15강

배열의 이름은 변수로 사용 x ⇒ LHS x

문자열 연산 x ⇒ 함수로 우회 ⇒ 이미 구현됨

ex) strcpy , strcat , strlen …

문자열X배열 → String 사용 1) 배열처럼 인덱스 접근 o
2) 그냥 변수처럼 이용한 초기화 o
3) 관계 연산자 사용 o

변수로 선언 o ⇒ LHS o

연산도 o

- 절차 지향 프로그래밍
- 객체 지향 프로그래밍

⇒ Class : 상태 (멤버변수) + 동작 (멤버함수)

객체로 인스턴스 · 레퍼런스 선언이 가능

+) Struct (Class 와 유사)

차이점은 ? Class : 접근제어자 default private
Struct : 접근제어자 default public

16강

private : 클래스 내부에서만

public : 클래스 외부에서도

17강

생성자 : 반드시 public · 중복정의 가능 · 반환값X

default 생성자 - 파라미터 x
↓ 파라미터가 모두 default 값인
생성자가 아무 것도 없다면 컴파일러가 만들어 줌
하나라도 있으면 x

소멸자 : 반드시 public · 매개변수 x → 중복 정의 x
아무것도 반환값X

정의 x ⇒ 컴파일러가 생성 + 호출

동적 할당을 받은 경우 , 메모리 누수 방지를 위해
동적할당 받은 메모리를 반납하는 소멸자 필요

초기화리스트 : 생성자에서 멤버 변수를 간단히 초기화

① 상수 멤버

② Reference

복사 생성자 : 자신과 같은 타입의 객체를 매개변수로
안 만들면 컴파일러가 + 모든 멤버 변수 복사

Operator overloading

1) 멤버 함수

operator <<, operator >>

다른 클래스의 객체 인자로 받음
스트림 객체 멤버 반환

2) 멤버 함수

operator =

반드시 함수를 호출한 객체의
레퍼런스를 반환

TV ⇒ return *this

깊은 복사인 경우 직접 구현
그게 아니라면 구현 X (컴파일러가 함)

그러나 상수 멤버가 포함하는 경우
컴파일러는 default 로 만들어 주지 X

+) prefix / postfix

↳ 레퍼런스 반환 X
임시 변수를 반환함

+) operator []

LHS 로 사용 가능하게 레퍼런스 타입 리턴

+) 임시 obj 는 모두 const

ex : z = x+y 에서 z.operator(x+y)
에러남 ↳ 이 연산 후 여기서 임시 obj 가 생기는데 임시 obj 는 다 const 임.

∴ operator = 의 파라미터를 const 처리 해줘야 함

Type casting

1) 변환 연산자

operator 데이터타입() 로 함

파라미터, 리턴 타입 X

ex : int n = b1;
b1은 Book type 의 객체
이를 int 로 형변환

2) 변환 생성자

ex : Book b1 = 123;
integer ⇒ Book type 으로 형변환

버그의 원인이 될 수 있으므로 변환 연산자 / 생성자 앞에 explicit 선언하면 이를 막을 수 O

+) 업캐스팅

포인터 / 레퍼런스 자동적으로 일어남

+) 다운캐스팅 ⇒ 명시 해줘야 함

포인터 (변환 생성자 필요 X)
Shape * → Rectangle *

레퍼런스 (변환 생성자 필요 O)
Shape → Rectangle

* 함수의 매개 변수로 객체를 받을 때는
서브 클래스보다 수퍼 클래스가 유리함
↳ 자동 업캐스팅!

가상 함수

실제 객체의 종류에 맞는 함수 호출

포인터 쓸 때 유의할 것

ex: print() 함수가 가상 함수

kvector *k[2] = {a, b}; 에서

k[1] → print() 하면 a와 b의 print 함수 호출

그러나 cout << *(k[1]) 하면 kvector 의 operator<< 호출

소멸자의 virtual 선언

: 객체의 포인터가 부모 클래스의 포인터인 경우 서브 클래스의 소멸자 호출 X

동적할당까지 받았다면 메모리 누수가 발생

⇒ 소멸자는 virtual 선언 해야 함

순수 가상 함수

virtual 함수명() = 0; 헤더만 작성, 구현 X

순수 가상 함수를 하나라도 가진 클래스가

추상 클래스 (객체 생성 X) → 인터페이스로 활용 가능함

↳ 추상 클래스를 상속받는

서브 클래스는 순수 가상 함수를 필수적으로 구현.

예외처리 throw ↔ catch

throw 에서 <클래스 타입의 객체> 던질 수 O

↳ 클래스 이름만 쓰면 안됨, 객체 O

예외 클래스가 상속 관계인 경우,

catch 문에 서브를 부모보다 앞에 적어야 함

Template

template <typename T>
template <class T>
선언이 필요

1) 템플릿 함수

2) 특수화된 템플릿 함수 ⇒ 우선순위 2 → 1

3) 그냥 일반 함수

클래스 템플릿도 가능함

객체 생성시 타입 명시 해주어야 함

클래스 외부에 함수 구현할 때

template <class T>

Box<T> :: Box(){}; 이렇게 구현해야함.

템플릿 클래스 사용 시 다른 파일에

선언부와 구현부 분리 X ⇒ 하나에 선언 + 구현

STL

1) 순차 컨테이너 사용자가 넣는대로

- vector

뒤에서만 원모 삽입 / 삭제 가능

- deque

앞에서도 가능

- list

무작위 접근 안 되는 deque

bidirectional iterator 라서

+, -, <, >, [] 등의 연산 x

2) 연관 컨테이너

key에 의한 접근

전부 bidirectional iterator

[] 등 특별 인덱스 접근 x

원모들이 다 정렬되어 있음

- set

중복 x

순서 x

- multiset

중복 가능한 set

- map

<key, value>의 pair로 구성

[] 연산이 인덱스 x

키 접근 O

⇒ value의 reference 반환

- multimap

중복 가능한 map

key 값이 unique x

[] 아예 x

3) 이터레이터

포인터 같은 것

공통적으로 *, ++, ==, != 사용 가능

대표적으로 begin()
end()
함수 등이 포함

⇒ iterator를 리턴

ex: vector<int>::iterator it = v1.begin()

↳ auto로 대체 가능,
초기화가 되어 있어야 함

그리고 당연히 v1과 vector<int>
타입이 일치해야 함

4) 알고리즘

변경 / 불변경 포함

- 글로벌 lower, upper, equal 어쩌구 …

정렬 x 순차 컨테이너들은 위 함수들이
멤버함수로 존재

정렬 후 글로벌 함수 사용 ⇒ 구간 지정이 가능

- 멤버 lower, upper, equal 어쩌구 …

연관 컨테이너는 위와 같은 함수가
글로벌 + 멤버 둘 다 포함됨

둘 다 사용이 가능함 ⇒ 멤버함수는 전체 구간