오류 처리

2021년 1월 4일 월요일 오후 6:46

C 에서의 오류처리

- C 언어는 예외(exception)을 지원하지 않음
- 그러면 어떻게 실행 중에 생기는 문제들에 대처하나요?

예외 처리가 언제나 좋은 것은 아님

- 사실 예외 처리가 반드시 소프트웨어에 품질을 높이진 않음
- 그냥 예외 처리라는 게 인간이 완벽히 하기에는 거의 불가능한 존재
- 그리고 예외 처리 기능이 존재하는 언어는 오히려 프로그래머를 게으르게 만들기도 함

일반적인 사람들의 사고방식

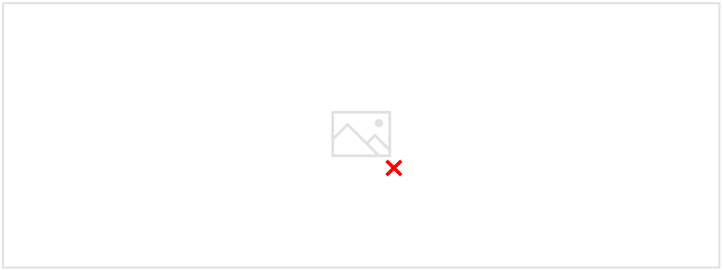
- 여러 단계의 일을 설계할 때 최상의 경우(happy path)만 고려
- 내가 작성하는 코드 한 줄 한 줄이 잘못될 수 있다는 생각을 안 하고 기능을 쭉쭉 작성해 나감
- 실행 중에 예외가 발생하면 제대로 고치지 않고 일단 패스 하는 경우도 많음
  - 그런다고 프로그램이 크래시 나지 않으니 일단 모른 척
- 그래서 실제로는 버그가 무수하나 어떻게든 작동하는 프로그램이 나오기도 함

크래시가 나면 다른 방법이 없음

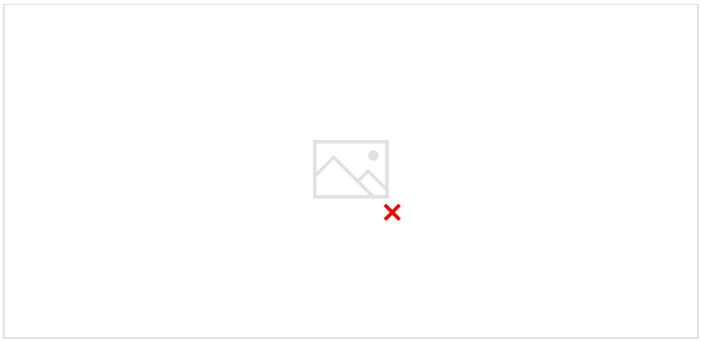
- 예외 처리가 없는 언어에서 문제가 발생하면?
  - 제대로 대처 안하면 크래시가 남
  - 즉, 프로그램이 뻗고 고객이 항의
  - 고객의 항의는 개발자를 일하게 만드는 원동력
- 덕분에 오히려 예외 상황이 발생하면 빠르게 대처함
  - 프로그램이 돌다 뻗어서 아무것도 못하니 제대로 고칠 수밖에
  - 참고로 말하면 지금 사용중인 운영체제도 다 C 기반

안 좋은 오류 처리의 예

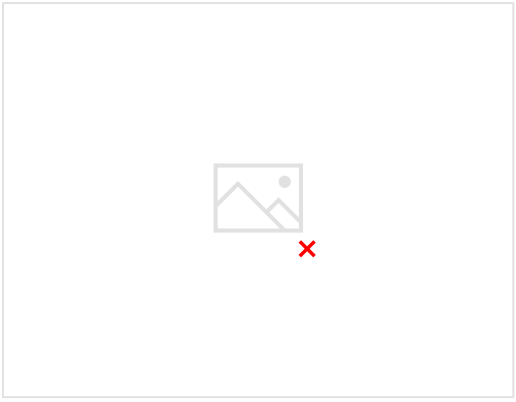
- 이런 코딩 스타일은 안 좋음
  - 일단 아무 생각 없이 무조건 동작한다고 생각하고 코드 작성



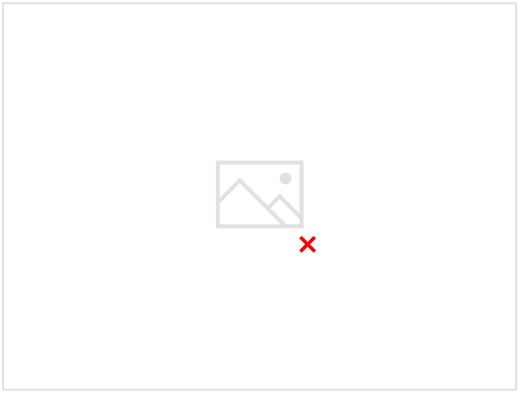
  - 그 뒤 특별한 원칙 없이 버그 나올 때마다 땜빵으로 고치면 어느 순간 코드가 이해하기 어려워짐
    - 예: 매개변수가 널 포인터인지 비교



        - 이렇게 모든 함수마다 NULL이 들어오지 못하게 체크를 해야할까?
            - 철통 방어니까 좋은거 아닐까?
- 현실에서의 비유: 두꺼비집
    - 과부하 걸리면 두꺼비집의 퓨즈가 끊어지거나 차단기가 내려감
    - 두꺼비집은 보통 집에 한 곳에 위치해 있다



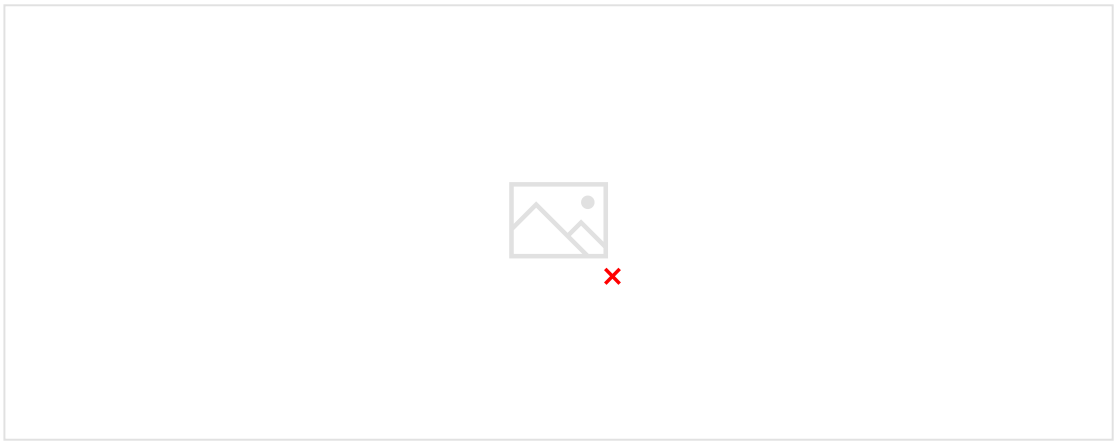
    - 근데 어디서 과부하가 걸릴 줄 모르니 전선 5cm마다 차단기를 달면?



        - 나중에 어느 차단기를 확인하고 올려야 하는지 조차 모름
        - 너무 많으니 차단기 확인도 대충 함
    - **문제는 한 군데에서만 찾는게 더 효율적**
        - 집중의 문제
        - 문제가 발생하면 -> 차단기 하나 확인 -> 올리고 확인
    - 사람은 생각보다 집중을 못한다
        - 책에서 중요한 내용은 마지막에 5줄로 요약했으니 이거는 꼼꼼히 읽으세요 하면 그곳을 꼼꼼히 읽음
        - 책 전체가 다 중요하니까 전체를 꼼꼼히 읽으라 하면 전체를 꼼꼼히 읽는 사람은 없다
- 다시 프로그래밍으로 돌아와서
    - 따라서 오류 처리를 할 때도 원칙이 있어야 함
        - 다른 언어에서 예외 처리할 때도 마찬가지
    - 생각 없이 무조건 작동한다고 코드 짜는 건 일단 OK
        - 그건 인간의 한계
    - 그러나 그 문제를 찾는 곳은 최소한인 게 좋음

올바른 오류 처리 전략

- 버그와 오류의 차이
  - 버그
    - 일어날 수 없다고 가정한 상황
    - 즉, 선조건(precondition) 및 후조건(postcondition)이 성립하지 않고 어서트에 실패하는 경우
    - 프로그램이 이런 상황에서도 올바르게 작동하게 대처해두는 건 말이 안 됨
    - 그냥 그 버그가 다시는 일어나지 않게 코드를 수정해서 재 컴파일
  - 오류
    - 실제 실행 중에 일어날 수 있는 예측 가능한 예외적인 상황들
    - 당연히 프로그램이 대처해야 함
    - 예: 사용자가 자유롭게 로딩할 수 있는 파일이 없음
  - 선조건과 후조건
    - 함수 이름이나 변수 이름으로 유추할 수 있어야 함
    - 불가능하다면 주석으로 설명
    - 두 조건이 참인지를 검사하는 어서트를 충분히 넣을 것
- 어서트 문제는 실행해야만 보인다는 것
  - C89에서는 컴파일 중에 판단 가능한 것도 모두 실행해야만 보임
    - 예: 구조체의 크기
  - C11은 정적 어서트(static assert)로 이러한 한계를 극복
    - 컴파일 도중에 어서트 조건을 판단해서 컴파일 오류를 던져줌
      - 구조체 크기가 몇이냐는 컴파일 도중에 결정 되어서 정적 어서트가 가능
- 어서트 예



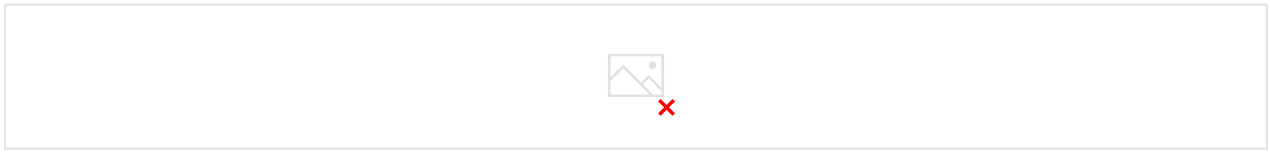
  - 선조건: 예금 들어오는게 0이 되면 안 됨
  - 후조건: 예금을 넣었으면 before_total보다 after_total이 더 커야 함
  - 버그를 다 고친 것일까?
    - 아님. 테스트로 발견한 버그를 다 고친 것임
    - 발견하지 못한 버그가 있을 수 있음
    - 고로, 버그가 보이지 않는 것일 뿐 버그는 여전히 존재할 수 있음
- 실행 중 오류 처리는 어떻게 하나요?
  - 일단 버그는 잡았다 "가정"
  - 가정을 했다는 건?
    - 내 함수에 들어오는 데이터는 다 유효(valid)하다는 것
      - 데이터로 널이 들어오냐 체크하면 끝이 없다

매개변수가 100개 넘게 들어온다고 하면 100개 다 널 체크를 할 수 없다

- 유효하지 않은 데이터가 들어오면 어디선가 걸러줘야 함
- 그 어딘가를 '경계'(boundary)라고 부를 것임
  - 경계란, 통제할수 있는 환경과 통제할수 없는 환경의 사이
  - 경계가 있는 곳에서 데이터가 유효한지 확인하면 된다
    경계가 아닌 코드부분은 유효한 데이터를 가진다고 생각하자
    - 그래서 일일이 함수마다 널 체크 안해도 됨
- 경계의 예
  - 내 프로그램 <-> 파일 시스템
  - 내 프로그램 <-> 키보드 입력
  - 내 프로그램 <-> 외부 라이브러리 (3rd-party library)

널 포인터를 허용한다면 함수나 변수에 명시하자

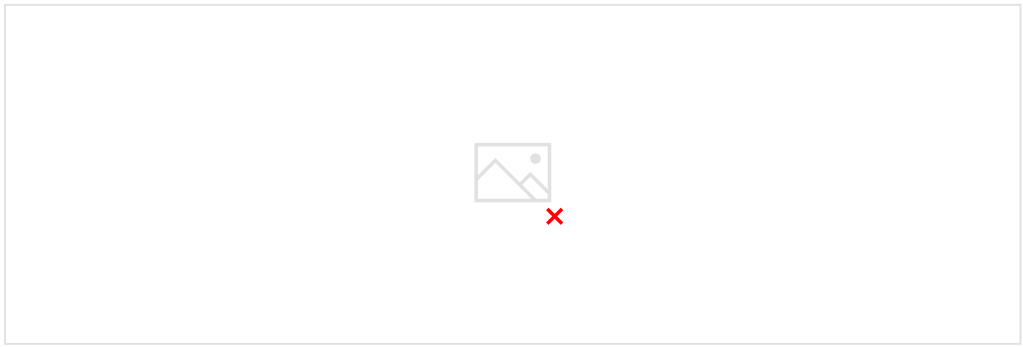
- 코딩 표준: 함수의 매개변수가 널 포인터를 허용한다면, 매개변수 이름 끝에 '_or_null'을 붙인다
  - 이렇게 명시해줘야 함수 호출자가 널을 받을 수 있다는걸 인지하고 대비함
    매개변수도 마찬가지로 널을 매개변수로 넣어도 될까 알 수 있음
  - '_or_null'을 붙이지 않는다면 널 포인터는 받지 않는다고 가정
- 함수도 마찬가지



- 주석으로 하면 사람들이 잘 안 봄
  - 함수 시그니처에 최대한 정보를 담자
  - 다만, 아까전에 deposit함수에서 0 을 입금하지 못한다는걸 명시하려면
    어쩔 수 없이 주석을 써야 함

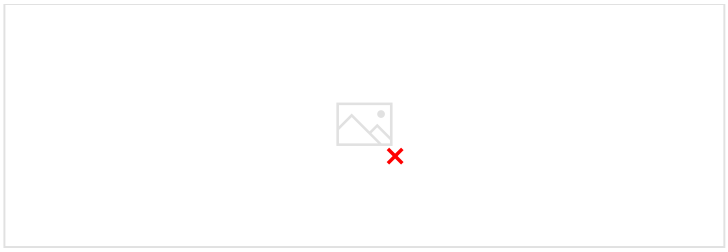
예외처리 대신 오류 코드를 반환하자!

- 오류를 처리해주는 함수/코드에서 오류가 있음을 알려줘야 함
- 가장 좋은 방법은 함수에서 곧바로 오류 코드를 반환하는 것

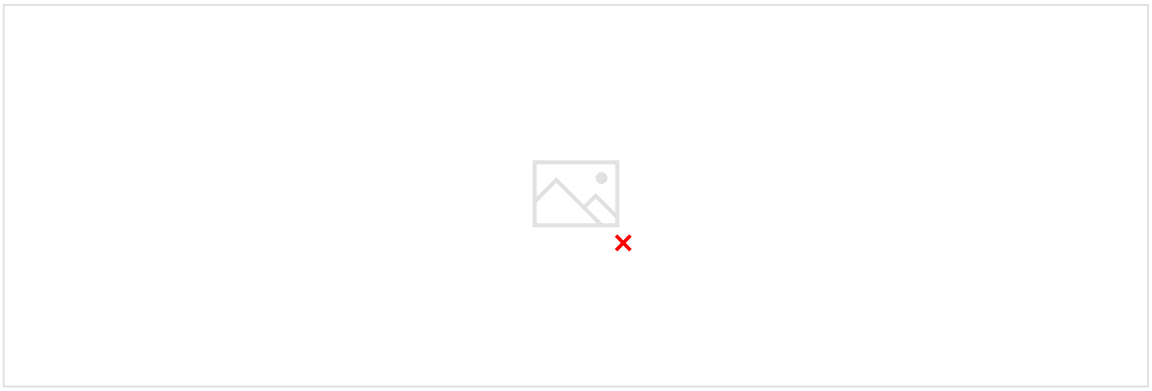


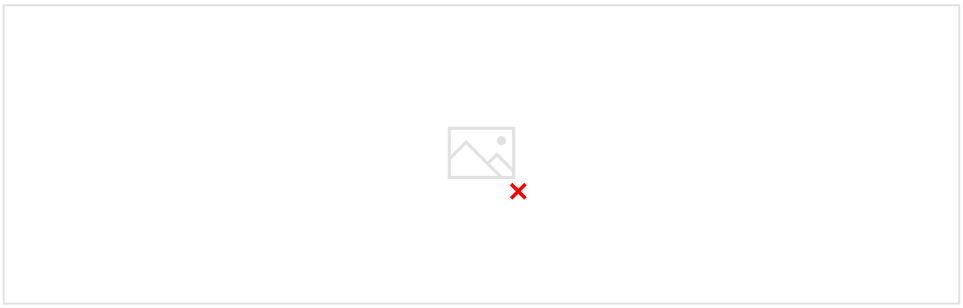
- try_get_student는 경계 처리 함수임. 외부에서 데이터를 받고 오류 코드만 반환을 함
  - 호출자가 알아서 오류를 처리 해줘야 한다
- 다른 언어에서처럼 예외를 반환하면 좋긴하지만 사람들이 이에 대해 문서를 읽지 않는다
- 차라리 libabc_error_t 같은 오류 코드를 함수에서 반환해버리면 사람들이 저게 뭔지 궁금해서 찾아볼 것

모든 오류 코드를 하나의 enum으로 만들자



- 구조체로 반환도 가능하나 C에서 많이 쓰는 방법은 아님
  - 아마 용량 때문에?
  - 그래도 좋은 방법이니 쓰자
- 오류 코드 만들 때는 해당 라이브러리에서 제공할 수 있는 모든 오류 코드를 하나의 enum으로 정의하는 게 좋다
- 함수마다의 오류 enum을 만드는 건 좋지 않음
  - enum_a와 enum_b가 있으면 그 둘의 오류 코드를 비교하는 실수를 저질를 수 있다
    - C에서는 다른 enum끼리 상호 비교가 가능
    - enum끼리 서로 대입이 돼서 실수도 가능





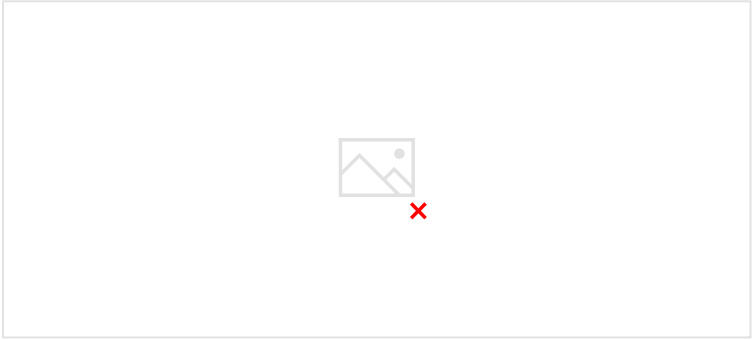
- enum의 타입을 보장하는 C#같은 언어에서는 각 함수마다 enum을 분리해도 좋다

- 전에 본 errno도 좀 별로
  - 전에 봤던 어떤 함수가 내부적으로 errno에 저장하는 방법도 있음
  - 이건 아주 훌륭한 방법은 아님
  - 함수가 errno을 세팅하는지 코드나 문서를 읽어보지 않는 한 모름
    - 이 함수가 만약 외부 라이브러리라면 더더욱 알기 힘듦
- 올바른 오류 처리 전략 정리
  1. 기본적으로 내가 작성하는 모든 함수에 들어오는 데이터는 유효하다 가정하고 어서트를 많이 쓸 것
  2. 그렇지 않은 함수는 매개변수나 함수 이름에서 그렇지 않다는 사실을 명백히 표시할 것
  3. 오류 상황을 처리하는 장소는 최소한으로 할 것
  4. 어떤 함수가 오류 처리를 한다는 사실을 반환형 등을 통해 확실히 보여줄 것

■ 오류 처리 후에도 발생하는 예외 상황

- 근데 그래도 프로그램이 오작동하면?

    - 버그임
    - 코드를 다시 고쳐야 함
        - 소프트웨어의 품질은 테스트/QA 프로세스의 문제
    - 어떤 경우에는 크래시도 남
        - 널 포인터 역 참조 등
- 이런 상황을 캐치해서 또 대비하고 싶다면?
    - C 표준에는 방법이 없음
        - try / catch 문 같은거 없음
    - 운영체제에서 이런 문제를 찾아서 SEH나 시그널을 주기는 함



- 운영체제의 예외 처리
    - 함수 포인터를 등록하고 OS가 보내는 예외 처리를 받아올 수는 있음
        - 콜백 함수
    - 근데 받아와도 어떻게 대처해야 할지 애매한 경우가 있음
        - 뒤에 배울 동적 메모리 할당에서 더 이상 메모리 못받아오면 NULL을 반환
            - 이걸 어떻게 해결하지?
        - 반드시 있어야 하는 파일 열다가 실패하는 경우는 또 어찌 해결하지?
    - 외부에서 생기는 문제는 해결하기 어렵거나 불가능한 경우가 많다
        - 이 경우 할 수 있는 방법은 크래쉬가 난 프로그램 실행 로그들을 모아서
          나중에 재현해내서 어떤 문제가 있나 보고 코드를 수정하는 것 밖에 없는듯 하다.
        - 이런 상황에서 예외처리하고 계속 실행하는 것도 불가능 할 듯
          차라리 크래쉬를 내고 프로그램을 다시 시작하는게 나음