[과목] C#

[챕터] Chapter 2. 처음 만드는 C# 프로그램

[소제] 2.2 첫 번째 프로그램 뜯어보기

# [내용]

using System;

using static System.Console;

namespace Hello

{

    class MainApp{

        static void Main(string[] args){

            if(args.Length == 0){

                WriteLine("사용법 : Hello.exe<이름>");

                return;

            }

            WriteLine("Hello, {0}!", args[0]);

        }

    }

}

## 2.2.1 using System;

using 은 C#의 키워드 중 하나. 키워드는 C# 언어의 규격에 미리 정의되어 있는 특별한 단어이다.

System 은 C# 코드가 기본적으로 필요로 하는 클래스를 담고 있는 네임스페이스이다. using System;은 System 네임스페이스 안에 있는 클래스를 사용하겠다고 컴퍼일러에 알리는 역할이다.

## 2.2.2 using static System.Console;

using 키워드만 사용하면 네임스페이스 전체를 사용한다는 의미지만 using static 은 어떤 데이터 형식의 정적 멤버를 데이터 형식의 이름을 명시하지 않고 참조하겠다고 선언하는 기능이다.

## 2.2.3 namespace Hello{}

네임스페이스는 성격이나 하는 일이 비슷한 클래스, 구조체, 인터페이스, 대리자, 열거 형식 등을 하나의 이름 아래 묶는 일을 한다.

네임스페이스를 만들 때는 다음과 같이 namespace 키워드를 이용하며, 네임스페이스\_ 이름 뒤에 따라오는 괄호 {와} 사이에는 이 네임스페이스에 소속되는 클래스 등이 들어간다.

namespace 네임스페이스\_이름

{

    //클래스

    //구조체

    //인터페이스 등...

}

## 2.2.4 class MainApp{}

class MainApp 구문은 MainApp 이라는 이름의 클래스를 만든다. 클래스는 C# 프로그램(C# 언어로 만드는 프로그램)을 구성하는 기본 단위로서 데이터와 데이터를 처리하는 기능(메소드 Method)으로 이루어진다.

2.2.5 staitc void Main(string[] args) {}

# staitc void Main(string[] args){}은 메소드이다. 프로그램의 진입점으로서 프로그램을

시작하면 실행되고, 이 메소드가 종료되면 프로그램도 종료된다.

모든 프로그램은 반드시 Main 이라는 이름을 가진 메소드를 하나 가지고 있어야 한다. staitc 은 한정자로서 메소드나 변수 등을 수식한다. C# 프로그램의 각 요소는 코드가 실행되는 시점에 비로소 메모리에 할당되는 반면, static 키워드로 수식되는 코드는 프로그램이 처음 구동될 때부터 진작에 메모리에 할당된다.

프로그램이 실행되면 CLR(Common Language Runtime)은 프로그램을 메모리에 올린 후 프로그램의 진입점을 찾는데, 이때 Main() 메소드가 static 키워드로 수식되어 있지 않다면 CLR 은 진입점을 찾지 못했다는 에러 메시지를 남기고 프로그램을 종료시킬 것이다. void 는 메소드의 반환 형식이다. 메소드가 어떤 결과도 돌려주지 않을 것이다. Main 은 메소드의 이름이며, 메소드 이름 뒤에 있는 괄호와 그 사이에 있는 코드는 메소드에 입력되는 매개변수이다.

## 2.2.6 if(args.Length == 0){}

if 문은 조건을 평가해서 프로그램의 흐름을 결정하는 코드인데, 매개변수 목록의 길이가 0(args.Length == 0)일 때 안내문을 출력하고 프로그램을 종료한다.

[챕터] Chapter 3. 데이터 보관하기

[소제] 3.2 변수

# [내용]

변수를 코드에서 보자면 값을 대입시켜 변화시킬 수 있는 요소이지만, 메모리 쪽에서 보면 “데이터를 담는 일정 크기의 공간”이라는 의미를 갖기도 한다.

변수는 데이터 형식을 명시하고 그 다음에 변수 식별자(이름)을 명시한다.

int x;

선언과 데이터 할당을 동시에 할 수도 있다.

int x = 100;

변수 여러 개를 동시에 선언할 수도 있다.

int a, b, c;

[소제] 3.3 값 형식과 참조 형식

[내용]

값 형식(Value Types)은 변수가 값을 담는 데이터 형식을 말하고, 참조 형식(Reference Types)은 변수가 값 대신 값이 있는 곳의 위치(참조)를 담는 데이터 형식을 말한다.

값 형식 = 스택 메모리 영역

참조 형식 = 힙 메모리 영역

3.3.1 스택과 값 형식

후입선출 : 나중에 들어온 것이 가장 먼저 나간다

값 형식의 변수는 모두 스택에 저장된다. 코드 블록 안에서 생성된 모든 값 형식의 변수들은 프로그램 실행이 중괄호 }를 만나면 메모리에서 제거된다.

{ // 코드 블록 시작

    int a = 100;

    int b = 200;

    int c = 300;

} // 코드 블록 끝

.

3.3.2 힙과 참조 형식

힙은 저장된 데이터를 스스로 제거하는 메커니즘을 갖고 있지 않다.

대신 CLR의 가비지 컬렉터(Garbage Collector)가 있다. 가비지 컬렉터는 프로그램 뒤에 숨어 동작하면서 힙에 더 이상 사용하지 않는 객체가 있으면 그 객체를 쓰레기로 간주하고 수거하는 기능을 한다.

힙을 사용하는 이유 : 코드 블록이 끝나는 시점과 상관없이 데이터를 유지하고 싶을 때.

{

    object a = 10;

    object b = 20;

}

위 코드에서 }를 만나는 곳에서 a와 b는 스택에서 사라진다. 하지만 힙에는 여전히 10과 20이 남아 있다.

힙은 더 이상 데이터를 참조하는 곳이 없을 때 가비지 컬렉터가 데이터를 치워주는 구조의 메모리 영역이다.

[소제] 3.4 기본 데이터 형식

[내용]

C#이 제공하는 기본 데이터 형식에는 모두 15가지가 있는데, 이들은 크게 숫자 형식, 논리 형식, 문자열 형식, 그리고 오브젝트 형식으로 나뉜다. 이 중에서 문자열 형식과 오브젝트 형식만 참조 형식에 해당하며, 나머지는 모두 값 형식이다.

3.4.1 숫자 데이터 형식

C#은 15가지 기본 자료 형식 중 12가지를 숫자 데이터 형식으로 제공한다. 이 12가지 형식은 정수 계열, 부동 소수 계열, 소수 계열로 나뉜다.

정수 계열 형식 : 정수 데이터를 담기 위해 사용한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 데이터 형식 | 설명 | 크기(바이트) | 담을 수 있는 값의 범위 |
| byte | 부호 없는 정수 | 1(8비트) | 0~255 |
| sbyte | signed byte 정수 | 1(8비트) | -128~127 |
| short | 정수 | 2(16비트) | -32,768~32,767 |
| ushort | unsigned short 부호 없는 정수 | 2(16 비트) | 0~65,535 |
| int | 정수 | 4(32 비트) | -2,147,483,648~2,147,483,647 |
| uint | unsigned short 부호 없는 정수 | 4(32비트) | 0~4,294,967,295 |
| long | 정수 | 8(64 비트) | -9,223,372,036,854,775,808~9,223,372,036,854,775,807 |
| ulong | unsigned long 부호 없는 정수 | 8(64 비트) | 0~18,446,744,073,709,551,615 |
| char | 유니코드 문자 | 2(16 비트) |  |

using System;

namespace IntegralTypes

{

    class MainApp

    {

        static void Main(string[] args)

        {

            sbyte a = -10;

            byte b = 40;

            Console.WriteLine($"a = {a}, b = {b}");

            short c = -30000;

            ushort d = 60000;

            Console.WriteLine($"c = {c}, d = {d}");

            int e = -1000\_0000;

            uint f = 3\_0000\_0000;

            Console.WriteLine($"e = {e}, f= {f}");

            long g = -5000\_0000\_0000;

            ulong h = 200\_0000\_0000\_0000\_0000;

            Console.WriteLine($"g = {g}, h= {h}");

        }

    }

}

부호 있는 정수와 부호 없는 정수 : 2의 보수법

1. 해당 숫자를 수 부분 비트에 입력
2. 나머지 비트 전체를 1은 0으로, 0은 1로 반전
3. 반전된 비트에 1을 더한다.
4. using System;
5. namespace SignedUnsigned
6. {
7. class MainApp
8. {
9. static void Main(string[] args)
10. {
11. byte a =255;
12. sbyte b = (sbyte)a;
13. Console.WriteLine(a);
14. Console.WriteLine(b);
15. }
16. }
17. }

오버플로 : 데이터 형식의 크기를 넘어선 값을 담으면 넘친다.

using System;

 namespace Overflow

 {

    class MainApp

    {

        static void Main(string[] args)

        {

            uint a = uint.MaxValue;

            Console.WriteLine(a);

            a = a + 1;

            Console.WriteLine(a);

        }

    }

 }

uint의 최대값은 4294967295이다. 이것은 2진수로 바꿀 때 1111 1111이 된다. 이것에 1을 더하면 1 0000 0000이 되기에 8비트만 담을 수 있는 uint는 1을 버리고 0000 0000만 저장하기에 결국 값은 최저 값인 0이 나온다.

3.4.2 부동 소수점 형식