**1. 저수준 언어와 고수준 언어**

* **저수준 언어(Low-Level)**
* 컴퓨터가 실질적으로 이해하는 언어
* 2진수(바이너리)로 이루어져 있음
* 이를 기계어(machine language)라고 함
* 기계어는 컴퓨터가 바로 이해하므로 별도의 변환 과정이 필요 없음
* **고수준 언어(High-Level)**
* 사람이 이해할 수 있는 언어
* 인간이 사용하는 언어로 이루어져 있음
* 컴퓨터가 이해하는 ‘기계어’로 변환하는 과정이 필요함
* **고수준 저수준은 상대적인 것**
* 컴퓨터가 이해하기 쉬운 언어에 가까울수록 ‘저수준’
* 인간이 이해하기에 쉬운 언어에 가까울수록 ‘고수준’
* **기계어(Machine Language)**
* 네이티브 코드(native code)라고도 함
* 2진수로 이루어져 있음
* 가장 낮은 수준(저수준)의 언어
* 모든 프로그래밍 언어는 기계어로 번역됨
* **어셈블리어(Assembly Language)**
* 기계어보다 사람이 이해하기 그나마 쉬운 언어
* 엄밀하게는 고수준 언어에 속함
* 기계어와 일대일 대응이 가능함 – 이 때문에 보통 저수준 언어로 취급됨
* **고수준 프로그래밍 언어(High-Level Programming Language)**
* 사람이 사용하는 언어(영어)로 구성되어 있음
* 그러나 기계어와 일대일 대응이 불가능함
* 메모리를 누가 관리하는가에 따라서 ‘매니지드’와 ‘언매니지드’ 언어로 나뉨

**2. 컴파일언어와 인터프리터 언어**

* **컴파일 언어(Compiled Language)**
* 소스 코드(고수준 언어 코드) → 컴파일러(complier) → 기계어 → 실행
* ‘컴파일’ 이라는 것은 소스 코드를 기계어로 만들어주는 행동을 의미
* 오프라인(offline) 컴파일러가 고수준 언어를 네이티브 코드로 컴파일
* 컴파일 언어의 동작 방식 - 실행 파일(exe 확장자)은 기계어 상태로 있음, 실행되면 기계어를 읽어서 실행
* 다양한 최적화가 가능  
  - 컴파일 1시간 – 컴파일러가 소스코드를 기계어로 컴파일 할 때 자동으로 최적화(단, 시간이 소요됨)  
  - 실행 0.1초 – 최적화된 실행 속도를 획득
* 플랫폼마다 다른 exe 파일을 컴파일해야 함  
  - CPU마다 이해할 수 있는 기계어가 다를 수 있음 → CPU(플랫폼) 각각의 exe 파일을 컴파일
* **JIT 언어(Just-In-Time Language)**
* 소스 코드(고수준 언어 코드) → 컴파일러 → 중간 언어 코드(Intermediate Language Code)  
  → (JIT 컴파일러 → 기계어) → 실행
* 오프라인 컴파일러가 기계어가 아닌 ‘중간 언어(IL)’로 컴파일하여 실행 파일을 생성
* 플랫폼 별로 가상머신을 설치해야 함
* 가상머신(JIT 컴파일러)가 중간언어(IL)–실행파일을 기계어로 바꾸어 실행함  
  - 즉, 실행 파일은 중간 언어(IL) 상태로 존재하게 됨
* 실행 파일은 곧바로 하드웨어에서 동작하는 것이 아니라  
  - 프로그램 사용 시, 중간 단계인 가상머신을 통해 기계어로 컴파일 된 후 실행됨  
  - Just-In-time: 사용될 때 컴파일 되어 실행된다는 의미
* JIT 언어들은 플랫폼(하드웨어)에 종속적이지 않고 VM에 종속적임
* **인터프리터 언어(Interpreter Language)**
* 소스 코드(고수준 언어 코드) → (인터프리터 → 기계어) → 실행
* 소스 코드 그대로를 실행할 수 있음
* 중간 단계가 없음  
  - 컴파일이나 가상머신 등의 단계를 거치지 않고 실행됨
* 인터프리터가 소스 코드를 기계어(혹은 기계어로 된 함수, 명령어)로 대응시켜 프로그램 실행
* 실행 중에 고수준 언어를 네이티브 코드로 바꿔서 실행함
* 실제로 실행하기 전까지 코드의 문제를 잡아낼 수 없음  
  - 오프라인 컴파일러가 없어서 불가능

**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