**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. 강한 타입과 약한 타입 언어**

* **강한 타입과 약한 타입의 개요**
* 변수 선언(변수의 용도)과 관련이 있음
* 변수형을 선언한 후에 변수 형식(자료형)을 변경할 수 있는가?  
  예 → 약한 타입, 아니오 → 강한 타입
* **약한 타입**
* 장점  
  - 자료형이 가변적이므로, 사용 시 자료형에 구애받지 않음 - 유연함
* 단점  
  - 코드의 가독성이 떨어짐, 코드의 의도가 불분명해짐  
  - 컴퓨터도 실제로 코드를 실행하기 전까지 실수를 찾을 수 없음
* 따라서 상용 코드를 작성할 때는 강한 타입 언어들이 일반적으로 더 나음
* **강한 타입**
* 한번 선언된 타입(데이터 형식)을 지속적으로 유지함
* 상용 코드나 타인과 협업을 할 때 유용함  
  - 제약적이기에 체계적이므로 규칙이 만들어지기 때문
* 컴파일러가 프로그램의 오류나 실수를 찾아줄 수 있음  
  - 약한 타입에 비해 일찍 오류나 실수를 발견하고 디버그 할 수 있음

**4. 매니지드 언어와 언매니지드 언어**

* **매니지드(Managed) 언어와 언매니지드(Unmanaged) 언어**
* **메모리**의 수명(라이프 사이클)을 누가 어떻게 관리하는지가 핵심
* 프로그램에서 큰 메모리 사용 시도  
  - 운영체제(OS)로부터 독점적으로 사용할 메모리를 확보해야 함
* 프로그램에서 메모리를 더 이상 사용하지 않음  
  - 다시 OS에게 메모리를 반환해야 함
* **매니지드(Managed) 언어**
* 프로그래밍 언어가 자동으로 더 이상 사용하지 않는 메모리를 관리  
  - 자동으로 메모리를 반환해 줌
* 장점  
  - 메모리 관리에 대한 실수가 적고 편리함
* 단점  
  - 메모리 반환 시점을 알 수 없어 명시적 표현보다 효율적이지 못함  
  - 메모리를 반환해야 하는지 모호함 – 프로그램에게 이를 확인하는 단계를 거침
* **언매니지드(Unmanaged) 언어**
* 프로그래머가 직접 수동적으로 메모리를 할당 및 반환하며 관리함
* 장점  
  - 규칙적이고 명시적으로 표현하기 때문에 오해의 소지가 없음
* 단점  
  - 메모리 누수의 위험이 있음
* **적재적소에서 활용**
* 성능이 중요하지 않은 프로그램  
  - 매니지드 언어를 많이 사용함 – Java, C#, JavaScript, Lua
* 성능이 중요한 프로그램  
  - 언매니지드 언어를 많이 사용함 – C, C++
* **언매니지드 언어를 배우면 하드웨어 지식을 쌓을 수 있음**
* 컴퓨터처럼 생각하는 방법을 기를 수 있음 – 하드웨어적 사고는 매우 중요함  
  - 메모리를 직접 관리해야 하기 때문에 – 메모리 제어를 이해
* **언매니지드 언어를 반드시 배워야 하는가?**
* 다음의 것들을 알기 위해 언매니지드 언어를 **꼭** 배워야 함  
  - 메모리 메커니즘, CPU 메커니즘, 컴퓨터적 사고방식, 매니지드 언어의 메커니즘
* 컴퓨터처럼 생각하는 습관이 일찍 들면 빠르게 성장할 수 있음
* 반드시 언매니지드와 하드웨어를 이해하고 알고 있어야 함

**4. 프로그래밍 패러다임: 절차적(Procedural) 언어**

* **절차적 언어**
* 절차, 순서를 핵심으로 하여 실행 후 결과를 도출하는 방식
* 가장 단순한 형태의 언어
* 함수를 실행하면 결과가 바로 반환됨
* 함수의 결과를 저장하고 싶으면 별도의 공간에 보관함
* 따라서 기능과 자료 저장이 분리되어 있음  
  - 연산은 CPU가, 데이터는 메모리에
* C언어가 가장 대표적

**5. 프로그래밍 패러다임: OOP(Objected-Oriented Programming) 언어**

* **객체지향 언어**
* 속성(상태)과 기능을 하나의 개체(Object)에 합침  
  - 물체의 개념  
  - 이러한 개체를 하나의 구성 단위로 바라보는 시각
* 가장 널리 쓰이는 프로그래밍 패러다임  
  - 사람에게 매우 직관적 – 하나의 사물, 물체로 바라봄  
  - 복잡한 프로그램의 구조를 잡기 쉬움

**6. 프로그래밍 패러다임: 함수형(Functional) 언어**

* **함수형 언어**
* 절차적 언어처럼 함수가 있음  
  - 함수를 절차적으로 실행하여 결과를 획득함
* 연산은 동일하게 함수에서 진행하되, 저장한 데이터는 읽을 수만 있으며 바뀔 수 없다.  
  - 그러나 상태의 변화가 없음 – 불변성(Immutable)

**7. C#은 어떤 언어인가?**

* **강한 타입 언어**
* 한번 선언한 타입을 다른 타입으로 바꿀 수 없음
* **매니지드 언어**
* 메모리를 자동으로 관리해줌
* **OOP(개체 지향 프로그래밍)**
* 프로그래밍 입문에만 집중하기 위해 주로 절차적 언어처럼 사용할 것임
* 절차적 언어는 OOP를 배우기 전에 알아야 할 기본 지식
* 중간고사 후에는 OOP를 간략하게 알아볼 예정