# 3장. 타입

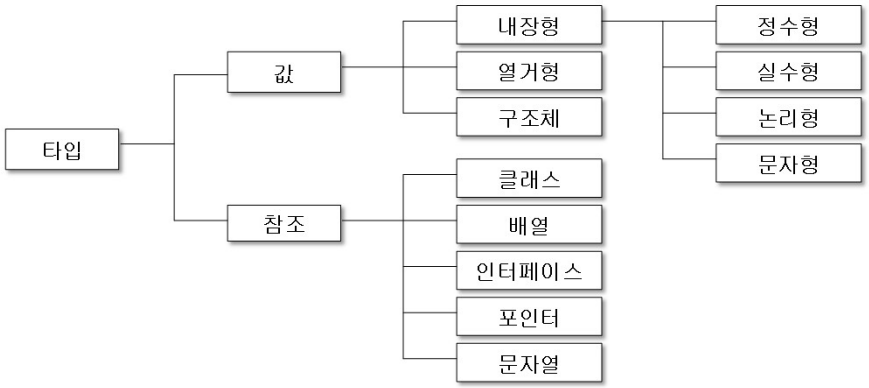
# 타입

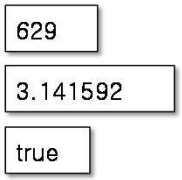
- 타입(Type)은 변수의 형식이며 길이나 값을 해석하는 방식을 결정한다.
- 닷넷은 CTS(Common Type System) 공용 타입 체계를 정의하고 언어는 별칭을 제공한다.

| CTS | 크기 | 부호 | C# | VB | C++ |
|---|---|---|---|---|---|
| System.Byte | 1 | 없음 | byte | Byte | unsigned char |
| System.SByte | 1 | 있음 | sbyte | 없음 | signed char |
| System.Int16 | 2 | 있음 | short | Short | short |
| System.UInt16 | 2 | 없음 | ushort | 없음 | unsigned short |
| System.Int32 | 4 | 있음 | int | Integer | int 또는 long |
| System.UInt32 | 4 | 없음 | uint | 없음 | unsigned |
| System.Int64 | 8 | 있음 | long | Long | __int64 |
| System.UInt64 | 8 | 없음 | ulong | 없음 | unsigned __int64 |
| System.Char | 2 | 없음 | char | char | wchar_t |
| System.Single | 4 | 있음 | float | Single | float |
| System.Double | 8 | 있음 | double | Double | double |
| System.Decimal | 16 | 있음 | decimal | Decimal | Decimal |
| System.Boolean | 1 | | bool | Boolean | bool |
| System.String | 가변 | | string | String | string |
| System.Object | 가변 | | object | Boolean | Object * |

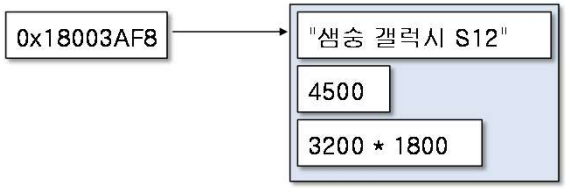
# 타입

- **CTS 타입이나 별칭으로 선언한다.**
  - System.Int32 Value;
  - int Value;
- **크게 값 타입과 참조 타입으로 분류한다.**
- **값 타입 : 스택에 할당되며 변수가 직접 값을 가진다. 작고 길이가 일정하다.**
- **참조 타입 : 값을 가지지 않고 값이 저장된 위치만 가진다. 크고 길이가 가변적이다.**



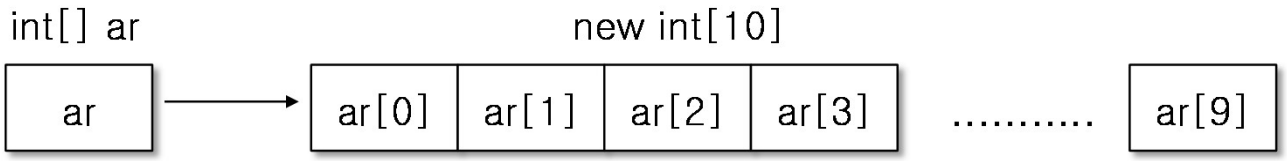

값 타입




참조 타입

# 변수의 선언

- **타입과 이름으로 선언한다.**
  - int i;
- **같은 타입은 한줄에 선언할 수 있으며 = 다음에 초기값을 지정한다.**
  - int i, j = 5;
- **사용하기 전에 반드시 선언해야 한다. 초기화하지 않으면 쓰레기값을 가진다.**
- **참조 타입은 선언 후 바로 사용할 수 없고 new 연산자로 기억장소를 별도로 할당해야 한다.**
  - int[] ar;
  - ar = new int[10];



# 정수형

- **소수점 이하가 없는 정수를 저장한다.**

| 크기 | 부호있음 | 부호없음 |
|---|---|---|
| 1 | sbyte(-128~127) | byte(0~255) |
| 2 | short(-32768~32767) | ushort(0~65535) |
| 4 | int($-2^{31}$~$2^{31}$-1) | uint(0~$2^{32}$-1) |
| 8 | long($-2^{63}$~$2^{63}$-1) | ulong(0~$2^{64}$-1) |

- **부호 없는 정수, long 형 정수는 접미 u, l을 붙인다.**
- **진법에 따라 접두를 붙인다.**
  - int h = 0x1a; // 십진수 26
  - int b = 0b1001; // 십진수 9
- **긴 리터럴은 중간에 _를 넣어 자리를 구분한다.**
  - int price = 12_3456_7890;

# 실수형

- **소수점 이하를 표현할 수 있는 수이다.**

| 타입 | 크기(바이트) | 범위 |
|---|---|---|
| float | 4 | $1.5*10^{-45}$~$3.4*10^{38}$ |
| double | 8 | $5.0*10^{-324}$~$1.7*10^{308}$ |
| decimal | 16 | $1.0*10^{-28}$~$7.9*10^{28}$ |

- **실수를 표현하는 국제 표준 포맷(IEEE 754) 규격을 따른다.**
- **항상 약간의 오차가 발생한다.**
- **실수 상수는 항상 double 타입이며 float나 decimal은 접미 f, m을 붙인다.**
  - float f = 3.14;
- **3.14가 double 상수여서 타입이 맞지 않다. 3.14f라고 적어야 한다.**

# 논리형

- 논리형은 참, 거짓 또는 예, 아니오 두 가지 상태만 가진다.
- 타입 이름은 bool이며 true, false 중 하나의 값을 가진다.
- 다음 코드는 C++에서는 가능하지만 C#에서는 에러이다.
  - if (num = 3) { ... }
- C#은 정수형과 논리형이 호환되지 않는다.
- "value가 0이 아닐 때"에 대한 조건
  - C++ : if (value)
  - C# : if (value != 0)
- 타입 점검이 엄격하고 깐깐해서 더 안전하다.

# 문자형

- 문자형인 char는 16비트 길이의 유니코드 문자 하나를 저장한다.
- 키보드로 입력할 수 없는 문자는 확장열로 표기한다.

| 확장열 | 유니코드 | 설명 |
|---|---|---|
| ₩' | 0x0027 | 홑따옴표 |
| ₩" | 0x0022 | 겹따옴표 |
| ₩₩ | 0x005c | 백슬레시 |
| ₩0 | 0x0000 | 널 |
| ₩a | 0x0008 | 벨 |
| ₩b | 0x0008 | 백스페이스 |
| ₩f | 0x000c | 폼 피드 |
| ₩n | 0x000a | 줄 바꿈 |
| ₩r | 0x000d | 줄 바꿈. 개행 |
| ₩t | 0x0009 | 가로 탭 |
| ₩v | 0x000b | 세로 탭 |
| ₩x16진값 | 16진값 | 유니코드 문자 |

- IsLower, IsUpper, IsDigit, IsNumber 등 정적 메서드로 문자의 종류를 판별한다.
- 문자 하나만 표현하며 문자열은 string 타입을 쓴다.

# 열거형

- 가능한 값의 집합을 정의하는 사용자 정의형 타입이다.
- 가독성에 유리하며 엉뚱한 값을 대입할 위험이 없어 안전하다.
- enum 키워드 다음에 열거 타입의 이름을 쓰고 { } 괄호 안에 열거 멤버를 콤마로 구분하여 나열한다.
  - enum Origin { East, West, South, North }
- 방향은 동서남북 넷 중 하나의 값만 가질 수 있어 열거형으로 정의한다.
- 열거 멤버는 "열거타입.멤버" 식으로 칭한다.
  - Origin Turn;
  - Turn = Origin.South;
- 열거 멤버는 0부터 시작되어 1씩 증가하는 정수값을 가진다. 값을 지정할 수도 있다.
  - enum Origin { East = 1, West = 5, South, North }
- 값이 생략된 멤버는 이전 멤버 + 1의 값을 가진다.

# 열거형

- **열거 타입 이름 다음에 내부 저장 타입을 지정하면 용량을 조금 아낄 수 있다.**
  - enum Origin : byte { East, West, South, North }
- **같은 이름의 멤버를 중복 정의할 수는 없다. 그러나 다른 이름으로 같은 값을 가지는 열거 멤버는 가능하다.**
  - enum Origin { East, East, West, South, North } // 에러
  - enum Origin { East = 1, West = 1, South, North } // 가능
  - enum Origin { East, West, South, North, Dong = East } // 가능
- **정수형과 호환되지만 캐스팅해야 대입할 수 있다.**
  - Turn = (Origin)Value;

# 구조체

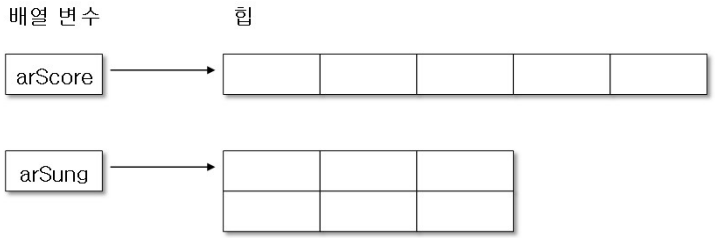
- **구조체는 타입이 다른 변수의 집합이다.**
  - struct 이름 {
  - 멤버 선언문;
  - }
- **C++의 구조체와는 약간 다른 면이 있다.**
  - 멤버의 디폴트 액세스 지정이 private여서 가급적 정보를 은폐한다.
  - 봉인되어 있어 더 이상 파생시킬 수 없으며 단독으로만 사용해야 한다.
- **구조체는 스택에 생성되는 값 타입이어서 선언만 하면 곧바로 사용할 수 있다. 일관성을 위해 new 연산자를 쓸 수도 있다.**
  - Book b = new Book();
- **생성자를 가질 수 있다.**
- **클래스에 비해 기능은 적지만 작고 빠르다.**

# 배열의 선언

- **배열은 동일 타입 변수를 하나의 이름으로 묶어 놓은 것이다.**
  - 타입[ ] 변수명;
- **배열의 차수는 [ ] 괄호 안의 콤마수 + 1이다. 선언예를 보자.**
  - int[] arScore; // 1차원 정수 배열
  - double[] arRate; // 1차원 실수 배열
  - int[,] arSung; // 2차원 정수 배열
  - string[,,] arName; // 3차원 문자열 배열
- **메모리가 허락하는 한까지 다차원 배열을 선언할 수 있다.**

# 배열 할당

- **배열에 요소를 저장하려면 new 연산자로 필요한 양만큼의 메모리를 힙에 할당해야 한다.**
  - arScore = new int[5];
  - arSung = new int[2,3];




- **데이터 저장을 위한 별도의 공간이 힙에 할당되고 배열 변수는 이 위치만 가리킨다.**
- **선언과 할당을 동시에 할 수 있다.**
  - int [] arScore = new int[5];
- **실행중에 할당되므로 변수로 크기를 지정할 수 있다.**
- **실행중에 배열 크기는 변경할 수 없다. 동적 배열이 필요하다면 ArrayList 컬렉션을 사용한다.**

# 배열의 초기값

- **선언과 동시에 초기화할 때는 { } 안에 원하는 초기값을 콤마로 구분하여 나열한다.**
  - int[] arScore = new int[5] { 1, 2, 3, 4, 5 };
- **크기와 타입은 생략 가능하다.**
  - int[] arScore = { 1, 2, 3, 4, 5 };
- **초기값은 최초 할당할 때 딱 한 번만 지정할 수 있으며 할당한 후에는 지정할 수 없다.**
- **2차원 배열도 비슷한 방법으로 초기화한다.**
  - string[,] arCity = {
  - {"서울", "용인", "수원", "의정부"},
  - {"춘천", "홍천", "평창", "양구"},
  - {"대전", "합덕", "논산", "당진"}
  - };
- **중간의 { } 괄호는 생략할 수 없으며 각 행의 초기값 개수는 일치해야 한다.**

# 배열 요소의 참조

- **배열 요소를 참조할 때는 [ ] 괄호 안에 요소의 첨자를 적는다.**
- **첨자는 항상 0부터 시작하며 최대 첨자는 배열 크기보다 1 더 작다.**
  - arScore[1] = 0;
  - arCity[1, 2] = "횡성";
- **배열의 첨자가 범위를 벗어나면 IndexOutOfRangeException 예외가 발생한다.**
- **C++에 비해 안정적이다.**

# 배열의 메서드

- 배열은 System.Array 클래스로 구현되며 이 클래스에는 배열을 관리하는 멤버가 포함되어 있다.

| 멤버 | 설명 |
|---|---|
| GetLength(n) | n 차원의 요소 개수를 조사한다. |
| GetUpperBound(n) | n 차원의 마지막 요소 첨자를 조사한다. 개수보다 항상 1 적다 |
| Length | 배열 요소의 총 개수를 조사한다. 모든 차수의 곱과 같다. |
| Rank | 배열의 차수를 조사한다. |
| Sort | 배열 요소를 크기순으로 정렬한다. 일정 범위의 요소만 정렬할 수도 있다. |
| Reverse | 배열 요소의 순서를 반대로 뒤집는다. 일정 범위의 요소만 뒤집을 수도 있다. |
| BinarySearch | 이분 검색으로 요소를 찾는다. 검색된 경우 그 첨자가 리턴된다. 이 메서드를 호출하려면 배열이 정렬되어 있어야 한다. |
| Clear | 지정한 범위의 요소를 삭제하여 기본값으로 만든다. |

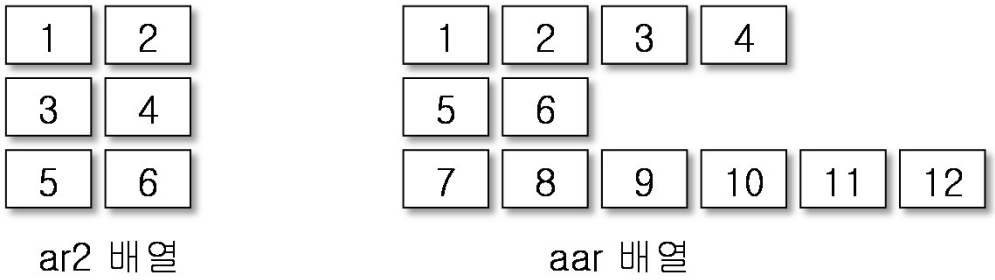
- 배열 요소의 개수를 조사하는 Length 프로퍼티를 자주 쓴다.
- 배열을 순회할 때 상수를 쓰지 말고 Length를 사용해야 한다.

```
for (int i = 0; i < ar.Length; i++) {
    Console.WriteLine(ar[i]);
}
```

- 배열을 정렬할 때는 Sort 메서드를 쓴다.

# 배열의 배열

- **배열끼리 중첩된 형태를 배열의 배열이라고 부른다.**
  - int[,] ar2; // 다차원 배열
  - int[][] aar; // 배열의 배열
- **다차원 배열에 비해 배열의 배열은 부분 배열의 개수를 다르게 지정할 수 있다.**




- **메모리를 절약할 수 있지만 자주 사용되지는 않는다.**

# 클래스

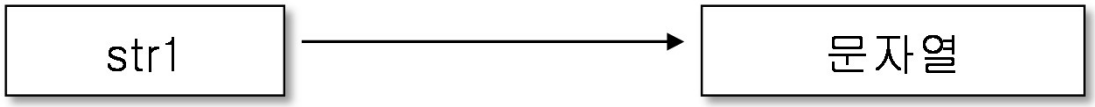
- 클래스는 객체지향 프로그래밍의 중심 개념이다.
- 속성을 표현하는 데이터와 동작을 기술하는 함수의 묶음이다.
- 키워드 class로 시작하며 클래스 이름을 지정하고 { } 블록 안에 멤버를 나열한다.

```
class Time {
        public int hour, min, sec;
        public Time(int ahour, int amin, int asec) {
                hour = ahour;
                min = amin;
                sec = asec;
        }
        public void OutTime() {
                Console.WriteLine($"현재 시간은
{hour}시 {min}분 {sec}초이다.");
        }
}
```

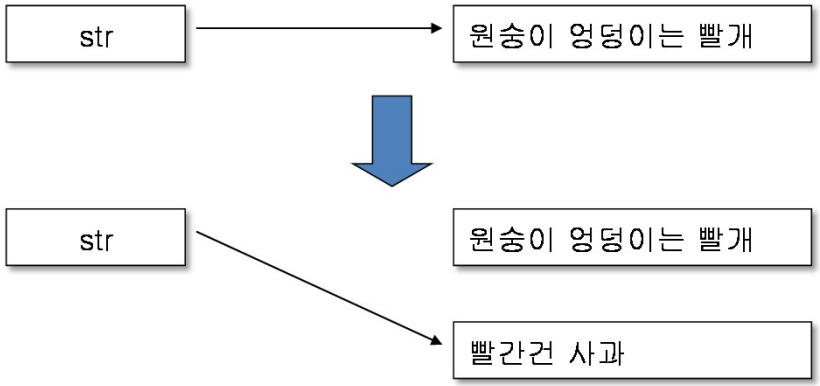
- 선언만으로 사용할 수 없으며 new 연산자로 할당해야 한다.
- 생성과 동시에 객체를 초기화하고 싶다면 생성자를 정의한다.

# 문자열

- 문자열은 string 타입으로 표현한다.
- 참조형이지만 new 연산자없이 선언할 수 있고 값 타입처럼 대입할 수 있다.
- str1 = "문자열"; 대입에 의해 str1이 이 문자열을 가리킨다.




- 읽기 전용이어서 한 번 초기화하면 내용을 변경할 수 없다. 대입시 문자열이 바뀌지 않고 참조만 바뀐다.
  - string str = "원숭이 엉덩이는 빨개";
  - str = "빨간건 사과";
- 필요없어진 문자열은 가비지
- 컬렉터가 삭제한다.



# 문자열

- **문자열 상수는 큰따옴표안에 표현하며 확장열을 모두 사용할 수 있다.**
- **@ 문자를 사용하면 확장열을 치환하지 않고 문자 그대로 해석한다.**
  - string path = "c:₩₩data₩₩file.txt";
  - string path = @"c:₩data₩file.txt";
- **@ 문자는 개행 문자까지도 그대로 해석하여 여러 줄로 된 문자열을 표현할 수 있다.**
  - string str = @"개행된
  - 문자열이다";
- **이때 들여쓰기를 해서는 안된다.**