과제

Operator

- Conversions
  - `const_cast`: 서로 다른 cv-qualifier(type의 속성을 수정하는 `const`(선언 이후 read-only)와 `volatile`(변수 값이 프로그램이 통제할 수 없을 수도 있음.)을 통칭하는 용어)를 가진 타입 간의 변환
    - 문법: `const_cast<target-type>(expression)`
  - `static_cast`: 하나의 타입을 다른 관련된 타입으로 변환
    - 문법: `static_cast<target-type>(expression)`
  - `dynamic_cast`: 상속 계층 구조에 따라 클래스 포인터랑 참조를 안전하게 변환
    - 문법: `dynamic_cast<target-type>(expression)`
  - `reinterpret_cast`: bit 패턴을 재해석해서 타입을 변환.
    - 문법: `reinterpret_cast<target-type>(expression)`
- 메모리 할당
  - new: 메모리 동적 할당
    - 문법: `new (type) new-initializer`
  - delete: 메모리 동적 비할당
    - 문법: `delete([]) expression`
- 기타
  - `constant`: 컴파일 타임에 처리되고 컴파일 타임 상황에서 사용됨
    - `literal type`: scalar, reference, literal 배열 등
  - `sizeof`: object나 타입의 크기를 알아낼 때 사용.
    - 문법: `sizeof(type)`, `sizeof expression`
  - `alignof`: typeid로 지정된 타입의 인스턴스에 필요한 바이트 단위의 alignment(`size_t` 타입의 음이 아닌 정수 값. 연속적인 주소 사이의 바이트 수)를 반환
    - 문법: `alignof(type-id)`

- `typeid`: 타입의 동적 유형을 알아야하거나 정적 타입 식별이 필요할 때 사용.static storage duration을 가짐.
  - 문법: `typeid(type)`, `typeid(expression)`
- `throw-expression`:

Classes

Object: instantiation of a class

ex) class - type, object - variable

정의: class, struct 키워드로 정의됨

```
class className{
    access_specifier: member1;
} object_names; //얘는 옵션
```

- 멤버(데이터, 선언된 함수, nested type, enumerator, member template 등)로 구성
  - nonstatic
  - static: 전체 프로그램에서 오직 하나의 instance(data member의 경우)이며 static storage duration을 가짐(예외 - `thread_local` 키워드가 사용될 경우 스레드별로 하나의 객체가 thread storage duration을 가지게됨)
    - `inline static`: `inline`으로 선언 가능. inline static data member는 클래스 안에서 정의될 수 있고 initializer 지정 가능
    - `const static`: constant static member. 클래스 정의 내에서 모든 표현식이 constant expession인 initializer로 초기화 가능
- access_specifier: 접근 제한자. `private`, `public`, `protected` 셋 중 하나
  - private: 같은 클래스의 다른 멤버들만 접근가능
  - protected: 같은 클래스의 다른 멤버들 + derived(자식) 클래스 멤버들 접근가능
  - public: object가 보이는 어디서나 접근 가능
- 클래스 속성
  - Trivially copyable class

        - copy/move constructor, copy/move assignment operator가 하나 이상 존재
        - copy/move constructor, copy/move assignment operator가 단순
    - Trivial class
        - 단순 복사 가능한 클래스
        - 하나 이상의 단순한 디폴트 생성자
    - Standard-layout class
    - Implicit-lifetime class
    - POD class

Coroutine

coroutine: 실행을 일시 중지하고 나중에 다시 시작할 수 있는 함수

- **stackless**: 실행 일시 중지 → 호출자에게 반환 & 실행 재개에 필요한 데이터는 스택과 별도로 저장
- 비동기적으로 실행되는 코드에서 순차적인 실행이 가능하게 해줌
- 다음이 포함되어 있으면 coroutine 함수라고 할 수 있음
    - co_await: 실행 중단 후 나중에 재개될 때까지 기다림

```
task<> tcp_echo_server()
{
   char data[1024];
   while (true)
   {
            std::size_t n=co_await socket.async_read_some(buffer(da
ta));
      co_await async_write(socket, buffer(data, n));
   }
}
```

    - co_yield: 값을 반환하는 실행 중단

```
generator<unsigned int> iota(unsigned int n = 0)
{
```

```
    while (true)
        co_yield n++;
}
```

- co_return: 값을 반환하는 실행을 완료

```
lazy<int> f()
{
    co_return 7;
}
```

- 제약: coroutine은 variadic arguments(...으로 파라미터를 넘겨주는 경우), 일반적인 return문, auto나 Concept 같은 placeholder return 타입을 사용할 수 없음. 생성자, destructor, main함수는 coroutine이 될 수 없음
- 실행
  - promise object: 코루틴 내부에서 조작. 코루틴이 결과나 예외를 전달하는 객체.
  - coroutine handle: 코루틴 외부에서 조작. 코루틴의 실행 재개, 코루틴 프레임 삭제하는 데 사용됨
  - coroutine state: 동적으로 할당된 저장소 객체. promise object, 매개변수(copied by value), 현재 일시 중지 지점의 일부 representation(재개 시 어디서부터 실행할지 알 수 있도록) 등이 포함됨

Promise

```
//기본
template<class R> class promise;
//스레드 간 객체 전달 시 사용
template<class R> class promise<R&>;
//상태가 없는 이벤트 전달 시 사용
template<> class promise<void>;
```

클래스 템플릿 `std::promise`는 나중에 `std::promise` 객체에 의해 생성된 `std::future` 객체를 통해 비동기적으로 획득할 수 있는 값 또는 예외를 저장하는 기능을 제공합니다. `std::promise` 객체는 한 번만 사용하도록 설계되었습니다.

- 나중에 promise 객체의 의해 생성된 future 객체를 통해 비동기적으로 얻을 수 있는 값/예외를 저장하는 기능
- 한 번만 사용할 수 있는 객체로 설계됨
- 각 promise는 shared state(state 정보와 아직 연산되지 않은 결과 등의 정보 저장)와 관련되어 있음. 이와 관련해 promise는 3가지 작업을 할 수 있음
    - make ready: shared state에 결과/예외 저장. 상태를 준비 완료로 설정하고 future에서 대기 중인 모든 스레드에 대해 unblock
    - release: shared state에 대한 참조를 포기함. 마지막 참조였을 경우 shared state가 파괴됨
    - abandon: future_error 타입의 예외를 저장한 후 shared state을 ready 상태로 만든 후 release함