HCI 프로그래밍

3. 프로그램 구성 요소와 데이터형 & 제어문과 예외처리

HCI
Human Computer Interaction

Identifier

식별자 (Identifier) 명명규칙

- 글자는 영문자, 밑줄(_)만 사용
- 숫자도 포함가능하나 첫 글자로 사용 못함
- 키워드는 사용 못함
- 대소문자구분 (Case sensitive)

Identifier

식별자 명명 권장사항

- 모두 대문자로 쓰는 것을 피함
- 밑줄(_)을 첫글자로 사용하지 않음
- 약어는 되도록 쓰지 않음
- 특수 문자의 사용을 피함

```
abstract → @abstract// 키워드가 아닌 식별자로 인식

Name, _Identifier, \u005fIdentfier   // 유니코드를 포함한 식별자

// 사용 불가능(숫자, 공백, 키워드, 특수기호 사용)

4int, code name, new, $abc
```

Keyword

- C#에서 특별한 의미가 있는 예약어 (Reserved Word)
- C# 키워드 (Keyword)는 lowercase를 사용함

abstract	as	base	bool	break	byte	case	catch
char	checked	class	const	continue	decimal	default	delegate
do	double	else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto	if	implicit
in	int	interface	internal	is	lock	long	namespace
new	null	object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte	sealed	short
sizeof	stackalloc	static	string	struct	switch	this	throw
true	try	typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatile	while			

Constant

변하지 않는 문자나 숫자값

정수형 상수

10진수, 16진수, 8진수로 표현되는 숫자 상수

실수형 상수

double, float

Boolean 상수

true, false

문자형 상수

유니코드, 작은따옴표(‘’) 안에 표현

문자열형 상수

큰따옴표(“”) 안에 표현
널(null)형 상수

널(null)형 상수

한번도 사용하지 않은 객체를 나타낼 때 사용

Operator and Separator

C#의 연산자

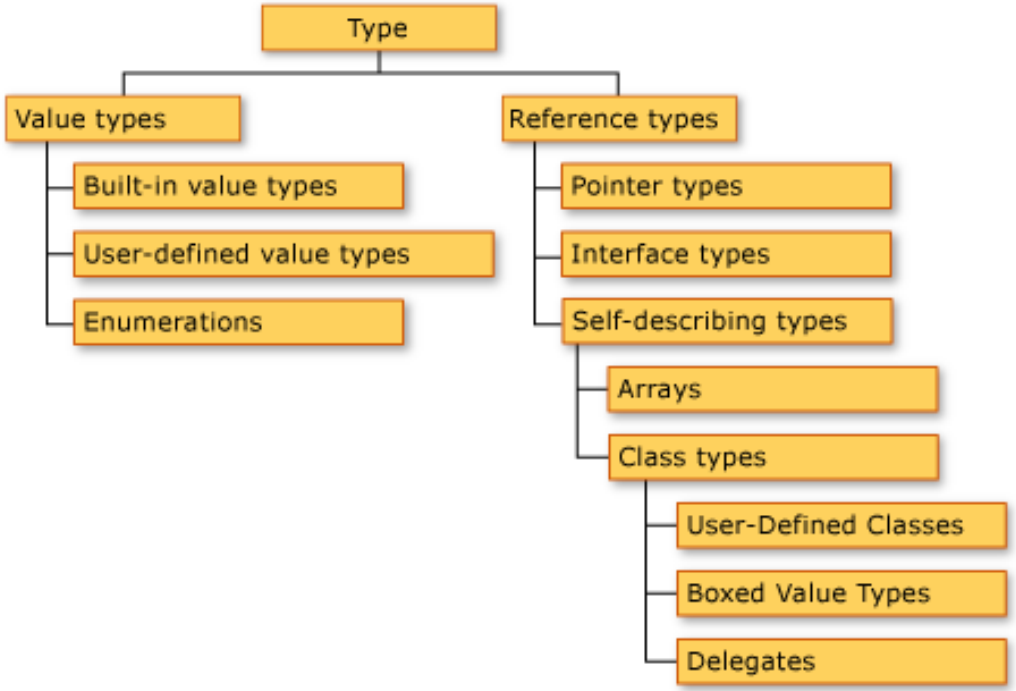
+	-	*	/	%	&	|	^	!	~	=
<	>	?	++	--	&&	||	<<	>>	==	!=
<=	>=	+=	-=	*=	/=	%=	&=	!=	^=	<<=
>>=	->									

C#의 구분자

{	}	[	]	(	)	.	,	;	:

CTS (Common Type System)

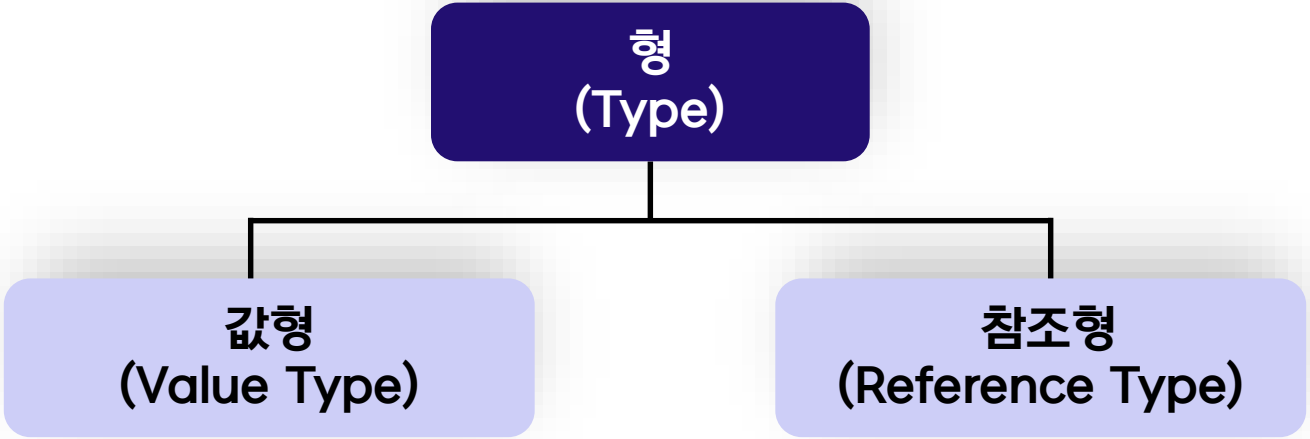
CTS (Common Type System)



http://msdn.microsoft.com/en-us/library/2hf02550.aspx

CTS (Common Type System)

CTS (Common Type System)



- 메모리영역: 스택(stack)
- 데이터형: int, float, struct, enum 등

- 메모리영역: 스택(stack)과 힙(heap)
- 데이터형: class, interface, delegate등

Data Type

예제

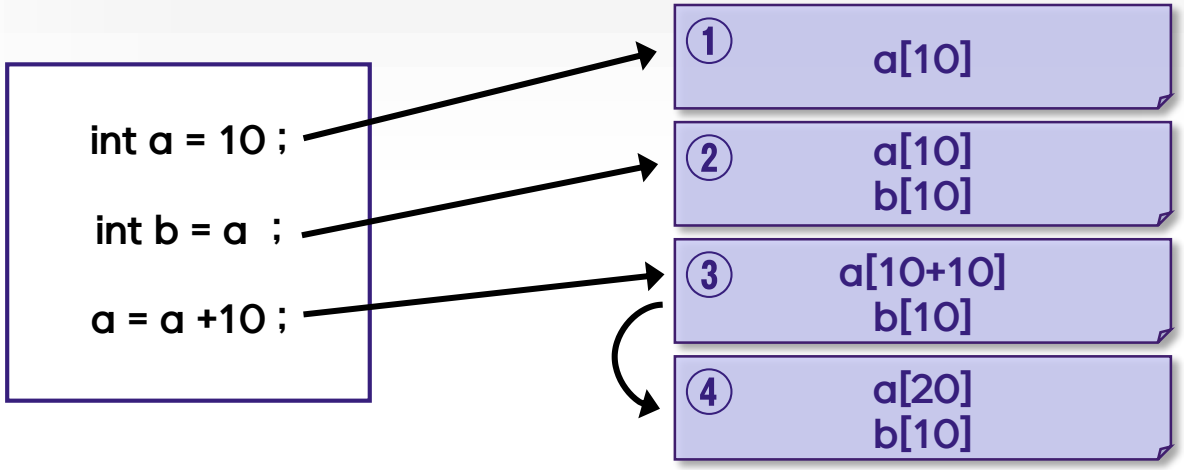
```
using System;
class Class1 {
    public int Value = 0;
}
class Test {
    static void Main(string[] args)  {
        int val1 = 0;
        int val2 = val1;
        val2 = 123;
        Class1 ref1 = new Class1();
        Class1 ref2 = ref1;
        ref2.Value = 123;
        Console.WriteLine("Value type: {0}, {1}", val1, val2);
        Console.WriteLine("Ref type: {0}, {1}", ref1.Value, ref2.Value);
    }
}
```

Value type: 0, 123
Ref type: 123, 123

Value Type

값 형식

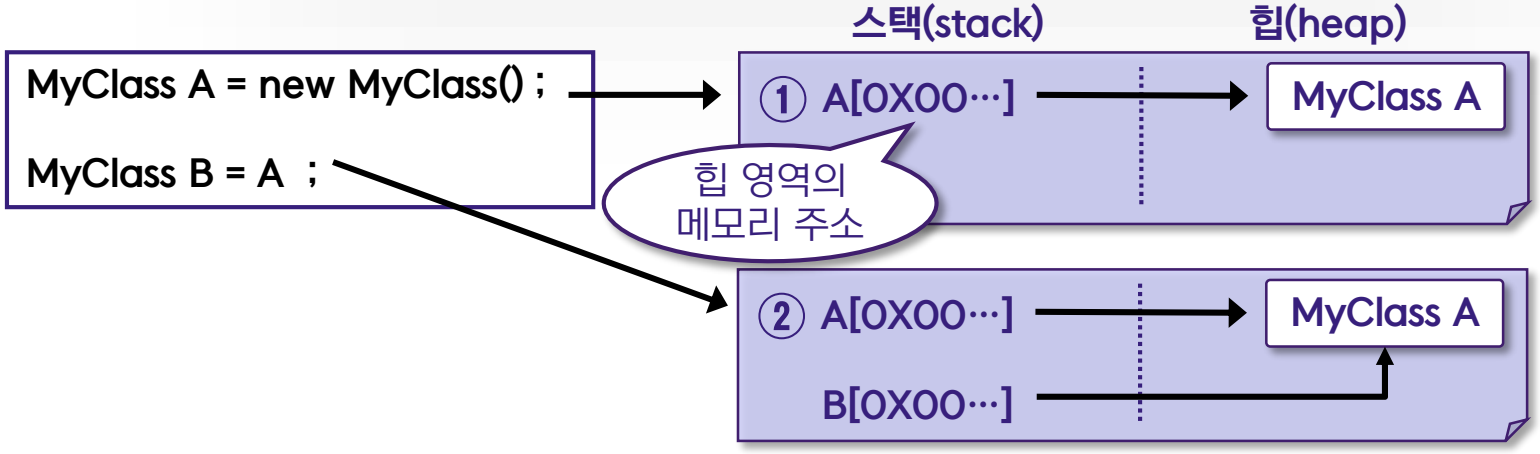
- 값 형은 직접 값을 메모리에 갖고 있으며 동일한 객체를 가리킬 수 없기 때문에 한 값의 변경이 다른 것에 영향을 줄 수 없음
- bool, byte, char, decimal, double, float, int, long, sbyte, short, uint, ulong, ushort, struct, enum, Tuple, Nullable



Reference Type

참조 형식

- 참조 형은 값을 갖지 않고 메모리에 있는 어떤 값을 가리킴
  - ✓즉, 동일한 객체를 가리키는 것이 가능하며 변경된 값이 다른 참조 형 값에 영향을 줄 수 있음
- class, delegate, interface, object, string, array



Built-in Types

분류	키워드	구조체형	크기(byte)
숫자형 (정수형)	sbyte byte short ushort int uint long ulong	System.SByte System.Byte System.Int16 System.UInt16 System.Int32 System.UInt32 System.Int64 System.UInt64	1 1 2 2 4 4 8 8
숫자형 (실수형)	float double decimal	System.Single System.Double System.Decimal	4 8 16
문자형	char	System.Char	2
문자열형	string	System.String	
불리언형	bool	System.Boolean	1비트
Object형	object	System.Object	

Numerical

- 정수형 : sbyte, byte, short, ushort, int, uint, long, ulong
- 실수형 : float, double, decimal

```
using System;
class NumericType {
    static void Main(string[] args) {
        int intVal = 10;
        Console.WriteLine(“intVal = {0}", intVal);        //intVal = 10
        intVal = 10 + 15;
        Console.WriteLine(“intVal = {0}", intVal);        // intVal = 25

        float floatVal = 1.234f;
        Console.Write( “floatVal = " );
        Console.WriteLine( floatVal );                    // floatVal = 1.234
        decimal decimalVal = 16.24m;
        Console.WriteLine( “decimal = “ + decimalVal ); // decimalVal = 16.24
    }
}
```

Character

문자형은 일반적인 문자를 다루는 데이터형 (유니코드 사용)

```
using System;
class CharType {
    static void Main(string[] args) {
        char engVal = 'A';
        Console.WriteLine("engVal = {0}", engVal);    //engVal = A

        char korVal = '가';
        Console.WriteLine("korVal = {0}", korVal);    // korVal = 가

        char chiVal = '漢';
        Console.WriteLine("chiVal = {0}", chiVal);    // chiVal = 漢
    }
}
```

Character

특수 문자	설명
\n	Newline
\t	Horizontal tab
\r	Carriage return
\\	Backslash
\’	Single quote
\"	Double quote
\0	Null
\b	Backspace
\f	Form feed
\v	Vertical tab
\u	Unicode print
\a	경고음 발생

Boolean

- 논리 값인 참(true)와 거짓(false)를 다루는 형

```
class BoolType {
    static void Main(string[] args) {
        bool boolVal = true;
        Console.WriteLine(boolVal);    // True
        int x = 123;
        if (x != 0)                    // x가 0이 아닌것이 true이면
            Console.WriteLine("The value {0} is non-zero.", x);
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();
        if (Char.IsLetter(c)) {        // 입력받은 글자가 letter 가 true이면
            if (Char.IsLower(c)) {     // 입력받은 글자가 lower case letter가 true이면
                Console.WriteLine("The character is lowercase.");
            } else {                   // 입력받은 글자가 lower case letter가 false이면
                Console.WriteLine("The character is uppercase.");
            }
        } else {                       // 입력받은 글자가 letter가 false이면
            Console.WriteLine("The character is not an alphabetic character.");
        }
    }
}
```

Enumeration

열거형 (enum)

- 문법

```
enum <식별자> {멤버1, 멤버2, 멤버3,…} ;
```

- 예

```
enum WeekDay {
        Monday, Tuesday, Wednesday,
        Thursday, Friday, Saturday, Sunday
   }

class EnumType {
   static void Main (string [] args) {
        WeekDay offday = WeekDay.Sunday;
        WeekDay today = (WeekDay)2;          // Wednesday
   }
}
```

Enumeration

열거형 (enum)

```
using System;
class EnumType {
    enum WeekDay1 { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
    static void Main( string[] args ) {
        int x1 = (int)WeekDay1.Sun;
        int y1 = (int)WeekDay1.Fri;

        Console.WriteLine("WeekDay1 Sun = {0}", x1);    // Sun = 0
        Console.WriteLine("WeekDay1 Fri = {0}", y1);    // Fri = 5
    }
}
```

Enumeration

열거형 (enum)

```
using System;
class EnumType {
    enum WeekDay2 { Sat = 1, Sun, Mon, Tue, Wed, Thu, Fri };
    static void Main( string[] args ) {
        int x2 = (int)WeekDay2.Sun;
        int y2 = (int)WeekDay2.Fri;

        Console.WriteLine("WeekDay2 Sun = {0}", x2);    // Sun = 2
        Console.WriteLine("WeekDay2 Fri = {0}", y2);      // Fri = 7
    }
}
```

Enumeration

열거형 (enum)

```
using System;
class EnumType {
    enum WeekDay3 {Sun, Mon, Tue, Wed, Thu, Fri=8, Sat };
    static void Main( string[] args ) {
        int x3 = (int)WeekDay3.Sun;
        int y3 = (int)WeekDay3.Fri;
        int z3 = (int)WeekDay3.Sat;

        Console.WriteLine("WeekDay3 Sun = {0}", x3);    // Sun = 0
        Console.WriteLine("WeekDay3 Fri = {0}", y3);    // Fri = 8
        Console.WriteLine("WeekDay3 Sat = {0}", z3);    // Sat = 9
    }
}
```

Enumeration

열거형 (enum)

```
using System;
class EnumType {
    enum WeekDay4 {Sun=100, Mon=200, Tue=500, Wed=1000,
                   Thu=3000, Fri=8000, Sat=10000};
    static void Main( string[] args ) {
        int x4 = (int)WeekDay4.Mon;
        int y4 = (int)WeekDay4.Tue;
        int z4 = (int)WeekDay4.Thu;

        Console.WriteLine("WeekDay4 {0} = {1}", WeekDay4.Mon, x4); // Mon=200
        Console.WriteLine("WeekDay4 {0} = {1}", WeekDay4.Tue, y4);//Tue=500
        Console.WriteLine("WeekDay4 {0} = {1}", WeekDay4.Thu, z4);//Thu=3000
    }
}
```

Enumeration

```
using System;
class DataType {
    enum Gender { Male, Female };
    static void Main(string[] args) {
        foreach (Gender g in Gender.GetValues(typeof(Gender))) {
            Console.WriteLine(g);
        }
    }
}
```

Structure

구조체 (struct)

- C#의 구조체는 여러 가지 다른 형태의 데이터를 그룹 하나로 묶어서 관리하는 데 사용함
  - ✓C#의 struct는 경량 객체를 캡슐화하기 위해 디자인 되어있음
  - ✓즉, 참조 형식이 아닌 값 형식이므로 값에 의해 전달됨
- C#에서의 struct는 class와 매우 다름
  - ✓C의 구조체가 데이터만을 member로 가질 수 있지만, C++의 구조체는 함수를 가질 수 있음
  - ✓C++에서는 struct와 class가 거의 차이가 없으며, 차이점은 아무런 명시를 하지 않았을 때 class는 멤버가 private 권한을 가지고 struct는 public을 가짐
  - ✓C#에서의 struct는 class와 유사하나, 매우 제한적임

Structure

구조체 (struct)

- C#의 struct는 기본 생성자(default constructor)나 소멸자 (destructor)를 선언할 수 없음
  - **물론, 기본 생성자 이외의 생성자는 선언이 가능함**
- C#의 struct는 다른 구조체나 클래스에서 상속받을 수 없으며, 파생시킬 수도 없음
  - **모든 struct는 System.ValueType에서 직접적으로 상속받음**
- C#에서는 struct를 다른 클래스가 상속받아서 사용하지 않으므로 protected를 선언할 수 없음
- C#의 struct는 interface를 구현할 수 있음
- C#의 struct는 nullable type처럼 사용할 수 있음

Structure

구조체 예

```
struct Point {
    public int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
class StructType {
    static void Main(string[] args) {
        Point p = new Point();
        Console.WriteLine("Point x={0} y={1}", p.x, p.y);   // x=0, y=0
        Point p2 = new Point(5,5);
        Console.WriteLine("Point2 x={0} y={1}", p2.x, p2.y); // x=5, y=5
        Point p3;
        p3.x = 10; p3.y = 20;
        Console.WriteLine("Point3 x={0} y={1}", p3.x, p3.y); // x=10, y=20
    }
}
```

String

String Type

- 문자열 객체는 스택이 아닌 힙 영역에 할당
- 어떤 문자열 변수를 다른 문자열 변수로 할당하면 메모리 상의 동일한 문자열을 가리키는 두 가지 참조가 생성

```
string a = "Hi Welcome to HCI Programming II";
string b = "Hi ";
b += "Welcome to HCI Programming II";
Console.WriteLine(a == b);                  // True – 문자열내용은 같음
Console.WriteLine((object)a == (object)b); // False – 문자열이 다른 곳을 참조

string str = "TEST";
char c = str[2];
Console.WriteLine(c);                       // c = 'S';

string d = "\nGood Morning \u0066";       // newline character \n
Console.Write(d);  // \udddd (dddd is 4-digit number)는 유니코드를 표현
string e = @"C:\MM\Courses\a.txt";      // "C:\\MM\\Courses\\a.txt“
Console.WriteLine(e);
```

String

```
string str = "Hi, Welcome to HCI Programming! Hi";
Console.WriteLine(“str의 길이는 : {0}”, str.Length);        // 길이는 34
Console.WriteLine(“대문자로 변환: ” + str.ToUpper());   // 대문자 출력
Console.WriteLine(“소문자로 변환: “ + str.ToLower());   // 소문자
Console.WriteLine(“Index of \”Hi\”: “ + str.IndexOf(“Hi”));          // 0
Console.WriteLine(“LastIndex of \”Hi\”: “ + str.LastIndexOf(“Hi”)); // 32
// Hello Welcome to HCI Programming Hello
Console.WirteLine(“Replace \”Hi\” to \”Hello\”: “ + str.Replace(“Hi”, “Hello”));
// Welcome to HCI Programming Hi
Console.WriteLine(“str의 Substring(3): {0}”, str.Substring(3));
// HCI Programming Hi
Console.WriteLine(“str의 Substring(14,19): {0}”, str.Substring(14,19));
// C# string Split
char[] separators = {' ', '\n', '\t', // white space
                     '.', '\"', ';', ',', '?', '!', ')', '(', '<', '>', '[', ']'};
string[] words = str.Split(separators, StringSplitOptions.RemoveEmptyEntries);
foreach (string s in words)    {
        Console.WriteLine(s);                    // 토큰화된 단어를 출력시킨다
}
```

Object Type

다른 모든 객체가 파생되는 근본 형식

System.Object instance 메소드

- bool Equals(Object) 메소드
  - ✓지정한 객체가 자신과 비교하여 동일하면 true 아니면 false
  - ✓Value 형인 경우 데이터형이 일치하고 값이 같은 경우 true
- int GetHashCode() 메소드
  - ✓객체의 Hash Code를 반환
- Type GetType() 메소드
  - ✓기본 형식, 메소드, 속성 등 객체의 데이터 형식(Type)을 가리키는 클래스의 방대한 정보를 제공
  - ✓다형성(Polymorphism)을 사용할 때 유용

Object Type

다른 모든 객체가 파생되는 근본 형식

System.Object instance 메소드

- string ToString() 메소드
  - ✓어떤 객체에 대한 설명을 간단하고 빠르게 문자열로 표현하기 위한 메소드
  - ✓디버깅할 경우와 같이 객체의 내용을 빨리 보고 싶은 경우 사용
- void Finalize()
  - ✓ 객체 소멸 시 garbage collector에 의해 실행
  - ✓ Garbage collector에서 object를 회수하기 전에 object가 리소스를 해제하고 다른 정리작업을 수행할 수 있게 함
- Object MemberwiseClone()
  - ✓ 현재 object의 단순 복사본을 만들어서 반환

Object Type

System.Object 의 static 메소드

- bool Equals(Object, Object)
  ✓지정한 두 객체 사이의 인스턴스 비교
- bool ReferenceEquals(Object, Object)
  ✓지정한 두 객체 사이의 참조 비교

Object Type

```
class DataType {
    static void Main(string[] args) {
        int i = 100;
        Console.WriteLine("i의 GetType?:{0}",i.GetType());
        Console.WriteLine("i의 GetType().BaseType?:{0}",i.GetType().BaseType);
        Console.WriteLine("i의 ToString?:{0}",i.ToString());
        Console.WriteLine();
        string s = “TEST";
        Console.WriteLine("s의 GetType?:{0}",s.GetType());
        Console.WriteLine("s의 GetType().BaseType?:{0}",s.GetType().BaseType);
        Console.WriteLine("s의 ToString?:{0}",s.ToString());
        Console.WriteLine();
        System.Int32 j = i;
        s = i.ToString();
        Console.WriteLine("s와 i는 같은가?{0}",s.Equals(i));
         Console.WriteLine("i와 j는 같은가?{0}",i.Equals(j));
    }
}
```

```
i의 GetType?:System.Int32
i의 GetType().BaseType?:System.ValueType
i의 ToString?:100
s의 GetType?:System.String
s의 GetType().BaseType?:System.Object
s의 ToString?:TEST
s와 i는 같은가? false
i와 j는 같은가?true
```

Nullable Type (C# 2.0)

- 정수 (int)나 날짜(DateTime) 같은 값형식(Value Type)은 일반적으로 NULL을 가질 수 없음. 하지만 DB에서는 가능
- Nullable Type으로 내부 형식의 값을 모두 나타낼 수 있을 뿐만 아니라 null 값을 추가로 나타낼 수 있음
  - System.Nullable<T> variable
  - 또는 T? variable
  - T는 struct을 포함한 모든 값 형식만 가능 참조 형식은 불가능

```
// nullable 형식으론 모든 값 형식에 사용가능
int?  x = 10;
double? pi = 3.14;
bool? flag = null;
char? Letter = 'a';
int?[] arr = new int?[10];

if (x.HasValue) System.Console.WriteLine(x.Value);
else System.Console.WriteLine("Undefined");
```

var (C# 3.0)

- var 키워드는 초기화 문의 오른쪽에 있는 식에서 변수 형식을 유추하도록 컴파일러에 지시함
- 유추된 형식은 기본 제공 형식, 사용자 정의 형식 또는 .NET Framework 클래스 라이브러리에서 정의된 형식임

```
var  x = 10;   // compiled as an int
var pi = 3.14;           // compiled as a double
var s = “Hello”;         // compiled as a string
var c = ‘a’;     // compiled as a char
var[] arr = new [] { 0, 1, 2 };  // compiled as int[]

var list = new List<int>();      // complied as List<int>
for (var x = 1; x < 10; x++)
foreach (var item in list){...}
using (var file = new StreamReader("C:\\myfile.txt")) {...}
```

Tuple (C# 4.0)

- 튜플은 여러 개의 멤버를 가지는 데이터 구조임 (멤버는 8개까지 가능)
- C++의 std::pair 형식과 비슷함
- KeyValuePair<Tkey, TValue>는 struct 이고, Tuple은 class 임

```
var p = new Tuple<string, int>("HCI", 2015);
// Item1, Item2, …, Item7, Rest (the final property)
Console.WriteLine("p={0} {1}", p.Item1, p.Item2);

// Create() factory method can construct from a 1-tuple to an 8-tuple
var p1 = new Tuple<int, string, double>(1, "hello", 3.14);
var p2 = Tuple.Create(2, "hci", 0.001); // Tuple<int, string, double>

var p3 = new Tuple<int, int, int, int, int, int, int, Tuple<int, int>>(1, 2, 3, 4, 5, 6, 7,
Tuple.Create(8, 9)); // For higher than 8 items, use nested tuple
Console.WriteLine("8th={0} 9th={1}", p3.Rest.Item1, p3.Rest.Item2);
```

데이터형 변환

- **Implicit type conversion (암시적 형 변환)**
- **Explicit type conversion (명시적 형 변환)**
  - Type Cast
- **User-defined conversion**
  - Conversion Operator (형변환 연산자)
- **Conversion with helper classes**
  - System.Convert 클래스
  - Parse, TryParse 메소드

데이터형 변환

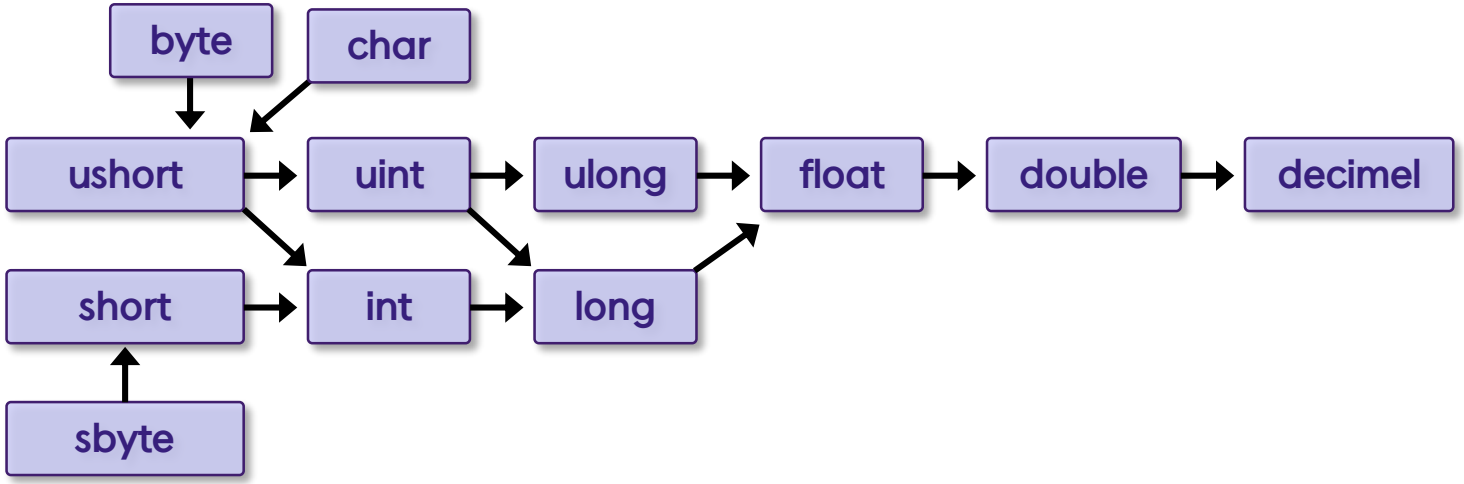
Implicit type conversion (암시적 형 변환)

- 직접 캐스팅하지 않고도 형의 변화가 일어남
- 데이터 형의 호환성, 사이즈 등이 적절할 때 가능

```
// int->long 암시적 형변환
int   intValue  = 127 ;
long longValue  = intValue ;            // 암시적 형변환

// 파생클래스->기반클래스 암시적 형변환
Derived d = new Derived();
Base b = d;                             // 암시적 형변환 가능
```

데이터형 변환

Implicit type conversion (암시적 형 변환)



데이터형 변환

Explicit type conversion (명시적 형 변환)

- 명시적으로 직접 캐스팅하여 형의 변화가 일어남

```
// long->int로는 type casting을 사용하여 명시적 형변환을 해야 함
long   longValue  = 32767 ;
int  intValue  = (int) longValue ;  // 형 변환을 위한 type casting

// 기반클래스->파생클래스로는 명시적 형변환을 해야 함
Dog d = new Dog();
Animal a = d;
Dog d2 = (Dog) a;    // 명시적 형변환
```

Convert Class

System.Convert 클래스를 사용하여 형 변환 처리

- ToBoolean, ToByte, ToChar, ToDecimal, ToDouble,ToInt16m Toint32m ToInt64, ToSByte, ToSingle, ToString, ToUInt16, ToUInt32, ToUInt64
- Char와 Boolean, Single, Double, Decimal 사이에 변환 시 예외발생

```
class TypeConvert {
    static void Main(string[] args) {
        string i = "230";
        int  j;
        j = Convert.ToInt32(i);          // i를 int형으로 변환
        Console.WriteLine("문자열로 처리: {0}",i + 70);
        Console.WriteLine("숫자로   처리: {0}",j + 70);
    }
}
```

```
문자열로 처리 : 23070
숫자로 처리 :   300
```

Parse Method

Parse 메소드를 사용하여 형 변환

- int Int32.Parse(string)는 string을 32-bit signed integer로 변환
- bool Int32.TryParse(string, out int)는 string을 32-bit signed integer로 변환하여 out에 보내주는 메소드로 성공하면 true 반환

```
class StringToIntConvert {
    static void Main(string[] args) {
        string s1 = “123”;
        string s2 = “12345678901234567890”;
        int result = Int32.Parse(s1);              // 123
        result = Int32.Parse(s2);                  // Exception
        // success=true, result=123
        bool success = Int32.TryParse(s1, out result);
        // success=false, result=0
        success = Int32.TryParse(s2, out result);
    }
}
```

Convert vs. Parse vs. TryParse

- Convert.ToInt32는 내부적으로 Int32.Parse를 불러서 string을 32-bit signed integer로 변환, 예외상황에서 exception 발생

```
string s1 = "1234";
string s2 = "1234.65";
string s3 = null;
string s4 = "12345678912345678912345678912345678912345678";
int result;
bool success;
result = Convert.ToInt32(s1); //-- 1234
result = Convert.ToInt32(s2); //-- FormatException
result = Convert.ToInt32(s3); //-- 0
result = Convert.ToInt32(s4); //-- OverflowException
result = Int32.Parse(s1); //-- 1234
result = Int32.Parse(s2); //-- FormatException
result = Int32.Parse(s3); //-- ArgumentNullException
result = Int32.Parse(s4); //-- OverflowException

success = Int32.TryParse(s1, out result); //-- success => true; result => 1234
success = Int32.TryParse(s2, out result); //-- success => false; result => 0
success = Int32.TryParse(s3, out result); //-- success => false; result => 0
success = Int32.TryParse(s4, out result); //-- success => false; result => 0
```

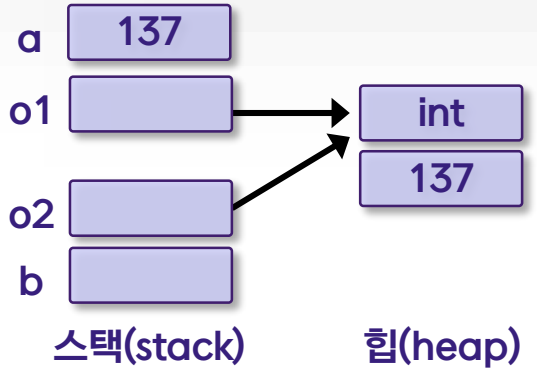
Boxing and Unboxing

Boxing

- Value 형식의 자료형을 Reference 형으로 바꾸는 것

Unboxing

- Reference 형식의 자료형을 Value 형으로 바꾸는 것
- Boxing 할 때 명시적인 변환은 필요하지 않지만 Unboxing을 할 때에는 필요함




```
{
    // boxing (값 형식 -> object 형식)
    int a = 137;
    object o1 = a;

    object o2 = o1;

    // unboxing (object 형식 -> 값 형식)

    int b = (int)o2;
}
```

Operator

분류		연산자
단항		+ - ! ~ ++ --
산술	곱셈/나눗셈	* / %
	덧셈/뺄셈	+ -
시프트		<< >>>
관계	비교	< > <= => is as
	등가	= = ! =
비트	비트 AND	&
	비트 OR	|
	비트 XOR	^
조건	조건 AND	&&
	조건 OR	||
	조건	?:
대입		= *= /= += -= <<= >>= &= ^= |=

Operator

Arithmetic operator

```
static void Main(string[] args) {
// 중간생략
        int sum = number1 + number2;
        int diff = number1 - number2;
        int mul = number1 * number2;
        int div = number1 / number2;
        int mod = number1 % number2;

        Console.WriteLine( "\n{0} + {1} = {2}.", number1, number2, sum );
        Console.WriteLine( "\n{0} - {1} = {2}.", number1, number2, diff );
        Console.WriteLine( "\n{0} * {1} = {2}.", number1, number2, mul );
        Console.WriteLine( "\n{0} / {1} = {2}.", number1, number2, div );
        Console.WriteLine( "\n{0} % {1} = {2}.", number1, number2, mod );
}
```

Operator Precedence

Operator Precedence 규칙

- ()를 가장 먼저 계산함
- /, *, %은 그 다음 순서로 계산 (왼쪽에서 오른쪽 순서로)
- +과 –을 마지막으로 계산 (왼쪽에서 오른쪽 순서로)

```
static void Main(string[] args) {
// 중간생략
        int result1 = a + b + c + d + e;
        int result2 = a + b * c - d / e;
        int result3 = a / (b + c) - d % e;
        int result4 = a / (b * (c + (d – e)));
        Console.WriteLine( "\n result1 = {0}", result1 ); // result1 = 15
        Console.WriteLine( "\n result2 = {0}", result2 ); // result2 = 7
        Console.WriteLine( "\n result3 = {0}", result3 ); // result3 = -4
        Console.WriteLine( "\n result4 = {0}", result4 ); // result4 = 0
}
```

Assignment Operators

Assignment operator	Sample expression	Explanation
+=	c += 7	c = c + 7
-=	d -= 4	d = d - 4
*=	e *= 5	e = e * 5
/=	f /= 3	f = f / 3
%=	g %= 2	g = g % 2

Increment and Decrement Operators

Operator	Called	Sample expression	Explanation
++	preincrement	**++a**	Increment **a** by 1, then use the new value of **a** in the expression in which **a** resides.
++	postincrement	**a++**	Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1.
--	predecrement	**--b**	Decrement **b** by 1, then use the new value of **b** in the expression in which **b** resides.
--	postdecrement	**b--**	Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1.

Increment and Decrement Operators

Increment operator

```
static void Main(string[] args) {
// 중간생략
        int i = 5;
        Console.WriteLine(i);          // 5
        Console.WriteLine(i++);        // 5
        Console.WriteLine(i);          // 6

        int j =5;
        Console.WriteLine(j);          // 5
        Console.WriteLine(++j);        // 6
        Console.WriteLine(j);          // 6
}
```

Equality and Relational Operators

- 비교 연산자는 식(변수, 상수, 변수의 계산식) 두 개를 비교해서 Boolean 결과 값을 반환함

Standard algebraic equality operator or relational operator	C# equality or relational operator	Example of C# condition	Meaning of C# condition
Equality operators			
=	==	**x == y**	**x** is equal to **y**
≠	!=	**x != y**	**x** is not equal to **y**
Relational operators			
>	>	**x > y**	**x** is greater than **y**
<	<	**x < y**	**x** is less than **y**
≥	>=	**x >= y**	**x** is greater than or equal to **y**
≤	<=	**x <= y**	**x** is less than or equal to **y**

Logical and Conditional Operators

논리 연산자는 Boolean 결과 값을 반환함

- ! ····· Logical NOT
- & ····· Logical AND
- | ····· Logical OR
- ^ ····· Logical exclusive OR(XOR)
- && ····· Conditional AND
- || ····· Conditional OR

Expression	!Expression
false	True
true	false

Expression1	Expression2	Expression1 ^ Expression2
false	false	false
false	true	true
true	false	true
true	true	false

Logical and Conditional Operators

Expression1	Expression2	Expression1 && Expression2
false	false	False
false	true	False
true	false	False
true	true	true

Expression1	Expression2	Expression1 || Expression2
false	false	False
false	true	true
true	false	true
true	true	true

Precedence and Associativity

high

low

Operators	Associativity	Type
() ++ --	left to right right to left	parentheses unary postfix
++ -- + - *(type)*	right to left	unary prefix
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Code Block

- 여러 명령문을 논리적으로 결합해야 할 때 중괄호 ({ })를 사용하여 명령문 그룹을 만들어 표현
  - 이러한 명령문 그룹을 코드 블록(code block)이라고 함
- 코드 블록 안에는 변수를 선언할 수 있고, 다른 코드 블록을 포함할 수도 있음

```
public static void Main(string[] args) {
    int outer;
    {
        int inner;
        outer = 1;
        inner = 2;
    }
    outer = 5;
    inner = 10;          // 오류
}
```

내부 코드블록

Main() 코드블록

Control Statement

제어문의 종류

- 제어문이란 프로그램을 실행할 때는 논리적인 흐름이 필요한데, 이러한 문장의 논리적인 흐름을 통제해 주는 것

조건문

조건식의 값에 따라 각각에 해당되는 명령문을 수행한다.

- if 문, switch 문

반복문

조건이 만족하는 동안 특정 명령문을 반복적으로 수행한다.

- while 문, do 문, for 문, foreach 문

점프문

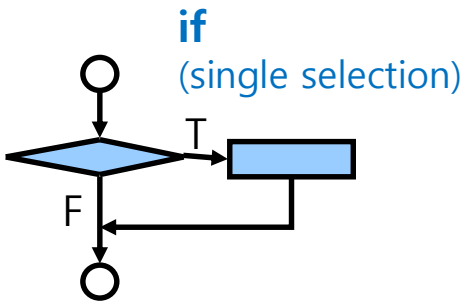
제어권을 이동시킬 때 점프문을 사용한다.

- goto 문, break 문, continue 문

If Statement

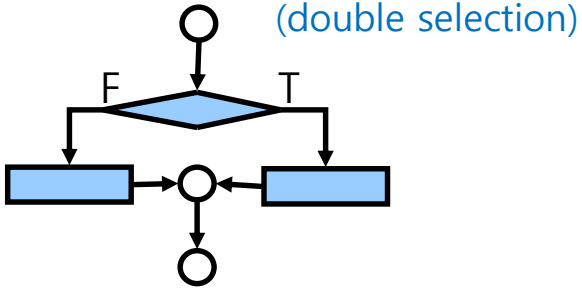
if 문

```
if(조건식)
        명령문1 ; // 조건식이 참일 때 실행
// 조건식이 거짓일 때 실행
```



if ~ else 문

```
if(조건식)
        명령문1 ; // 조건식이 참일 때 실행
else
        명령문2 ; // 조건식이 거짓일 때 실행
```



If Statement

다중 if ~ else 문

- If가 여러 번 쓰일 경우는 코드블록{}을 이용하여, 조건식의 참/거짓 실행문을 명확히 해야 함
- 컴파일러는 else 문가 "indentation과는 상관없이" 가장 마지막으로 unmatched if 문에 연결해서 해석함

```
if (point >=0 && point <=100) {
        if (point >50)
                result = "Pass";
}
else {
        Console.WriteLine("에러:범위를 벗어났습니다.");
}
```

Unbalanced if-else Statements

```
if (point >=0 && point <=100)
        if (point >50)
                result = “Pass”;
else
        Console.WriteLine(“에러:범위를 벗어났습니다.”);
```

컴파일러는 아래와 같이 해석함

```
if (point >=0 && point <=100) {
        if (point >50)
                result = “Pass”;
else
        Console.WriteLine(“에러:범위를 벗어났습니다.”);
}
```

Ternary Conditional Operator (? :)

exp1 ? exp2 : exp3 Operator

- if ~ else 문과 비슷함

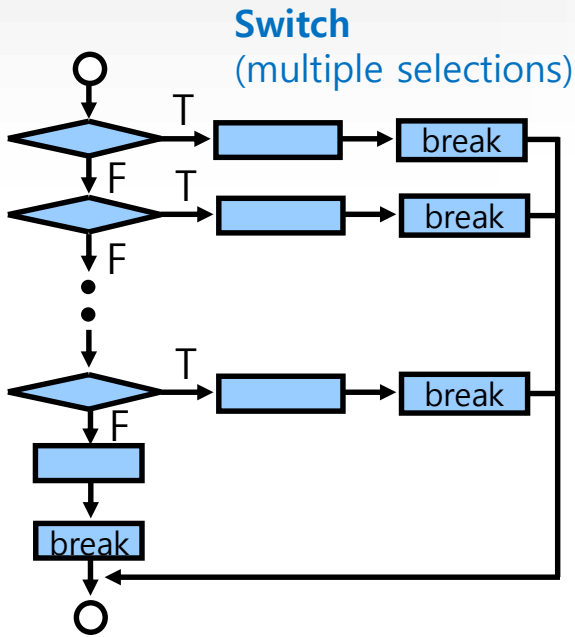
```
string result;

int num;

result = (num == 1) ? “Quarter” : “Quarters”;
```

Switch Statement

switch 문

- 조건변수에 해당하는 특정 레이블로 이동한 후 명령문을 바로 실행할 수 있음

```
switch (조건변수){
case 값1 : // 조건변수의 값이 값1과 일치할 때 실행
        명령문1 ;
        break ;
...
case 값n : // 조건변수의 값이 값n과 일치할 때 실행
        명령문n ;
        break;
default :  // 어떠한 값도 만족하지 않을 때 실행
        명령문z ;
        break;
}
```



Switch Statement

switch 문의 예제

```
int num = 23 ;
switch(num%5) {
case 1 :
        Console.WriteLine("나머지의 값은 1입니다.");
        break ;
case 2 :
        Console.WriteLine("나머지의 값은 2입니다.");
        break ;
case 3 :
        Console.WriteLine("나머지의 값은 3입니다.");
        break ;
case 4 :
        Console.WriteLine("나머지의 값은 4입니다.");
        break ;
default :
        Console.WriteLine("5의 배수입니다.");
        break;
}
```

num 변수의 값을 5로 나눈 나머지의 값과 동일한 값을 갖는 case 블록으로 이동해 해당 case 블록의 명령문을 수행한다.

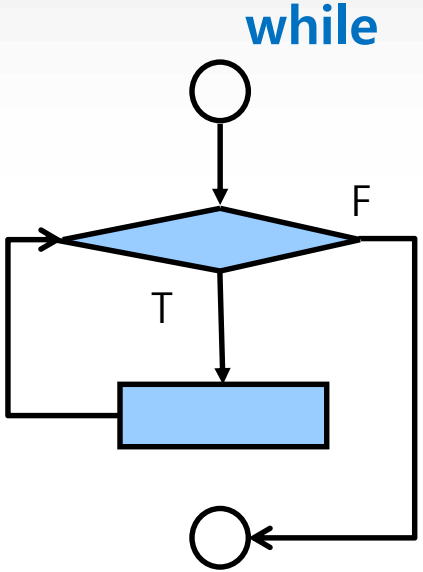
이 예제에서는 num 변수가 23이므로 case 3 블록으로 이동"나머지의 값은 3입니다."를 출력한다.

While Statement

while 문

- while 문은 조건식을 만족하는 동안, 반복적으로 명령문을 실행하고, 조건식이 false가 되어야 while 문을 빠져나감

```
while (조건식) {
        명령문1 ;
        명령문2 ;
        …
        명령문n ;
}
```

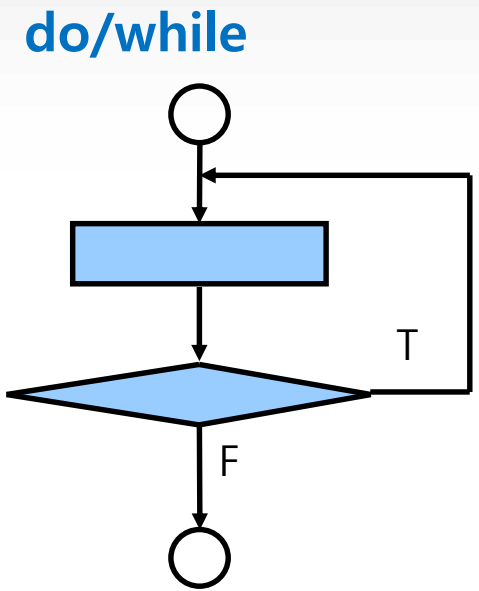


Do while Statement

do while 문

- do while 문은 명령문을 먼저 실행한 후에 조건을 검색해야 하는 경우에 사용

```
do {
        명령문1 ;
        명령문2 ;
        …
        명령문n ;
}  while (조건식)
```



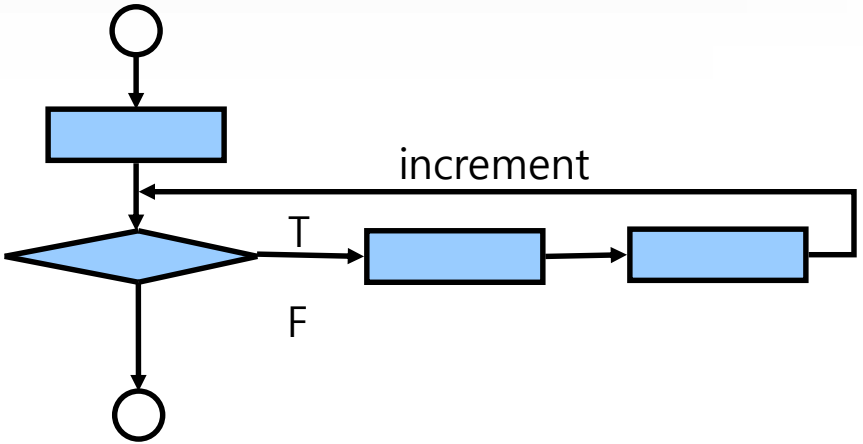
For Statement

for 문

- while 문과 do..while 문이 조건에 따라 반복 수행한다면 for문은 반복 횟수가 먼저 결정되었을 때 사용

```
for (초기값; 조건식; 증감)
{
        명령문1 ;
        명령문2 ;
        …
        명령문n ;
}
```

for /foreach



Foreach Statement

foreach 문

- 일련의 데이터를 하나의 객체에 저장하고자 할 때 배열이나 컬랙션 객체를 이용
- foreach 문은 이런 배열과 컬랙션 객체 내의 데이터에 순차적으로 접근할 때 사용

```
foreach (멤버타입 변수명 in 배열/컬랙션명) {
        명령문1 ;
        명령문2 ;
        …
        명령문n ;
}
```

Goto Statement

goto 문

- goto 문은 프로그램 실행의 제어권을 이동시킴
- goto 문이 지시하는 레이블은 반드시 존재해야 하고 goto 문이 접근 가능한 위치여야 함
- goto 문은 사용된 곳보다 상위 코드 블록으로는 제어권을 이동할 수 있지만 하위 코드 블록으로는 제어권을 이동할 수 없음
- goto 문을 사용하면 자칫 프로그램의 흐름을 혼란스럽게 만들 수 있기 때문에 switch 문 외에는 사용을 자제하는 편이 좋음

Break, Continue Statement

break, continue 문

- break 문의 역할은 반복문의 코드 블록 밖으로 제어권이 나오게 하는 것임
- 이와 비슷하지만 continue 문은 제어권을 반복문의 조건식으로 이동시킴
- 문장에서 break 문을 만나면 반복문 (while, do.. While, for, foreach 문)이나 switch 문은 실행을 멈춤
- 이에 반해 continue 문을 만나면 반복문의 조건식을 다시 조사하여 실행여부를 결정함

Break Statement

```
static void Main( string[] args ) {
        string output = "";
        int count;

        for ( count = 1; count <= 10; count++ )
        {
          if ( count == 3 )
            break;          // break out of the loop, if count==3

          output += count + " ";

        } // end for loop

        output += "\nBroke out of loop at count = " + count;
        Console.WriteLine( output );
}
```

Continue Statement

```
static void Main( string[] args ) {
        string output = "";
        int count;

        for ( count = 1; count <= 10; count++ )
        {
          if ( count == 5 )
            continue;          // skip remaining code in loop, only if count==5

          output += count + " ";

        } // end for loop

        output += "\nContinue to skip printing 5 “;
        Console.WriteLine( output );
}
```

Exception Handling

try/catch 문

- 프로그램이 실행 중에 에러가 나거나 멈출 수 있는데, 이러한 현상을 통틀어 예외현상 (exception)이라고 함
- 예외현상을 사전에 방지하기 위해 예외현상이 발생 가능한 지점에 try/catch 문을 적용함

```
try {
        예외처리가 필요한 구문
}
catch(예외종류) {
        예외현상 발생 시 실행할 명령
}
```

Exception Handling

try/catch/finally 예제

```
try  {
    // b=0일 때, 예외현상 발생
    c = a/b;
}
catch(Exception e) {
    Console.WriteLine("변수의 값이 올바르지 않습니다.");
    Console.WriteLine(“발생에러 : {0}”,e.ToString());
    Environment.Exit(0);
}
finally  {
   //예외현상 발생여부와 상관없이 실행 한다.
   Console.WriteLine("프로그램을 종료 합니다.");
}
```

Reference

C# Types

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types