**객체지향 프로그래밍이란:**

하나의 역할을 수행하는 함수와 변수의 묶음인 객체를 기본단위로 나눠 이들간의 상호작용으로 프로그래밍 하는 방식. 객체지향의 요소로는 캡슐화, 상속, 디형이 있습니다. C++

캡슐화 : 변수와 함수를 하나의 단위로 묶는것입니다. 캡슐화로 파생되어 정보은닉을 가능하게 합니다.

**상속관계와 다형성:**

상속은 자신의 멤버 함수와 멤버 변수를 자식 클래스에게 물려 주는 것입니다. 자식은 부모클래스의 모든 멤버를 물려 받습니다.

다형성은 객체들의 타입이 다르면 같은 함수를 호출해도 다르게 동작하는 성질. C++의 예로는 가상 함수, 함수 템플릿, 함수 오버로드 연산자 오버로드가 있습니다.

가상 함수: 부모 클래스와 동일한 이름의 함수를 자식클래스에서 다르게 작동하게 할 수 있는 함수입니다.

함수 템플릿: 같은 함수 이름으로 다양한 자료형을 처리 할 수 있는 형태입니다.

함수 오버로드: 함수를 인자의 개수나 자료형을 다르게 해서 중복 정의 할 수 있도록 하는 것입니다.

연산자 오버로드: 같은 연산자를 자료형에 따라 다르게 작동할 수 있게 하는 것.

**생성자란:**

객체가 생성될 때 호출돼서 멤버를 초기화 할 수 있는 함수, 반환형 없이 클래스 이름과 동일한 이름으로 구현합니다. 따로 구현하지 않으면 디폴트 생성자가 생성됩니다. (디폴트 생성자는 생성자가 하나라도 있으면 자동으로 생기지 않는다)

**소멸자:**

객체가 제거될 때 호출되는 함수입니다. 메모리 반환할때 사용합니다.

**virtual사용 하는 이유:**

파생 클래스에서 재정의 할 것으로 기대되는 가상함수에 virtual을 사용합니다. 소멸자에 virtaul을 사용하지 않으면 부모클래스로 자식 포인터를 호출한 경우는 자식클래스의 소멸자는 호출되지 않아서 leak이 남을 가능성이 생깁니다.  
**malloc vs new:** 둘다 동적할당을 위해 사용합니다. 차이는 malloc은 함수고 new는 연산자입니다.

함수는 함수 코드가 있는 주소로 이동하게 하지만 연산자는 컴파일러가 정해놓은 명령으로 컴파일 합니다.

**Struct와 class차이:** struct는 기본 한정자가 public, class는 기본 한정자가 private

빌드과정:

전처리 -> 컴파일러 -> 어셈블러 -> 링커

전처리: 코드 주석을 지우고 define했던것들을 치환.

컴파일러: 저수준 언어인 어셈블리어로 바꿔서 어셈블리 파일로 변환시키는 단계

어셈블러: 오브젝트 파일로 변환하는 단계(오브젝트 파일은 0과1로이뤄진 코드)

링커: 오브젝트 파일들을 묶어서 실행 코드 파일로 변환하는 단계,

함수 호출 스택:

오른값참조, 콜바이밸류

**자료구조**

선형자료구조: 배열(vector(동적 배열)), 연결 리스트(list), 스택, 큐

비선형자료구조: 트리, 그래프

벡터와 리스트의 차이:

벡터는 인덱스를 통해 임의접근이 가능합니다, 반면에 리스트는 임의접근이 불가능합니다. 저장방식으로도 벡터는 인접한 메모리에 연속으로 저장되지만 리스트는 비연속적인 메모리에 저장됩니다. 벡터는 삽입 삭제가 느립니다.

동적 배열의 할당 정책: 메모리의 여유분을 두고 실제로 사용할 메모리 보다 1.5배정로 예약을 한다.

(List의 insert는 넣어주는 iterator바로앞에 데이터 추가.)

스택: 후입선출(LIFO)

큐: 선입선출(FIFO) (front << [1][2][3][4] << rear(back))

데크: 앞과 뒤 모두에서 데이터 출력 가능

그래프: 정점과 간선으로 이뤄져서 현실 세계의 사물이나 추상적인 개념간의 연결 관계를 표현하는 자료구조

가중치그래프: 간선에 가중치를 둔 그래프