**1. 메인 함수**

* **시작점(Entry Point)**
* C# 프로그램은 반드시 어떤 메소드에서부터 실행되어야 함  
  - 그 메소드가 Main() 메소드
* 컴퓨터가 어디서부터 코드를 시작해야 하는지 알려주는 지점
* 실행 파일을 실행하면, 이 메소드가 자동으로 실행됨
* static  
  - 메인 메소드는 단일해야 하므로  
  - 메인 메소드는 어디서든 접근 가능해야 하므로
* string[] args  
  - 메인 메소드가 호출(실행 파일이 실행)되면 외부로부터 받는 데이터  
  ex) 파일 복사 프로그램 실행 시 – 원본 파일, 파일을 복사할 디렉토리 – 를 전달
* void  
  - 메인 메소드는 void 형식, 즉 아무것도 반환하지 않음
* 메인 메소드의 경우, 정수 값을 반환하게 할 수 있음  
  - 커맨드 라인은 이 반환형을 받아서 exe 프로그램이 올바르게 실행됐는지 여부를 알 수 있음  
  - 0: 대부분 정상적인 종료를 의미  
  - 0 외의 값: 비정상적인 종료를 의미(오류 발생)
* **Console.WriteLine();**
* 화면에 글자를 출력할 때 사용하는 메소드
* 콘솔(명령 프롬프트)에 한 줄의 글을 쓰는 기능
* 컴퓨터가 이해하는 ‘기계어’로 변환하는 과정이 필요함
* **using System;**
* using 지시어(directive)  
  - C#에서는 using을 지시어라고 함  
  - using 지시어 뒤에는 사용할 라이브러리 이름을 넣음
* 라이브러리(library)  
  - 함수들을 모아 놓은 것을 라이브러리라고 함  
  - C#에서는 네임스페이스(namespace)라고 함  
  - Java에서는 패키지(package)라고 함
* 프로그램에서 사용할 함수들이 들어있는 네임스페이스를 코드의 맨 위에 using 지시어를 사용해서 컴퓨터에게 알려줘야 함

**2. 컴파일(Compile)이란?**

* **컴파일**
* 소스 코드를 기계 또는 VM(Virtual Machine)이 이해할 수 있는 언어로 변경하는 행위
* 컴파일러(compiler)라는 프로그램을 사용  
  - IDE(통합 개발 환경)에 포함되어 있음
* 컴파일을 할 때, 오류(error) 또는 경고(warning)가 발생할 수 있음  
  - 컴파일러는 시간을 충분히 들여서 코드를 분석해 줄 수 있음
* **컴파일 오류(Error)**
* 작성한 코드가 프로그래밍 언어의 규격에 위반되는 경우
* 오류가 있을 경우 컴파일 에러가 발생하며, 실행 파일이 생성되지 않음  
  - 따라서 프로그램을 실행조차 할 수 없음
* 컴파일 오류는 프로그램 실행 중에 문제를 발견하는 것보다 이득임  
  - 프로그램 실행 중에 문제가 있는 것을 **버그**라고 함  
  - 버그가 발생하는 상황을 프로그래머가 직접 찾고 재현해야 함  
  - 버그를 재현한 후에는 프로그래머가 직접 **디버그**해야 함
* **컴파일 경고(Warning)**
* 경고가 있더라도, exe 파일이 생성됨  
  - 따라서 프로그램을 실행은 할 수 있음
* 경고란 프로그래머의 실수처럼 보이는 것을 컴파일러가 찾아준 것  
  - 경고는 실제로는 문제가 아닌 경우가 보통임
* 그러나 경고를 수정하지 않으면, 나중에 진짜 문제를 찾기 매우 어려울 수 있음  
  - 따라서 경고는 곧바로 고치는 것이 좋음  
  - 경고 관리도 매우 중요한 요소임  
  - 실제 업계에서는 릴리즈 빌드 시에 경고를 오류처럼 다루기도 함
* **디버그(Debug) 빌드와 릴리즈(Release) 빌드**
* 디버그 빌드는 개발자가 개발 중에 사용하는 실행파일  
  - 디버깅에 유용한 많은 정보가 담겨 있음(경고나 오류 등)  
  - 최적화는 거의 기대하지 않으며, 따라서 성능도 잘 나오지 않는다
* 릴리즈 빌드는 실제 사용자(고객)에게 배포하는 실행파일  
  - 디버깅을 위한 정보는 적음  
  - 최적화가 잘 되어있음  
   - 디버그 빌드 시의 디버깅에 유용한 체킹 과정이 없으므로  
   - 기능을 바꾸지 않는 선에서, 성능을 향상시키는 코드로 만들어 줌 – 빌드 시의 환경에 맞게  
   - 따라서 성능이 디버깅 모드보다 엄청 빠름

**3. 구문(statement) – 명령을 내리는 단위**

* **한 줄로 이루어진 코드**
* 보통 ‘;(세미콜론)’이 끝에 있는 문장 단위
* **여러 줄로 이루어진 코드 – 블록(block)**
* 범위(scope)라고도 함
* 코드를 중괄호({ })로 감싼 형태

**4. 주석(comment)**

* **코드와 관련된 내용(설명)을 메모할 때 사용**
* **컴파일할 때는 무시되는 문장**
* **오직 사람을 위해서만 사용됨**
* **주석을 지나치게 많이 사용하지 말 것**
* 주석은 코드와 마찬가지로 계속 관리해야 함
* 주석이 너무 많으면 업데이트를 못하는 주석들이 생길 수 있음
* 너무 많으면 오히려 다른 프로그래머들이 코드를 읽는데 불편함
* 반드시, 꼭 필요한 내용만 주석으로 남기는 것이 좋음
* 주석이 없으면 없을수록 좋은 코드임  
  - 코드만으로 다른 사람들이 이해할 수 있다는 것이므로  
  - 코드 자체로 이해가 잘 되는 코드를 작성해야 함
* 소스 코드를 주석 처리하여 남겨놓는 행위는 지양해야 함  
  - ‘혹시라도 필요할지도 모른다’는 생각은 버전 관리 프로그램으로

**5. 상수**

* **상수(constant)**
* 변하지 않는 수(데이터)
* ‘리터럴’이라고도 함, 이 표현이 맞음
* 숫자 뿐만 아니라 문자까지 포함함
* 리터럴, 상수, 매직넘버 등으로 혼용하여 사용하기도 함
* **상수의 별칭 – 매직 넘버(Magic Number)**
* 왜 상수를 매직 넘버라 부르는가?  
  - 마치 마법처럼 (목적이) 무엇인지 전혀 알 수 없기 때문  
  - 따라서 매직 넘버는 되도록이면 사용하지 말 것  
   - 상수를 다른 값으로 바꾸기 위해서 수동으로 직접 그 값을 수정해야 함  
   - 이 경우 변수를 사용하여 관리하는 편이 훨씬 수월하게 작업을 진행할 수 있음