# 객체지향 프로그래밍

## C# - example

KYUNGSUNG UNIVERSITY SINCE 1955

일반화 메소드(제네릭 메소드)

```
using System;

namespace A093_GenericMethod
{
  class Program
  {
    static void Main(string[] args)
    {
      int[] a = { 1, 2, 3 };
      double[] d = { 0.1, 0.2, 0.3 };
      string[] s = { "tiger", "lion", "zebra" };

      PrintArray<int>(a);
      PrintArray<double>(d);
      PrintArray<string>(s);
    }

    private static void PrintArray<T>(T[] a)
    {
      foreach (var item in a)
        Console.Write("{0,8}", item);
      Console.WriteLine();
    }
  }
}
```

# 일반화 클래스(제네릭 클래스)

```
using System;

namespace A094_GenericClass
{
  class MyClass<T>
  {
    private T[] arr;
    private int count = 0;

    public MyClass(int length)
    {
      arr = new T[length];
      count = length;
    }

    public void Insert(params T[] args)
    {
      for (int i = 0; i < args.Length; i++)
        arr[i] = args[i];
    }

    public void Print()
    {
      foreach (T i in arr)
        Console.Write(i + " ");
      Console.WriteLine();
    }
```

```
    public T AddAll()
    {
      T sum = default(T);
      foreach (T item in arr)
        sum = sum + (dynamic)item;
      return sum;
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      MyClass<int> a = new MyClass<int>(10);
      MyClass<string> s = new MyClass<string>(5);

      a.Insert(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      s.Insert("Tiger", "Lion", "Zebra", "Monkey", "Cow");

      a.Print();
      s.Print();

      Console.WriteLine("a.AddAll() : " + a.AddAll());
      Console.WriteLine("s.AddAll() : " + s.AddAll());
    }
  }
}
```

# dynamic 형을 사용하는 일반화 프로그램

```
using System;

namespace A095_GenericMethodsUsingDynamic
{
  class Program
  {
    static void Main(string[] args)
    {
      int[] a = { 10, 45, 32, 47, 85, 46, 93, 47, 50, 71 };
      double[] d = { 0.1, 5.3, 6.7, 8.5, 4.9, 6.1 };
      float[] f = { 1.2f, 5.3f, 7.8f, 6.1f, 3.4f, 8.8f };
      decimal[] c = { 123, 783, 456, 234, 456, 748 };

      PrintArray<int>("a[] :", a);
      CalcArray<int>(a);
      PrintArray<double>("d[] :", d);
      CalcArray<double>(d);
      PrintArray<float>("f[] :", f);
      CalcArray<float>(f);
      PrintArray<decimal>("c[] :", c);
      CalcArray<decimal>(c);
    }
```

<T>로 정의된 값들은 더하거나 비교하는 부분에서 컴파일 시에 에러 메시지가 나온다. 왜냐하면 <T>는 사용자가 만든 클래스를 포함해서 어떠한 자료형도 올 수 있는데, 이 자료들이 더하거나 비교할 수 있는 데이터인지 알 수 없기 때문이다.

이 문제를 dynamic 키워드로 처리할 수 있다. dynamic 형은 형식 검사를 컴파일 시에 하지 않고 실행할 때 한다.
실행할 때는 <T>에 대치되는 int, double 등의 형식 사용되고 이런 데이터들은 더하거나 비교할 수 있기 때문에 에러가 발생하지 않는다.

저장하는 데이터의 형이 int, double, float, decimal 등 어떤 숫자형 배열이라도 적용할 수 있는 메소드를 dynamic과 일반화 프로그램으로 작성한다.

```
    private static void CalcArray<T>(T[] a) where T : struct
    {
      T sum = default(T);
      T avg = default(T);
      T max = default(T);

      foreach (dynamic item in a)
      {
        if (max < item)
          max = item;
        sum += item;
      }
      avg = (dynamic)sum / a.Length;
      Console.WriteLine("       Sum = {0}, Average = {1}, Max = {2}", sum, avg, max);
    }

    private static void PrintArray<T>(string s, T[] a) where T : struct
    {
      Console.Write(s);
      foreach (var item in a)
      {
        Console.Write(" {0}", item);
      }
      Console.WriteLine();
    }
  }
}
```

LinkedList.cs

```
using System;

namespace A096_LinkedList
{
  class Node
  {
    internal int data;
    internal Node next;
    public Node(int data)
    {
      this.data = data;
      next = null;
    }
  }

  class LinkedList
  {
    Node head;

    internal void InsertFront(int data)
    {
      Node node = new Node(data);
      node.next = head;
      head = node;
    }
```

```
    internal void InsertLast(int data)
    {
      Node node = new Node(data);
      if (head == null)
      {
        head = node;
        return;
      }
      Node lastNode = GetLastNode();
      lastNode.next = node;
    }

    internal Node GetLastNode()
    {
      Node temp = head;
      while (temp.next != null)
      {
        temp = temp.next;
      }
      return temp;
    }
```

LinkedList.cs

```
// prev 뒤에 data를 갖는 노드를 삽입하기
internal void InsertAfter(int prev, int data)
{
  Node prevNode = null;

  // find prev
  for (Node temp = head; temp != null; temp = temp.next)
    if (temp.data == prev)
      prevNode = temp;

  if (prevNode == null)
  {
    Console.WriteLine("{0} data is not in the list");
    return;
  }
  Node node = new Node(data);
  node.next = prevNode.next;
  prevNode.next = node;
}
```

```
// key 값을 저장하고 있는 노드를 삭제하기
internal void DeleteNode(int key)
{
  Node temp = head;
  Node prev = null;
  if (temp != null && temp.data == key) // head가 찾는 값이면
  {
    head = temp.next;
    return;
  }
  while (temp != null && temp.data != key)
  {
    prev = temp;
    temp = temp.next;
  }
  if (temp == null) // 끝까지 찾는 값이 없으면
  {
    return;
  }
  prev.next = temp.next;
}
```

LinkedList.cs

```
    internal void Reverse()
    {
      Node prev = null;
      Node current = head;
      Node temp = null;
      while (current != null)
      {
        temp = current.next;
        current.next = prev;
        prev = current;
        current = temp;
      }
      head = prev;
    }

    internal void Print()
    {
      for (Node node = head; node != null; node = node.next)
        Console.Write(node.data + " -> ");
      Console.WriteLine();
    }
  }
}
```

program.cs

```
using System;

namespace A096_LinkedList
{
  class Program
  {
    static void Main(string[] args)
    {
      LinkedList list = new LinkedList();
      Random r = new Random();

      for (int i = 0; i < 5; i++)
        list.InsertLast(r.Next(100));

      Console.WriteLine("랜덤한 5개 값의 리스트입니다");
      list.Print();

      Console.Write("₩n리스트의 맨 앞에 10, 맨 뒤에 90을 삽입합니다. <Enter>를 입력하세요");
      Console.ReadLine();
      list.InsertFront(10);
      list.InsertLast(90);
      list.Print();

      Console.WriteLine("₩nx 노드 뒤에 y값을 저장하려고 합니다.");
      Console.Write(" x값을 입력하세요: ");
      int x = int.Parse(Console.ReadLine());
      Console.Write(" y값을 입력하세요: ");
      int y = int.Parse(Console.ReadLine());
```

```
            list.InsertAfter(x, y);
            list.Print();

            Console.Write("₩n삭제할 노드의 값을 입력하세요: ");
            int z = int.Parse(Console.ReadLine());
            list.DeleteNode(z);
            list.Print();

            Console.WriteLine("₩n리스트를 뒤집어서 출력합니다. <Enter>를 입력하세요");
            Console.ReadLine();
            list.Reverse();
            list.Print();
        }
    }
}
```

LinkedList 클래스를 활용한 프로그램

```
using System;

namespace A097_UsingLinkedList
{
  class Program
  {
    static void Main(string[] args)
    {
      LinkedList list = new LinkedList();
      Random r = new Random();

      for (int i = 0; i < 5; i++)
        list.InsertLast(r.Next(100));

      Console.WriteLine("랜덤한 5개 값의 리스트입니다");
      list.Print();

      Console.Write("₩n리스트의 맨 앞에 10, 맨 뒤에 90을 삽입합니다.
<Enter>를 입력하세요");
      Console.ReadLine();
      list.InsertFront(10);
      list.InsertLast(90);
      list.Print();

      Console.WriteLine("₩nx 노드 뒤에 y값을 저장하려고 합니다.");
      Console.Write(" x값을 입력하세요: ");
      int x = int.Parse(Console.ReadLine());
      Console.Write(" y값을 입력하세요: ");
      int y = int.Parse(Console.ReadLine());
```

```
            list.InsertAfter(x, y);
            list.Print();

            Console.Write("₩n삭제할 노드의 값을 입력하세요: ");
            int z = int.Parse(Console.ReadLine());
            list.DeleteNode(z);
            list.Print();

            Console.WriteLine("₩n리스트를 뒤집어서 출력합니다. <Enter>를 입력하세요");
            Console.ReadLine();
            list.Reverse();
            list.Print();
        }
    }
}
```

stack.cs

```
using System;

namespace A098_StackImplementation
{
  class MyStack<T>
  {
    const int maxSize = 10;
    private T[] arr = new T[maxSize];
    private int top;

    public MyStack()
    {
      top = 0;
    }
    public void Push(T val)
    {
      if (top < maxSize)
      {
        arr[top] = val;
        ++top;
      }
      else
      {
        Console.WriteLine("Stack Full");
        return;
      }
    }
    public T Pop()
    {
      if (top > 0)
      {
        --top;
        return arr[top];
      }
      else
      {
        Console.WriteLine("Stack Empty");
        return default(T);
      }
    }
  }
}
```

program.cs

```
using System;

namespace A098_StackImplementation
{
  class Program
  {
    static void Main(string[] args)
    {
      MyStack<int> stack = new MyStack<int>();
      Random r = new Random();

      for (int i = 0; i < 10; i++)
      {
        int val = r.Next(100);
        stack.Push(val);
        Console.WriteLine("Push(" + val + ") ");
        //Console.WriteLine(" top = " + stack.top);
      }

      Console.WriteLine();
      for (int i = 0; i < 10; i++)
      {
        Console.WriteLine("Pop() = " + stack.Pop());
      }
    }
  }
}
```

Myqueue.cs

```
using System;

namespace A100_QueueImplementation
{
  class Node<T>
  {
    internal T value;
    internal Node<T> next;

    public Node (T value)
    {
      this.value = value;
      this.next = null;
    }
  }
```

```
class MyQueue<T>
{
  internal Node<T> first = null;
  internal Node<T> last = null;

  internal void EnQueue(Node<T> node)
  {
    if (last == null)
      first = last = node;
    else
    {
      last.next = node;
      last = node;
    }
  }
```

Myqueue.cs

```
    internal T DeQueue()
    {
      if (first == null)
      {
        Console.WriteLine("Queue Empty");
        return default(T);
      }
      else
      {
        T value = first.value;
        first = first.next;
        return value;
      }
    }

    internal void Print()
    {
      for (Node<T> t = first; t != null; t = t.next)
        Console.Write(t.value + " -> ");
      Console.WriteLine();
    }
  }
}
```

program.cs

```
using System;

namespace A100_QueueImplementation
{
  class Program
  {
    static void Main(string[] args)
    {
      Random r = new Random();
      MyQueue<float> que = new MyQueue<float>();
      for (int i = 0; i < 5; i++)
        que.EnQueue(new Node<float>(r.Next(100)/100.0F));

      que.Print();

      for (int i = 0; i < 3; i++)
        Console.WriteLine("DeQueue: {0}", que.DeQueue());

      que.Print();

    }
  }
}
```

컬렉션이란 같은 형의 데이터를 모아서 처리하는 자료구조이다. 배열도 컬렉션의 하나이다. 켈력션에는 "제네릭 컬렉션 " 과 "제네릭이 아닌 컬렉션 " 의 두 가지 유형이 있다. .NET Framework 2.0에서 추가된 제네릭 컬렉션은 컴파일 타임에 형식이 안전한(type-safe) 컬렉션을 제공한다.

| 수행할 작업 | Generic collection | Non-generic collection |
|---|---|---|
| 키별로 빠르게 조회할 수 있도록 키/값 쌍으로 저장 | Dictionary<Tkey, Tvalue> | Hashtable |
| 인덱스별로 항목 액세스 | List<T> | Array<br>ArrayList |
| FIFO 방식으로 항목 사용 | Queue<T> | Queue |
| LIFO 방식으로 데이터 사용 | Stack<T> | Stack |
| 순서대로 항목 액세스 | LinkedList<T> | - |
| 컬렉션에 항목을 추가하거나 삭제할 때 알림 표시 | ObservableCollection<T> | - |
| 정렬된 컬렉션 | SortedList<Tkey, Tvalue> | SortedList |
| 수학 함수용 집합 | HashSet<T><br>SortedSet<T> | - |

```
using System;
using System.Collections;

namespace A102_ArrayList
{
  class Program
  {
    static void Main(string[] args)
    {
      ArrayList a = new ArrayList();
      Random r = new Random();

      PrintValues(a);

      for (int i = 0; i < 10; i++)
        a.Add(r.Next(100));

      PrintValues(a);
      a.Sort();
      PrintValues(a);

      a.RemoveAt(3);
      PrintValues(a);
    }
```

컬렉션 ArrayList의 사용

```
    private static void PrintValues(ArrayList a)
    {
      Console.WriteLine("Print Values in ArrayList");
      Console.WriteLine("   Count = {0}", a.Count);
      Console.WriteLine("   Capacity = {0}", a.Capacity);
      foreach (var i in a)
        Console.Write("   {0}", i);
      Console.WriteLine();
    }
  }
}
```

```
using System;
using System.Collections.Generic;

namespace A103_List
{
  class Program
  {
    static void Main(string[] args)
    {
      List<int> a = new List<int>();
      Random r = new Random();

      PrintValues(a);

      for (int i = 0; i < 10; i++)
        a.Add(r.Next(100));

      PrintValues(a);
      a.Sort();
      PrintValues(a);

      a.RemoveAt(3);
      PrintValues(a);
    }
```

List<T> 컬렉션

```
    private static void PrintValues(List<int> a)
    {
      Console.WriteLine("Print Values in List<int>");
      Console.WriteLine("  Count = {0}", a.Count);
      Console.WriteLine("  Capacity = {0}", a.Capacity);
      foreach (var i in a)
        Console.Write("  {0}", i);
      Console.WriteLine();
    }
  }
}
```

# List<T>와 배열의 정렬

```
using System;

namespace A104_ListAndArraySort
{
  class Program
  {
    static void Main(string[] args)
    {
      List<string> lstNames = new List<string>();
      lstNames.Add("dog");
      lstNames.Add("cow");
      lstNames.Add("rabbit");
      lstNames.Add("goat");
      lstNames.Add("sheep");
      lstNames.Sort();
      foreach (string s in lstNames)
        Console.Write(s + " ");
      Console.WriteLine();
```

```
      //Array arrNames = new Array(100);
      string[] arrNames = new string[100];
      arrNames[0] = "dog";
      arrNames[1] = "cow";
      arrNames[2] = "rabbit";
      arrNames[3] = "goat";
      arrNames[4] = "sheep";
      Array.Sort(arrNames);
      foreach (string s in lstNames)
        Console.Write(s + " ");
      Console.WriteLine();
    }
  }
}
```

배열을 내림차순으로 정렬하는 방법

```
using System;
using System.Collections;

namespace A105_IComparer
{
  // 내림차순 정렬
  public class ReverseComparer : IComparer
  {
    public int Compare(object x, object y)
    {
      string s1 = (string)x;
      string s2 = (string)y;
      return string.Compare(s2, s1);
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      string[] animalsEn = { "dog", "cow", "rabbit", "goat", "sheep", "mouse", "horse", "deer" };
      string[] animalsKo = { "개", "소", "토끼", "염소", "양", "쥐", "말", "사슴" };

      Display("초기 배열", animalsEn);
      Array.Sort(animalsEn);
      Array.Reverse(animalsEn);
      //Array.Sort(animalsEn, 2, 3);
      Display("Sort() 후 Reverse()", animalsEn);
```

```
      Display("초기 배열", animalsKo);
      Array.Sort(animalsKo, 2, 3);
      Display("[2]에서 3개 정렬 후", animalsKo);

      IComparer revComparer = new ReverseComparer();

      Array.Sort(animalsKo, revComparer);
      Display("내림차순 정렬", animalsKo);
    }

    private static void Display(string comment, string[] arr)
    {
      Console.WriteLine(comment);
      for (int i = arr.GetLowerBound(0); i <= arr.GetUpperBound(0); i++)
      {
        Console.Write("  {0}", arr[i]);
      }
      Console.WriteLine();
    }
  }
}
```

두 개의 배열을 쌍으로 정렬

```
using System;
using System.Collections;

namespace A092_3_SortArrayPair
{
  // 내림차순 정렬
  public class ReverseComparer : IComparer
  {
    public int Compare(object x, object y)
    {
      string s1 = (string)x;
      string s2 = (string)y;
      return string.Compare(s2, s1);
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      string[] animalsEn = { "dog", "cow", "rabbit", "goat", "sheep", "mouse"};
      string[] animalsKo = { "개", "소", "토끼", "염소", "양", "쥐" };

      Display("Before Sort", animalsEn, animalsKo);
      Array.Sort(animalsEn, animalsKo);
      Display("After Sort", animalsEn, animalsKo);
```

```
      Array.Sort(animalsKo, animalsEn);
      Display("After Sort by Korean", animalsEn, animalsKo);

      IComparer revCom = new ReverseComparer();
      Array.Sort(animalsEn, animalsKo, revCom);
      Display("After Descending Sort", animalsEn, animalsKo);
    }

    private static void Display(string comment, string[] arr1, string[] arr2)
    {
      Console.WriteLine(comment);
      for (int i = 0; i<arr1.Length ; i++)
      {
        Console.WriteLine("   [{0}] : {1,-8} {2,-8}", i, arr1[i], arr2[i]);
      }
      Console.WriteLine();
    }
  }
}
```

Icomparable 인터페이스를 이용한 객체의 정렬

```
using System;
using System.Collections.Generic;

namespace A107_IComparable
{
  class Program
  {
    static void Main(string[] args)
    {
      Artists[] famousArtists =
      {
        new Artists("레오나르도 다빈치", "이탈리아", 1452, 1519),
        new Artists("빈센트 반 고흐", "네덜란드", 1853,  1890),
        new Artists("클로드 모네",  "프랑스", 1840,  1926),
        new Artists("파블로 피카소",  "스페인", 1881,  1973),
        new Artists("베르메르",  "네덜란드", 1632,  1675),
        new Artists("르노아르",  "프랑스", 1841,  1919)
      };

      List<Artists> artists19C = new List<Artists>();
      foreach (var artist in famousArtists)
      {
        if (artist.Birth > 1800 && artist.Birth <= 1900)
          artists19C.Add(artist);
      }
```

```
    // IComparable을 사용하는 방법
    artists19C.Sort();
    Console.WriteLine("19세기 미술가를 탄생 순 정렬: IComparable");
    foreach (var a in artists19C)
      Console.WriteLine(a.ToString() );
  }
}

class Artists : IComparable
{
  public string Name { get; set; }
  public string Country { get; set; }
  public int Birth { get; set; }
  public int Die { get; set; }

  public Artists(string name, string country, int birth, int die)
  {
    Name = name;
    Country = country;
    Birth = birth;
    Die = die;
  }
```

```
    public int CompareTo(object obj)
    {
      Artists a = (Artists)obj;
      return this.Birth.CompareTo(a.Birth);
    }

    public override string ToString()
    {
      return string.Format(" {0}, {1}, {2}, {3}", Name, Country, Birth, Die);
    }
  }
}
```

Queue<T> 컬렉션의 사용 방법

```
using System;
using System.Collections.Generic;
using System.Collections;

namespace A108_Queue
{
  class Program
  {
    static void Main(string[] args)
    {
      Queue<string> que = new Queue<string>();
      que.Enqueue("Tiger");
      que.Enqueue("Lion");
      que.Enqueue("Zebra");
      que.Enqueue("Cow");
      que.Enqueue("Rabbit");
      PrintQueue("que: ", que);

      Console.WriteLine(" Dequeuing '{0}'", que.Dequeue());
      Console.WriteLine(" Peek: '{0}'", que.Peek());

      Queue<string> que2 = new Queue<string>(que.ToArray());
      PrintQueue("que2:", que2);
```

```
    string[] array = new string[que.Count];
    que.CopyTo(array, 0);
    Queue<string> que3 = new Queue<string>(array);
    PrintQueue("que3:", que3);

    Console.WriteLine("que.Contains(Lion) = {0}", que.Contains("Lion"));
    que3.Clear();
    Console.WriteLine("Count = {0}, {1}, {2}", que.Count, que2.Count, que3.Count);

    // 제너릭이 아닌 Queue
    Queue myQ = new Queue();
    myQ.Enqueue("one");
    myQ.Enqueue("two");
    myQ.Enqueue("three");

    // Displays the properties and values of the Queue.
    Console.WriteLine("myQ");
    Console.WriteLine("₩tCount:    {0}", myQ.Count);
    Console.Write("₩tValues:");
    PrintValues(myQ);
}
```

```
    private static void PrintQueue(string s, Queue<string> que)
    {
      Console.Write("{0,-8}", s);
      foreach(var item in que)
        Console.Write("{0,-8}", item);
      Console.WriteLine();
    }

    private static void PrintValues(Queue myQ)
    {
      foreach (string v in myQ)
        Console.Write("    {0}", v);
      Console.WriteLine();
    }

    public static void PrintValues(IEnumerable myCollection)
    {
      foreach (Object obj in myCollection)
        Console.Write("    {0}", obj);
      Console.WriteLine();
    }
  }
}
```

## Stack<T> 와 Polish 계산기

```
using System;
using System.Collections.Generic;

namespace A109_Stack
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("계산할 수식을 Polish 표기법으로 입력하세요: ");
      string[] token = Console.ReadLine().Split();

      foreach (var i in token)
        Console.Write(" {0}", i);
      Console.Write(" = ");
      Stack<double> nStack = new Stack<double>();
      foreach(var s in token)
      {
        if(isOperator(s))
        {
          switch (s)
          {
            case "+":
              nStack.Push(nStack.Pop() + nStack.Pop());
              break;
            case "-":
              nStack.Push(-(nStack.Pop() - nStack.Pop()));
              break;
            case "*":
              nStack.Push(nStack.Pop() * nStack.Pop());
              break;
            case "/":
              nStack.Push(1.0/(nStack.Pop() / nStack.Pop()));
              break;
          }
        }
        else
        {
          nStack.Push(double.Parse(s));
        }
      }

      Console.WriteLine("결과는 {0}", nStack.Pop());
    }
```

```
    private static bool isOperator(string s)
    {
      if (s == "+" || s == "-" || s == "*" || s == "/")
        return true;
      else
        return false;
    }

    private static bool isNumber(string s)
    {
      if (Double.TryParse(s, out double number))
        return true;
      else
        return false;
    }
  }
}
```

## Hashtable과 Dictionary<Tkey, Tvalue>

```
using System;
using System.Collections.Generic;

namespace A110_Dictionary
{
  class Program
  {
    static void Main(string[] args)
    {
      Dictionary<string, string> colorTable = new Dictionary<string, string>();

      colorTable.Add("Red", "빨간색");
      colorTable.Add("Green", "초록색");
      colorTable.Add("Blue", "파란색");

      foreach(var v in colorTable)
        Console.WriteLine("colorTable[{0}] = {1}", v.Key, v.Value);
      Console.WriteLine();
```

```
        try
        {
          colorTable.Add("Red", "빨강");
        }
        catch(ArgumentException e)
        {
          Console.WriteLine(e.Message);
        }

        try
        {
          Console.WriteLine("Yellow => {0}", colorTable["Yellow"]);
        }
        catch(KeyNotFoundException e)
        {
          Console.WriteLine(e.Message);
        }

        Console.WriteLine("₩n"+colorTable["Red"]);
        Console.WriteLine(colorTable["Green"]);
        Console.WriteLine(colorTable["Blue"]);
      }
    }
}
```

SortedList와 SortedList<Tkey, Tvalue>

```
using System;
using System.Collections.Generic;

namespace A111_SortedList
{
  class Program
  {
    static void Main(string[] args)
    {
      SortedList<int, string> s1 = new SortedList<int, string>();
      s1.Add(3, "Three");
      s1.Add(4, "Four");
      s1.Add(1, "One");
      s1.Add(2, "Two");

      for (int i = 0; i < s1.Count; i++)
        Console.Write("k: {0}, v: {1} / ", s1.Keys[i], s1.Values[i]);
      Console.WriteLine();
```

```
      foreach(var kvp in s1)
        Console.Write("{0, -10} ", kvp);
      Console.WriteLine();

      SortedList<string, int> s2 = new SortedList<string, int>();
      s2.Add("one", 1);
      s2.Add("two", 2);
      s2.Add("three", 3);
      s2.Add("four", 4);

      //Console.WriteLine(s2["one"]);
      Console.WriteLine(s2["two"]);

      foreach (var kvp in s2)
        Console.Write("{0, -10} ", kvp);
      Console.WriteLine();

      //for (int i = 0; i < s2.Count; i++)
      //  Console.Write("k: {0}, v: {1}", s2.Keys[i], s2.Values[i]);
```

```
        int val;

        if (s2.TryGetValue("ten", out val))
          Console.WriteLine("key: ten, value: {0}", val);
        else
          Console.WriteLine("[ten] : Key is not valid.");

        if (s2.TryGetValue("one", out val))
          Console.WriteLine("key: one, value: {0}", val);

        Console.WriteLine(s2.ContainsKey("one"));  // returns true
        Console.WriteLine(s2.ContainsKey("ten"));  // returns false

        Console.WriteLine(s2.ContainsValue(2)); // returns true
        Console.WriteLine(s2.ContainsValue(6)); // returns false

        s2.Remove("one");// 키가 'one' 요소 삭제
        s2.RemoveAt(0);  // 첫번째 요소 삭제

        foreach (KeyValuePair<string, int> kvp in s2)
          Console.Write("{0, -10} ", kvp);
        Console.WriteLine();
      }
    }
}
```

# Predicate<T> 델리게이트

```
using System;
using System.Collections.Generic;

namespace A118_Predicate
{
  class Program
  {
    static void Main(string[] args)
    {
      Predicate<int> isEven = n => n % 2 == 0;
      Console.WriteLine(isEven(6));

      Predicate<string> isLowerCase = s => s.Equals(s.ToLower());
      Console.WriteLine(isLowerCase("This is a lowercase string"));
    }

    static bool IsEven(int n) => n % 2 == 0;
    static bool IsLowerCase(string s) => s.Equals(s.ToLower());
  }
}
```

Predicate<T>는 Func나 Action과 같이 미리 정의된 델리게이트 형식이다. Predicate 델리게이트 메소드는 하나의 매개변수를 갖고 리턴 값이 bool인 델리게이트이다.

List<T>에서 Predicate<T> 델리게이트 사용

```
using System;
using System.Collections.Generic;

namespace A119_ListAndLambda
{
  class Program
  {
    static void Main(string[] args)
    {
        List<String> myList = new List<String> {
        "mouse", "cow", "tiger", "rabbit", "dragon", "snake"
      };

        bool n = myList.Exists(s => s.Contains("x"));
        Console.WriteLine("이름에 'x'를 포함하는 동물이 있나요 : " + n);

        Console.Write("이름이 3글자인 첫번째 동물 : ");
        string name = myList.Find(s => s.Length == 3);
        Console.WriteLine(name);

        Console.Write("6글자 이상의 동물들: ");
        List<string> longName = myList.FindAll(s => s.Length > 5);
        foreach (var item in longName)
        {
          Console.Write(item + " ");
        }
        Console.WriteLine();

        Console.Write("대문자로 변환: ");
        List<string> capList = myList.ConvertAll(s => s.ToUpper());
        foreach (var item in capList)
          Console.Write(item + " ");
        Console.WriteLine();
    }
  }
}
```

# LINQ의 기초

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace A120_LinqBasic
{
  class Program
  {
    // List에서 짝수 데이터를 꺼내어 정렬하여 출력한다
    static void Main(string[] args)
    {
      List<int> data = new List<int> { 123, 45, 12, 89, 456, 1, 4, 74, 46 };
      List<int> lstSortedEven = new List<int>();

      foreach (var item in data)
      {
        if (item % 2 == 0)
          lstSortedEven.Add(item);
      }
      lstSortedEven.Sort();
```

LINQ는 Language-Integrated Query의 약자로 C#에 통합된 데이터 질의 기능이다. SQL과 같은 Query 언어는 데이터 베이스에서 사용되던 언어인데, 이를 C#에 도입한 것이다. LINQ를 사용하면 컬렉션에서 데이터를 다루는 방법이 훨씬 간단해진다.

LINQ를 사용하려면 원본 데이터가 IEnumerable이나 IEnumerable<T> 인터페이스를 상속하는 형식이어야 한다. 배열과 리스트 등의 컬렉션들은 모두 여기에 해당된다.

List에서 짝수 데이터만을 꺼내서 정렬하여 출력하는 프로그램을 기존 방법과 LINQ를 사용하는 두 가지 방법으로 작성했다.

```
        Console.WriteLine("Using foreach: ");
        foreach (var item in lstSortedEven)
          Console.Write(item + " ");
        Console.WriteLine();

        IEnumerable<int> sortedEven = from item in data
                where item % 2 == 0
                orderby item
                select item;

        Console.WriteLine("₩nUsing Linq: ");
        foreach (var item in sortedEven)
          Console.Write(item + " ");
        Console.WriteLine();
    }
  }
}
```

# LINQ를 이용한 조건 검색과 정렬

1) data 리스트에서 20보다 큰 짝수를 내림차순으로 정렬하여 저장
2) 이 값을 2씩 곱하여 오름차순으로 정렬하여 저장

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace A121_LinqBasic2
{
  class Program
  {
    static void Main(string[] args)
    {
      List<int> data = new List<int> { 123, 45, 12, 89, 456, 1, 4, 74, 46 };

      Print("data : ", data);

      var lstEven = from item in data
                    where (item > 20 && item % 2 == 0)
                    orderby item descending
                    select item;

      Print("20보다 큰 짝수 검색 결과 : ", lstEven);

      var lstSorted = from item in lstEven
                      orderby item ascending
                      select item * 2;

      Print("2를 곱해서 오름차순 정렬 : ", lstSorted);
    }
```

```
    private static void Print(string s, IEnumerable<int> data)
    {
      Console.WriteLine(s);
      foreach (var i in data)
        Console.Write(" " + i);
      Console.WriteLine();
    }
  }
}
```

# LINQ의 결과를 리스트나 배열로 반환

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace A122_LinqToList
{
  class Program
  {
    static void Main(string[] args)
    {
      List<int> lstData = new List<int> { 123, 456, 132, 96, 13, 465, 321};
      Print("Data: ", lstData);

      List<int> lstOdd = new List<int>();
      lstOdd = SelectOddAndSort(lstData);
      Print("Ordered Odd: ", lstOdd);

      int[] arrEven;
      arrEven = SelectEvenAndSort(lstData);
      Print("Ordered Even: ", arrEven);
    }
```

LINQ의 결과는 IEnumerable<T>가 된다. LINQ의 결과를 리스트로 받고 싶으면 ToList<T>( ) 메소드를 사용한다. 이와 비슷하게 LINQ의 결과를 배열로 받고 싶으면 ToArray<T>( ) 메소드를 사용하면 된다.

리스트의 값 중에서 홀수를 찾아 정렬하여 리스트로 반환하고 짝수를 찾아 정렬하여 배열로 반환한다.

```
    private static List<int> SelectOddAndSort(List<int> lstData)
    {
      return (from item in lstData
              where item % 2 == 1
              orderby item
              select item).ToList<int>();
    }

    private static int[] SelectEvenAndSort(List<int> lstData)
    {
      return (from item in lstData
              where item % 2 == 0
              orderby item
              select item).ToArray<int>();
    }

    private static void Print(string s, IEnumerable<int> data)
    {
      Console.WriteLine(s);
      foreach (var i in data)
        Console.Write(" " + i);
      Console.WriteLine();
    }
  }
}
```

# 쿼리의 결과를 새로운 객체 컬렉션으로 저장하는 방법

이름, 학번, 성적이 포함된 Student 클래스를 사용하여 시험 점수와 평균, 그리고 각 시험에서 커트라인 이상의 점수를 받은 학생들을 출력한다

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace CSharp200
{
  class Student
  {
    public string Name { get; set; }
    public int Id { get; set; }
    public List<int> Scores { get; set; }
  }

  class Program
  {
    static List<Student> students;

    static void Main(string[] args)
    {
      students = new List<Student>
      {
        new Student {Name="PjKim", Id=19001001, Scores = new List<int>{86,90,76}},
        new Student {Name="BsKim", Id=19001002, Scores = new List<int>{56,92,93}},
        new Student {Name="YsCho", Id=19001003, Scores = new List<int>{69,85,75}},
        new Student {Name="BiKang", Id=19001004, Scores = new List<int>{88,80,57}},
      };
```

```
    Print(students);
    HighScore(0, 85);
    HighScore(1, 90);
    //HighScore(2, 90);
  }

  private static void HighScore(int exam, int cut)
  {
    var highScores = from student in students
                     where student.Scores[exam] >= cut
                     select new { Name = student.Name, Score = student.Scores[exam] };

    Console.WriteLine($"{exam + 1}번째 시험에서 {cut} 이상의 점수를 받은 학생");
    foreach (var item in highScores)
    {
      Console.WriteLine($"₩t{item.Name, -10}{item.Score}");
    }
  }
```

```
    private static void Print(List<Student> students)
    {
      foreach(var item in students)
      {
        Console.Write($"{item.Id, -10}{item.Name, -10}");
        foreach (var score in item.Scores)
        {
          Console.Write($"{score,-5}");
        }
        Console.WriteLine(item.Scores.Average().ToString("F2"));
      }
    }
  }
}
```

# LINQ, Group By로 데이터를 분류

Student 클래스를 사용하여 평균 점수가 80점 이상인 학생들과 80점 미만인 학생들로 그룹을 나누고 각 그룹별로 학생 수, 평균 점수, 최고 점수를 출력한다.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace A124_LinqGroupBy
{
  class Student
  {
    public string Name { get; set; }
    public int Id { get; set; }
    public List<int> Scores { get; set; }
  }

  class Program
  {
    static List<Student> students;

    static void Main(string[] args)
    {
      students = new List<Student>
      {
        new Student {Name="PjKim", Id=19001001, Scores = new List<int>{86,90,76}},
        new Student {Name="BsKim", Id=19001002, Scores = new List<int>{56,92,93}},
        new Student {Name="YsCho", Id=19001003, Scores = new List<int>{69,85,75}},
        new Student {Name="BiKang", Id=19001004, Scores = new List<int>{88,80,57}},
      };
```

```
    var result = from student in students
                 group student by student.Scores.Average() >= 80 into g
                 select new
                 {
                   key = g.Key == true ? "80점이상" : "80점미만",
                   count = g.Count(), //해당 구간의 학생수
                   avr = g.Average(student => student.Scores.Average()),
                   max = g.Max(student => student.Scores.Average())
                 };

       foreach (var item in result)
       {
         Console.WriteLine("{0} : 학생 수 = {1}", item.key, item.count);
         Console.WriteLine("{0} : 평균 점수 = {1:F2}", item.key, item.avr);
         Console.WriteLine("{0} : 최고 점수 = {1:F2}", item.key, item.max);
         Console.WriteLine();
       }
     }
   }
}
```

Reference


- ✓ C# 프로그래밍 입문, 오세만 외4, 생능출판
- ✓ 초보자를 위한 C# 200제, 강병익, 정보문화사
- ✓ 프랙티컬 C#, 이데이 히데유키, 김범준, 위키북스
- ✓ C#언어 프로그래밍 바이블, 김명렬 외1, 홍릉과학출판사
- ✓ C# and the .NET Platform, Andrew Troelsen, 장시혁, 사이텍미디어
- ✓ https://docs.microsoft.com/ko-kr/learn/browse/?products=dotnet&terms=c%23