01. 데이터에도 종류가 있다

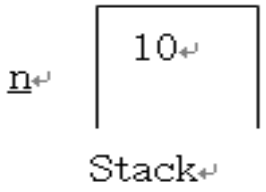
- C#의 데이터 형식 체계는 기본 데이터 형식(Primitive Data Type)과 복합 데이터 형식(Complex Data Type)
  - 기본 데이터 형식 : int, double, decimal …
  - 복합 데이터 형식 : string, Socket, Image …

- 그리고 값 형식과 참조 형식으로 나뉨
  - 값 형식 : 변수에 데이터의 값을 담는 형식
  - 참조 형식 : 변수에 데이터의 위치를 담는 형식

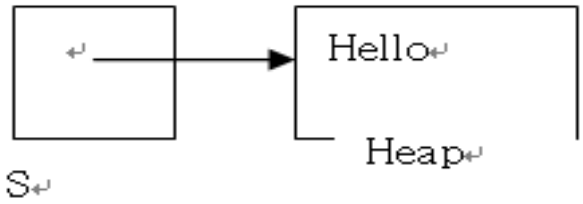
01. 데이터에도 종류가 있다

CTS(Common Type System:공통형식체계) - .NET 공통 type

1, Value Type
    Struct : int n=10;
    Enum



2, Reference Type
    Class : String s = "hello";
    Interface
    Delegete



** Vaule type **

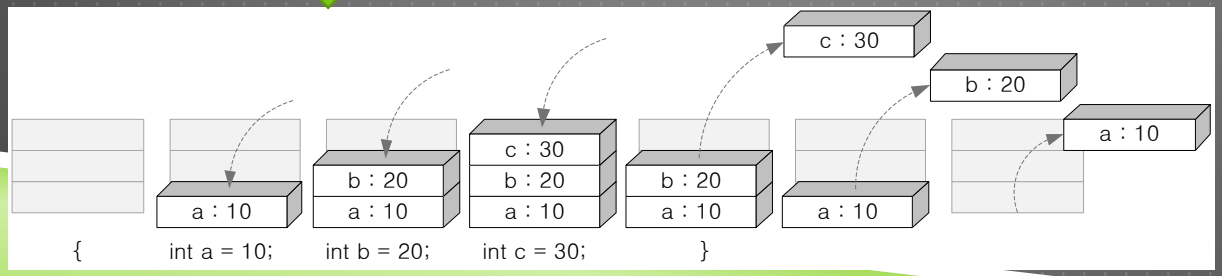
sbyte - 1byte(-128~127)
byte - 1byte(0~255)
double - 8byte 실수형(소수이하 정밀화)
decimal - 16byte(정수부분 정밀화, decimal a=3.14M)

char - 2byte 저장(unicode를 사용하므로)
    char charVaule='金';

03. 값 형식과 참조 형식 (1/2)

- 값 형식 : 변수가 값을 담는 데이터 형식
- 값 형식 변수는 스택에 할당되고 변수 생성 코드 범위를 벗어나면 해제됨.

```
{ // 코드 블록 시작
    int a = 100;
    int b = 200;
    int c = 300;
} // 코드 블록 끝
```



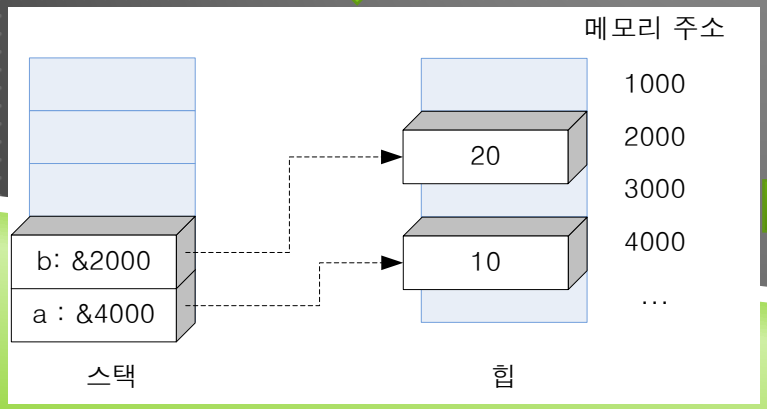
03. 값 형식과 참조 형식 (2/2)

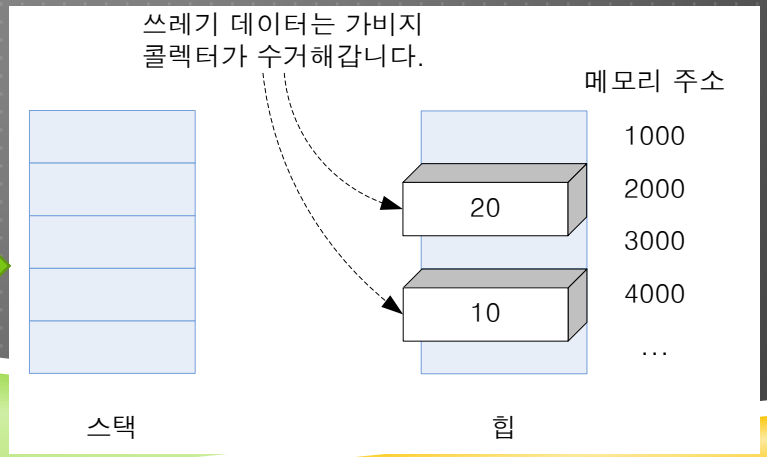
- 참조 형식 : 변수가 값 대신 값이 있는 곳의 위치(참조)를 담는 데이터 형식
- 참조 형식 변수는 힙에 할당되고 가비지 컬렉터에 의해 해제됨.

```
{
    object a = 10;
    object b = 20;
}
```

c/c++ : malloc() , free()
new, delete
>> 즉시 지워짐, 메모리 누수 발생가능성 존재

c#/java: new, GC(가베지컬렉터)
>> 메모리 누수 발생없음, 즉시 지워지지 않음





04. 기본 데이터 형식 (1/15)

- 정수 계열 형식(Integral Types) (1/3)

데이터 형식	설명	크기(Byte)	담을 수 있는 값의 범위
byte	부호 없는 정수	1 (8bit)	0 ~ 255
sbyte	signed byte 정수	1 (8bit)	-128 ~ 127
short	정수	2 (16bit)	-32,768 ~ 32,767
ushort	unsigned short 부호 없는 정수	2 (16bit)	0 ~ 65535
int	정수	4 (32bit)	-2,147,483,648 ~ 2,147,483,647
uint	unsigned int 부호 없는 정수	4 (32bit)	0 ~ 4294967295
long	정수	8 (64bit)	-922337203685477508 ~ 922337203685477507
ulong	unsigned long 부호 없는 정수	8 (64bit)	0 ~ 18446744073709551615
char	유니코드 문자	2 (16bit)	

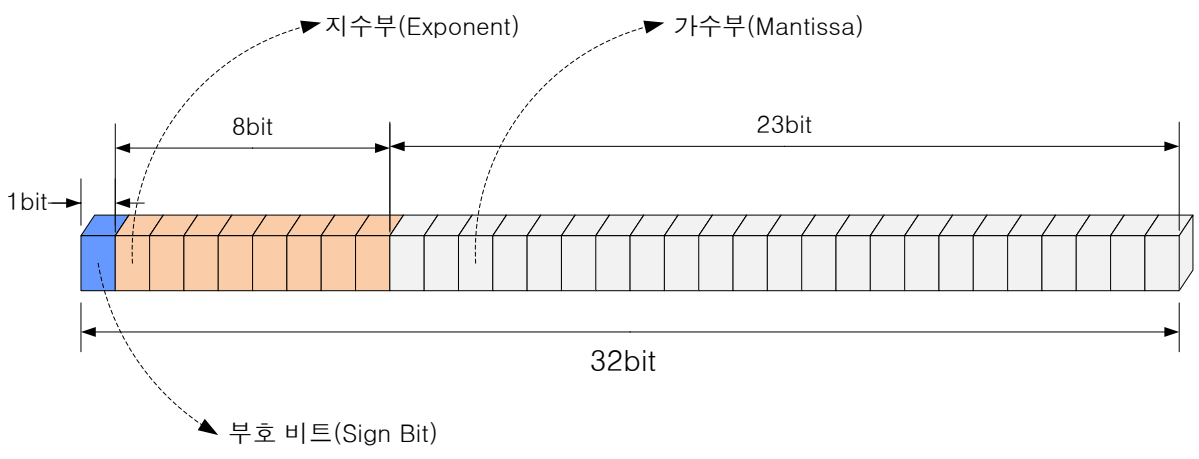
04. 기본 데이터 형식 (4/15)

- 부동 소수점 형식(Floating Point Types) (1/2)
  - "부동"이라는 말은 뜰 부(浮), 움직일 동(動), 즉 떠서 움직인다는 뜻
  - 부동 소수점이라는 이름은 소수점이 고정되어 있지 않고 움직이면서 수를 표현한다는 뜻에서 지어진 이름
    - 소수점을 이동시켜 수를 표현하면 고정시켰을 때보다 보다 제한된 비트를 이용해서 훨씬 넓은 범위의 값을 표현할 수 있음.

데이터 형식	설명	크기(byte)	범위
float	단일 정밀도 부동 소수점 형식 (7개의 자릿수만 다룰 수 있음.)	4 (32bit)	-3.402823e38 ~ 3.402823e38
double	복수 정밀도 부동 소수점 형식 (15~16개의 자릿수를 다룰 수 있음.)	8 (64bit)	-1.79769313486232e308 ~ 1.79769313486232e308

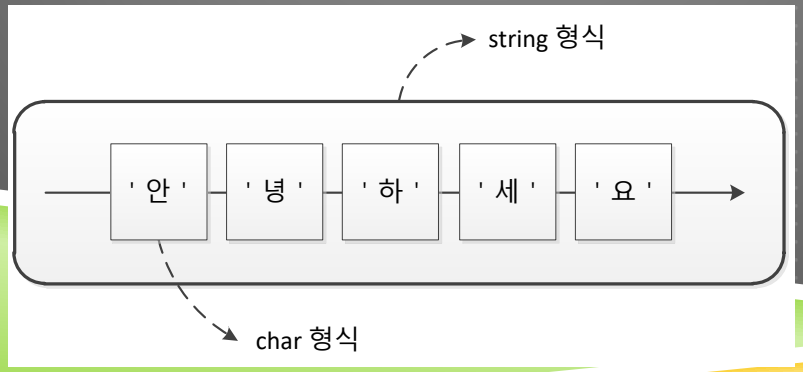
04. 기본 데이터 형식 (5/15)

- 부동 소수점 형식(Floating Point Types) (2/2)
  - IEEE754
    - 4바이트(32비트) 크기의 float 형식은 수를 표현할 때
      - 1비트를 부호 전용으로 사용
      - 가수부 23비트를 수를 표현하는 데 사용
      - 나머지 지수부 8비트를 소수점의 위치를 표현하는데 사용



04. 기본 데이터 형식 (6/15)

- 문자 형식과 문자열 형식
  - char 형식(문자 데이터)에 데이터를 담을 때는 작은 따옴표 '와 '로 감싸줌(2byte)
    예) char a = '가';
    char b = 'a';
  - 문자열(**文字列**, string): 문자들이 연속해서 가지런히 놓여있는 줄
    예) string a = "안녕하세요?";
    string b = "박상현입니다.";



04. 기본 데이터 형식 (7/15)

- 논리 형식
  - 논리 형식이 다루는 데이터는 참(true)과 거짓(false) 두가지 뿐

데이터 형식	설명	크기(byte)	범위
bool	논리 형식	1 (8bit)	true, false

  - 논리 형식 없는 언어(예:C)에서 거짓과 참을 0과 0이 아닌 수로 대신
    - 나중에 코드를 읽을 때 거짓을 의미하는지 또는 숫자 0을 의미하는지 혼란
    - C#은 이런 문제를 논리 형식을 별도로 도입함으로써 해결

04. 기본 데이터 형식

- 기본 자료형 이해 : SimpleType1
- 자료형 형변한 : SimpleType2
- 데이터 입력 : ConsoleInput
- 데이터 출력 : ConsoleOutput

04. 변수명 규칙

- 첫문자 숫자안 됨, 다음 문자는 unicode문자 가능, 키워드 사용불가능

- **Casing Converntions (대소문자 관례)**

1) **PascalCasing** : 첫글자 대문자, 새로운 단어 시작 시 대문자

권장사항 : namespace명, class명, 구조체, 열거형, 대리자, 인터페이스, 메서드, 속성, 이벤트, 기타 Public 멤버

예) void InitializeDate();

2) **camelCasing**(캐멀,낙타표기법) : 첫글자 소문자, 새로운 단어 시작 시 대문자

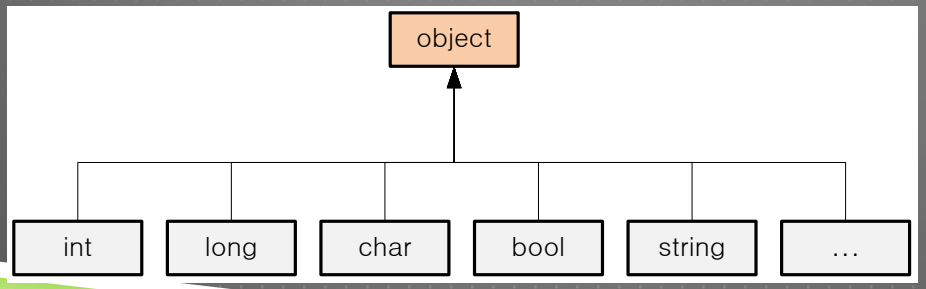
권장사항 : loopCasing (변수, 지역변수, 매개변수, 필드)

예) int loopCountMax(); //첫 글자 소문자, 새로운 단어 시작 시 대문자

지역변수 : 메서드 안에 선언.(스택에 저장) 값을 저장하지 않고 출력하면 오류

04. 기본 데이터 형식 (8/15)

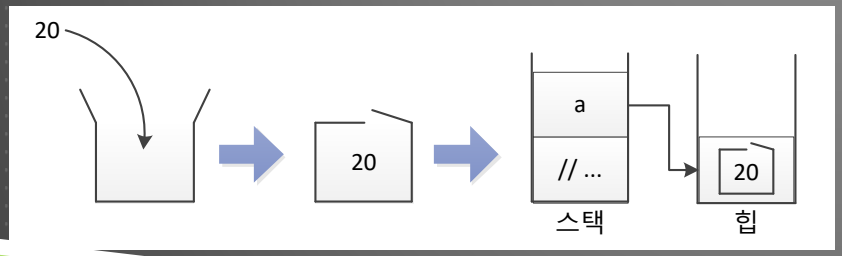
- object 형식 (1/3)
  - 어떤 형식이든 다룰 수 있는 데이터 형식
  - 참조 형식
  - 부모 클래스로부터 파생된 자식 클래스는 부모 클래스와 동일하게 취급하는 "상속"을 이용한 데이터 형식
    - C#의 모든 데이터 형식은 object 형식으로 파생받음
    - 즉, C#의 모든 데이터 형식은 object 형식으로 다룰 수 있음.



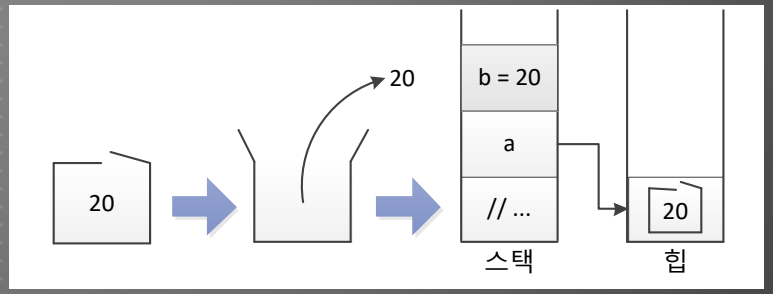
04. 기본 데이터 형식 (10/15)

- object 형식 (3/3)
  - Boxing과 Unboxing
    - Boxing : 값 형식 데이터를 상자에 담아 힙에 올려놓고, 그 힙의 위치를 object 형식 변수가 가리키도록 하는 것
    - Unboxing : 힙 안에 올라가 있는 상자를 풀어 값형식 데이터를 꺼내는 것

Boxing 예)
object a = 20;



Unboxing 예)
object a = 20;
int b = (int)a;



05. 상수와 열거 형식 (1/2)

- 상수(Constant)
  - 변수와는 달리 그 안에 담긴 데이터를 절대 바꿀 수 없는 메모리 공간
  - 변수를 선언하고 프로그래머가 바꾸지 않으면 되지, 왜 상수가 필요한가?
    - 프로그래머는 실수를 하는 사람이기 때문에
    - 아무리 똑똑하고 꼼꼼해도 코드가 수천, 수만 라인에 이르면
      어떤 변수를 바꿔도 되는지 또 어떤 변수는 바꾸면 안되는지 잊기 시작
  - 상수 선언 방법 : 아래와 같이 const 키워드를 이용하여 선언

```
const int a = 3;
const double b = 3.14;
const string c = "bcdef";
```

이렇게 선언한 상수를 변경하려는 코드가 있을 때, 컴파일러가 에러를 일으킴

05. 상수와 열거 형식 (2/2)

- 열거형(Enumeration)
  - 열거된 형태의 상수
  - enum 키워드를 이용하여 선언

상수

```
const int RESULT_NO = 2;
const int RESULT_CONFIRM = 3;
const int RESULT_CANCEL = 4;
const int RESULT_OK = 5;
```

VS

열거형

```
enum DialogResult { YES, NO, CANCEL, CONFIRM, OK }
```

중복된 값을 가진 상수를 선언할 위험도 없고, 읽기에도 편함

06. NULLABLE 형식

- C# 컴파일러는 변수의 메모리 공간에 반드시 어떤 값이든 넣도록 강제함
- 하지만 프로그래밍을 하다 보면 어떤 값도 가지지 않는 변수가 필요할 때가 생김. 0이 아닌, 정말 비어있는 변수, 즉 null한 변수가 필요할 때가 생김
- Nullable 형식은 이런 경우를 위해 사용
  - 데이터 형식 뒤에 '?'만 붙여주면 끝

    예) int? a = null;

    float? b = null;

    double? c = null;

07. VAR: 자동 형식 지정

- C#은 강력한 형식 검사를 하는 언어이지만, 약한 형식 검사를 하는 언어의 편리함도 지원
- int, string 같은 명시적 형식 대신 **var** 를 사용해서 변수를 선언하면 컴파일러가 자동으로 해당 변수의 형식을 지정

```
예) var a = 3;          // a는 int 형식
    var b = "Hello";    // b는 string 형식
```

08. 공용 형식 시스템 (CTS) (1/2)

- C#의 모든 데이터 형식 체계는 사실 C# 고유의 것이 아님
- .NET 프레임워크의 형식 체계의 표준인 공용 형식 시스템(Common Type System)을 따르고 있을 뿐임
- 공용 형식 시스템의 이름은 "모두가 함께 사용하는 데이터 형식 체계"라고 뜻. 즉, 공용 형식 시스템은 .NET 언어들 이라면 반드시 따라야 하는 데이터 형식 표준
- 마이크로소프트가 "공용"형식 시스템을 도입한 이유는 .NET 언어들끼리 서로 호환성을 갖도록 하기 위해서임
- 우리가 앞에서 살펴봤던 모든 데이터 형식에 관련한 이야기가 곧 CTS의 이야기

08. 공용 형식 시스템 (CTS) (2/2)

- CTS 형식 표

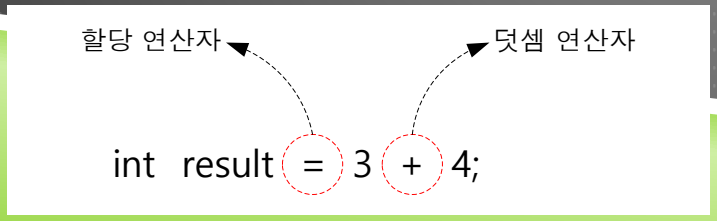
Class name	C# 형식	C++ 형식	비주얼 베이직 형식
System.Byte	byte	unsigned char	Byte
System.SByte	sbyte	char	SByte
System.Int16	short	short	Short
System.Int32	int	int 또는 long	Integer
System.Int64	long	__int64	Long
System.UInt16	ushort	unsigned short	UShort
System.UInt32	uint	unsigned int 또는 unsigned long	UInteger
System.UInt64	ulong	unsigned __int64	ULong
System.Single	float	float	Single
System.Double	double	double	Double
System.Boolean	bool	bool	Boolean
System.Char	char	wchar_t	Char
System.Decimal	decimal	Decimal	Decimal
System.IntPtr	없음.	없음.	없음
System.UIntPtr	없음.	없음.	없음
System.Object	object	Object*	Object
System.String	string	String*	String

01. C#에서 제공하는 연산자 둘러보기

- C#이 제공하는 주요 연산자의 목록

분류	연산자
산술 연산자	+, -, *, /, %
증가/감소 연산자	++, --
관계 연산자	<, >, ==, !=, <=, >=
조건 연산자	?:
논리 연산자	&&, ||, !
비트 연산자	<<, >>, &, |, ^, ~
할당 연산자	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=

- 연산자 사용 예



03. 증가 연산자와 감소 연산자 (1/2)

- 증가 연산자는 피연산자의 값을 1 증가
- 감소 연산자는 피연산자의 값을 1 감소

연산자	이름	설명	지원 형식
++	증가 연산자	피연산자의 값을 1 증가시킵니다.	모든 수치 데이터 형식과 열거 형식
--	감소 연산자	피연산자의 값을 1 감소시킵니다.	모든 수치 데이터 형식과 열거 형식

- 사용 예

```
int a = 10;
a++; // a는 11
a--;  // a는 10
```

07. 조건 연산자

- 조건 연산자(Conditional Operator) ?:는 조건에 따라 두 값 중 하나의 값을 반환
  - 사용 형식

```
조건식 ? 참일_때의_값 : 거짓일_때의_값
```

  - 사용 예

```
int a = 30;
string result = a == 30 ? "삼십" :"삼십아님" ;   // result는 "삼십"
```

10. 연산자의 우선 순위

우선 순위	종류	연산자
1	증가/감소 연산자	후위 ++/-- 연산자
2	증가/감소 연산자	전위 ++/-- 연산자
3	산술 연산자	* / %
4	산술 연산자	+ -
5	시프트 연산자	<< >>
6	관계 연산자	< > <= >= is as (is와 as 연산자는 뒷부분에서 설명합니다.)
7	관계 연산자	== !=
8	비트 논리 연산자	&
9	비트 논리 연산자	^
10	비트 논리 연산자	|
11	논리 연산자	&&
12	논리 연산자	||
13	조건 연산자	?:
14	할당 연산자	= *= /= %= += -= <<= >>= &= ^= |=

02. 반복문 (1/8)

- 반복문(Loop Statement)
  - 특정 조건을 만족하는 동안 코드 또는 코드 블록을 계속 반복해서 실행하도록 하는 문장
  - C#은 다음 네 가지의 반복문을 제공
    - while
    - do while
    - for
    - foreach

03. 점프문 (1/3)

- break
  - break는 이름(영어로' 탈출하다',' 중단하다'라는 뜻) 그대로 현재 실행 중인 반복문이나 switch 문의 실행을 중단하고자 할 때 사용

```
int i = 0;  // i를 초기화하는 코드가 실행되고

while ( i >= 0) // 반복이 실행되다가
{
    if ( i == 10 )
        break;     // i가 10이 되면 while 문에서 탈출

    Console.WriteLine( i++ );
}

// 프로그램의 실행 위치는 while 블록 다음으로 이동
Console.WriteLine("Prison Break");
```

03. 점프문 (2/3)

- continue
  - 반복문을 멈추게 하는 break와는 달리, continue 문은 한 회 건너 뛰어 반복을 계속 수행

```
for ( int i=0; i<5; i++ )
{
    if ( i == 3 )
        continue;
    Console.WriteLine( i );
}
```

가독성이 더 뛰어남

VS

```
for ( int i=0; i<5; i++ )
{
    if ( i != 3 )
    {
        Console.WriteLine( i );
    }
}
```

03. 점프문 (3/3)

- goto
  - 지정한 레이블로 바로 코드의 실행을 이동하는 점프문
  - 형식

```
goto 레이블;

레이블 :
   // 이어지는 코드
```

가독성이 더 뛰어남

  - 사용 예제

```
{
    Console.WriteLine( " 1 " );
    goto JUMP;
    Console.WriteLine( " 2 " );
    Console.WriteLine( " 3 " );

JUMP:
    Console.WriteLine( " 4" );
}
```

실행 결과 :

1
4