.0 into g  select new { averCheck = g.Key, data = g };  //혹은 3항 연산자를 g.Key인 녀석에게 걸어줘서 판별도 ㅇㅋ  foreach (var s in stuQuery)  {  if (s.averCheck)  {  Console.WriteLine("평균 80점 미만:");  foreach (var a in s.data)  Console.WriteLine(a.Name + " : " + a.Average + "점");  }  else  {  Console.WriteLine("평균 80점 이상:");  foreach (var a in s.data)  Console.WriteLine(a.Name + " : " + a.Average + "점");  }  }  } |

▲ Student 객체를 가지는 List, listStudent를 생성. 쿼리문을 통하여 리스트 내 student 객체들의 average 값이 80 미만인지를 비교.

▲ group by를 사용하여 해당 조건이 참일 때, 거짓일 때로 분류. g.Key는 true, false 값이며 data라는 변수에 저장한 g는 g.Key의 참, 거짓 값에 따른 Student 객체를 가진다.

# Student class

|  |
| --- |
| class Student  {  public String Name { get; set; }  public double Average { get; set; }  } |

▲ Student 객체가 필요한 프로퍼티 정의

2) 실행결과

|  |
| --- |
|  |

# 문제 5번

내부 조인과 외부 조인을 이용하여 예상출력결과에 맞추어 출력할 것.

- join을 사용하여 내부조인, 외부조인을 구현할 것.

-필드중 Name이 겹침을 확인할 것.

1. 작성한 코드내용

# Program class (Main)

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  List<MyAverage> listAverage = new List<MyAverage> {  new MyAverage() { Name = "김철수", Average = 78.5 },  new MyAverage() { Name = "김영희", Average = 91.2 },  new MyAverage() { Name = "홍길동", Average = 77.3 },  new MyAverage() { Name = "김길수", Average = 80.8 },  new MyAverage() { Name = "김영순", Average = 54.2 },  new MyAverage() { Name = "김상수", Average = 90.8 },  new MyAverage() { Name = "이한수", Average = 61.4}  };  List<MyHobby> listHobby = new List<MyHobby> {  new MyHobby() { Name = "김영순", Hobby = "자전거 타기" },  new MyHobby() { Name = "홍길동", Hobby = "컴퓨터 게임" },  new MyHobby() { Name = "이한수", Hobby = "피아노 연주" },  new MyHobby() { Name = "김철수", Hobby = "축구" }  };  //내부조인  var innerJoin =  from aver in listAverage  join hobby in listHobby on aver.Name equals hobby.Name  select new  {  Name = aver.Name,  Average = aver.Average,  Hobby = hobby.Hobby  };  //외부조인  var outJoin =  from aver in listAverage  join hobby in listHobby on aver.Name equals hobby.Name into result  from hobby in result.DefaultIfEmpty(new MyHobby() { Hobby = "없음" })  select new  {  Name = aver.Name,  Average = aver.Average,  Hobby = hobby.Hobby  };  Console.WriteLine("<내부조인의 예상 출력결과>");  foreach (var s in innerJoin)  {  Console.WriteLine("이름 : {0}", s.Name);  Console.WriteLine(" - 평균 : {0}", s.Average);  Console.WriteLine(" - 취미 : {0}", s.Hobby);  }  Console.WriteLine("<외부조인의 예상 출력결과>");  foreach (var s in outJoin)  {  Console.WriteLine("이름 : {0}", s.Name);  Console.WriteLine(" - 평균 : {0}", s.Average);  Console.WriteLine(" - 취미 : {0}", s.Hobby);  }  }  } |

▲ MyAverage 객체를 가지는 List, listAverage와 MyHobby 객체를 가지는 List, listHobby 리스트 컬렉션 생성.

▲ 내부조인의 경우 교집합 연산과 유사하다. join 키워드를 사용하며 MyAverage와 MyHobby 객체가 동일하게 가지는 Name 필드를 합쳐 사용한다. select에서 innerJoin 변수가 Name, Average, Hobby 필드를 가지도록 쿼리문 작성.

▲ 외부조인의 경우 합집합 연산과 유사하나, 이름이 겹치더라도 합쳐지는 것이 아니라 별개의 요소로 나눈다. 따라서 동일한 join 문에 into를 사용하여 join한 결과를 result에 저장한다. 이 후, hobby가 없는 이름은 Hobby 요소가 없다고 출력하도록 한다.

# MyAverage class

|  |
| --- |
| class MyAverage  {  public String Name { get; set; }  public double Average { get; set; }  } |

▲ MyAverage 객체가 필요한 프로퍼티 정의

# MyHobby class

|  |
| --- |
| class MyHobby  {  public String Name { get; set; }  public String Hobby { get; set; }  } |

▲ MyHobby 객체가 필요한 프로퍼티 정의

2) 실행결과

|  |
| --- |
|  |