전북대 소중해유(You)

디버깅 실습:
실제 에러 사례를 통한 디버깅 실습

전북대학교 JEONBUK NATIONAL UNIVERSITY
전북대학교 SOFTWARE중심대학사업단
SW중심대학
과학기술정보통신부
IITP 정보통신기획평가원

목차

디버깅(debugging)?

- 프로그램에서 발생하는 **오류를 찾아 수정하는 과정**이며, 코드의 정확성과 안정성을 확보하기 위한 필수적인 단계
- 소프트웨어가 예상대로 작동하지 않으면 컴퓨터 프로그래머는 오류가 일어나는 원인을 알아내기 위해 코드를 분석해야 하며, 이들은 디버깅 도구를 사용해 소프트웨어를 제어된 환경에서 실행하고 코드를 단계별로 확인하여 문제를 분석하고 수정하도록 함.
- 컴퓨터 프로그래머들은 1950년대에 버그와 디버깅이라는 용어를 처음 사용한 것으로 기록되었고, 1960년대 초에 프로그래밍 커뮤니티에서 디버깅이라는 용어를 일반적으로 사용한 것에서 유래

디버깅(debugging)이 중요한 이유?

- 컴퓨터 프로그래밍은 추상적이고 개념적인 활동이므로 버그와 오류가 발생하기 마련임.
- 컴퓨터는 전자 신호 형태로 데이터를 처리하며, 프로그래밍 언어는 이를 사람이 이해할 수 있는 방식으로 추상화함.
- 모든 소프트웨어에는 여러 가지 추상화 계층이 존재하며, 구성 요소가 서로 통신하여 애플리케이션이 정상적으로 작동하도록 설계됨.
- 오류가 발생하면 문제를 찾아 해결하는 것이 어려울 수 있음.
- 디버깅 도구와 전략은 문제를 신속하게 해결하고 개발자의 생산성을 높이는 데 기여함.
- 결과적으로 소프트웨어 품질과 최종 사용자 경험이 개선됨.

디버깅(debugging)이란?

디버깅 프로세스

- **오류 식별**: 개발자, 테스터 및 최종 사용자가 소프트웨어를 테스트하거나 사용하는 동안 발견된 버그를 보고하는 단계임. 개발자가 버그의 원인이 된 정확한 코드 줄 또는 모듈을 찾는 과정이 포함됨.
- **오류 분석**: 프로그램 상태 변경 및 데이터 값을 기록하여 오류를 분석하는 단계임. 소프트웨어 기능에 미치는 영향을 기준으로 버그 수정의 우선순위를 정하며, 팀이 개발 목표와 요구 사항에 따라 버그 수정 일정을 수립함.
- **수정 및 검증**: 개발자가 버그를 수정한 후 소프트웨어가 정상적으로 작동하는지 확인하는 단계임. 추가적으로, 미래에 동일한 버그가 재발하지 않도록 새로운 테스트 케이스를 작성하는 과정이 포함됨.

디버깅(debugging)이란?

디버깅 vs 테스트

- 디버깅과 테스트는 소프트웨어 프로그램이 제대로 실행되도록 보장하는 보완적인 프로세스임.
- 프로그래머는 코드의 일부 또는 전체를 작성한 후, 버그와 오류를 식별하기 위해 테스트를 수행함.
- 버그가 발견되면 개발자는 디버깅 프로세스를 시작하여 소프트웨어에서 오류를 제거함.

디버깅이 필요한 코딩 오류

- **구문 오류**: 컴퓨터 프로그램에 잘못된 명령문이 있을 때 발생하는 오류임. 워드 프로세서에서의 오타나 철자 오류와 유사함. 구문 오류가 있으면 프로그램이 컴파일되거나 실행되지 않으며, 코드 편집 소프트웨어는 이러한 오류를 강조 표시함.
- **의미론적 오류**: 프로그래밍 명령문을 잘못 사용했을 때 발생하는 오류임. 예를 들어, x/(2π)라는 표현식을 Python에서 다음과 같이 잘못 작성할 수 있음.

```
y = x / 2 * math.pi
```

- 이 경우 Python에서 곱셈과 나눗셈이 동일한 우선순위를 가지며 왼쪽에서 오른쪽으로 계산되므로, 이 식은 (xπ)/2로 계산되어 버그가 발생함.

디버깅이 필요한 코딩 오류

- **논리 오류**: 프로그래머가 프로그램의 단계적 처리 과정이나 알고리즘을 잘못 작성했을 때 발생하는 오류임. 예를 들어, 루프가 너무 일찍 종료되거나 잘못된 if-then 결과가 발생할 수 있음. 여러 입력/출력 시나리오를 통해 논리 오류를 찾아낼 수 있음.
- **런타임 오류**: 소프트웨어 코드가 실행되는 컴퓨팅 환경에서 발생하는 오류임. 예로는 메모리 부족이나 스택 오버플로가 있음. 이 오류는 try-catch 블록으로 예외를 처리하거나 적절한 메시지를 로깅하여 해결할 수 있음.

디버깅 도구 및 기법

- **브라우저 개발자 도구**: 네트워크 트래픽 분석, DOM 검사, 콘솔 로그 확인 등 다양한 기능을 제공
- **IDE 디버거**: 코드 중단점 설정, 변수 값 확인, 호출 스택 분석 등의 기능을 통해 효과적으로 디버깅할 수 있는 도구
- **로그 사용**: console.log, console.error 등을 활용해 코드 실행 중 발생하는 상황을 확인할 수 있음
- **타임 트래블 디버깅**: 특정 시점으로 돌아가 코드의 상태를 확인하는 디버깅 방법 등이 존재

Node.js에서 사용 가능한 디버깅 도구 및 기법

- **Node.js 기본 디버거 (node inspect)**: Node.js에는 기본적으로 제공되는 디버거 도구인 node inspect가 있으며, 해당 도구를 사용하면 코드 실행 중 중단점(Breakpoint)을 설정하고, 변수 값 확인, 함수 호출 스택 확인 등의 작업을 수행할 수 있음.
- **IDE 디버거**: Visual Studio Code(VS Code)는 Node.js 개발에 널리 사용되는 통합 개발 환경(IDE)으로, 강력한 디버깅 기능을 제공함. 코드 중단점 설정, 변수 및 객체의 현재 상태 확인, 디버깅 콘솔을 통한 실시간 코드 실행 등이 가능함.
- **Chrome DevTools**: Node.js 애플리케이션을 디버깅하기 위해 Chrome DevTools를 사용할 수 있음. 브라우저에서 제공하는 개발자 도구를 활용하여 서버 측 Node.js 코드의 디버깅을 지원함. (node –inspect app.js)
- **winston 및 morgan을 활용한 로깅**: winston과 morgan과 같은 로그 관리 라이브러리를 사용하여 애플리케이션의 동작을 로그로 기록하고, 오류나 중요한 이벤트 발생 시 이를 분석할 수 있음.
- **mocha와 chai를 사용한 테스트 기반 디버깅**: 테스트 코드를 작성하고, 이를 통해 코드의 동작을

실제 사례 분석 및 디버깅

- **참조 오류 (reference error)**
  - 오류 메시지를 확인하고, greeting 변수가 선언되지 않았음을 인지할 수 있음.
  - 변수 선언을 추가하여 문제를 해결함.

```
// 오류 발생 코드
function sayHello() {
    console.log(greeting);  // ReferenceError: greeting is not defined
}

sayHello();

// 수정된 코드
function sayHello() {
    const greeting = 'Hello, World!';
    console.log(greeting);  // 출력: Hello, World!
}

sayHello();

```

실제 사례 분석 및 디버깅

- **비동기 코드 문제 (Promise와 Async/Await)**
  - try-catch 블록을 추가하여 비동기 함수에서 발생하는 오류를 잡아내고 로그로 기록함.
  - await로 비동기 함수가 올바르게 호출되고 있는지 확인함.

```
// 오류 발생 코드
async function fetchData() {
    let data = await getData();  // getData 함수가 제대로 반환되지 않음
    console.log(data);
}

fetchData();

// 수정된 코드
async function fetchData() {
    try {
        let data = await getData();
        console.log(data);
    } catch (error) {
        console.error('Error fetching data:', error);
    }
}

fetchData();

```

실제 사례 분석 및 디버깅

- **타입 오류 (TypeError)**
  - 오류 메시지에서 user.getName이 함수가 아님을 확인하고, getName 메서드를 추가하여 문제를 해결함.

```
// 오류 발생 코드
let user = {
    name: 'Alice',
    age: 25
};

user.getName();  // TypeError: user.getName is not a function

// 수정된 코드
let user = {
    name: 'Alice',
    age: 25,
    getName: function() {
        return this.name;
    }
};

console.log(user.getName());  // 출력: Alice

```

실제 사례 분석 및 디버깅

- **배열과 객체 관련 오류**
  - 배열과 객체의 크기와 구조를 확인하고, 접근하려는 값이 없는 경우 기본 값을 제공하는 방법으로 오류를 방지함.

```
// 오류 발생 코드
let numbers = [1, 2, 3];
console.log(numbers[3]);  // undefined, 인덱스 초과

let person = { name: 'Bob' };
console.log(person.age);  // undefined, 키가 존재하지 않음

// 디버깅 방법: 값이 없는 경우 기본 값을 설정
console.log(numbers[3] || 'No value');  // 출력: No value
console.log(person.age || 'Age not provided');  // 출력: Age not provided

```

실제 사례 분석 및 디버깅

- **Null 또는 Undefined 오류**
  - 배열과 객체의 크기와 구조를 확인하고, 접근하려는 값이 없는 경우 기본 값을 제공하는 방법으로 오류를 방지함.

```
1  // 오류 발생 코드
2  function printLength(str) {
3      console.log(str.length);  // TypeError: Cannot read property 'length' of null
4  }
5
6  printLength(null);
7
8  // 수정된 코드
9  function printLength(str) {
10     if (str) {
11         console.log(str.length);
12     } else {
13         console.log('Invalid string');
14     }
15 }
16
17 printLength(null);  // 출력: Invalid string
18
```

실제 사례 분석 및 디버깅

- **메모리 누수**
  - 메모리 사용량을 모니터링하고, 필요 없는 데이터나 이벤트 리스너를 적절히 제거하여 메모리 누수를 방지함.

```
1  // 오류 발생 코드
2  let data = [];
3
4  function addData() {
5      data.push(new Array(1000000).fill('Memory Leak'));
6  }
7
8  setInterval(addData, 1000);
9
10 // 수정된 코드: 이벤트 리스너 제거 및 데이터 정리
11 let data = [];
12 let intervalId;
13
14 function addData() {
15     if (data.length > 10) {
16         data.shift();  // 오래된 데이터 제거
17     }
18     data.push(new Array(1000000).fill('Memory Leak'));
19 }
20
21 intervalId = setInterval(addData, 1000);
22
23 // 메모리 누수 방지를 위해 일정 시간 후 interval 해제
24 setTimeout(() => clearInterval(intervalId), 60000);  // 1분 후 해제
25
```

실제 사례 분석 및 디버깅

- **퍼포먼스 문제**
  - 퍼포먼스 프로파일러를 사용하여 실행 시간 분석 후, 코드 최적화 기법을 적용함.

```
// 오류 발생 코드
function slowFunction() {
    let total = 0;
    for (let i = 0; i < 1000000000; i++) {
        total += i;
    }
    return total;
}

console.time('slowFunction');
slowFunction();
console.timeEnd('slowFunction');

// 수정된 코드: 최적화된 코드로 대체
function optimizedFunction() {
    return (1000000000 * (1000000000 - 1)) / 2;  // 가우스 공식 사용
}

console.time('optimizedFunction');
optimizedFunction();
console.timeEnd('optimizedFunction');

```

실제 사례 분석 및 디버깅

- **네트워크 오류 및 API 호출 실패**
  - 네트워크 요청에 대한 오류 처리와 상태 코드 확인을 추가하여 더 안전하게 데이터를 가져오고, 발생한 오류를 적절히 로그에 기록함.

```
// 오류 발생 코드
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));

// 수정된 코드: 오류 처리 및 상태 코드 확인 추가
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok ' + response.statusText);
        }
        return response.json();
    })
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error.message));

```

실제 사례 분석 및 디버깅

- **데이터베이스 연결 및 쿼리 오류**
  - 데이터베이스 연결 정보가 올바르게 설정되었는지 확인하고, 쿼리 실행 시 발생할 수 있는 오류를 철저히 검토함.

```
// 오류 발생 코드
const { Client } = require('pg');
const client = new Client();

client.connect()
    .then(() => console.log('Connected to database'))
    .catch(err => console.error('Connection error', err.stack));

// 수정된 코드: 연결 정보 추가 및 오류 처리 강화
const { Client } = require('pg');
const client = new Client({
    user: 'user',
    host: 'localhost',
    database: 'mydb',
    password: 'password',
    port: 5432,
});

client.connect()
    .then(() => console.log('Connected to database'))
    .catch(err => console.error('Connection error', err.stack));

```

유용한 디버깅 모범 사례

- **작은 단위로 문제를 나눠서 해결:** 복잡한 문제를 작은 단위로 나누어, 한 단계씩 해결하는 접근법.
- **코드를 읽기 쉽게 작성:** 가독성이 높은 코드 작성은 디버깅을 쉽게 해 줌.
- **의심되는 부분부터 확인:** 문제가 발생할 가능성이 높은 부분부터 검토.
- **변경 사항 하나씩 적용:** 여러 가지 변경을 동시에 적용하지 말고, 한 가지씩 수정하면서 테스트.
- **주기적인 로그 기록:** 코드 실행 중 발생하는 중요한 이벤트나 변수 값을 주기적으로 로그에 기록하여 추적.

감사합니다.

- 본 온라인 콘텐츠는 2024년도 과학기술 정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 'SW중심대학'의 결과물이라는 출처를 밝혀야 합니다.

전북대학교 JEONBUK NATIONAL UNIVERSITY
전북대학교 SOFTWARE중심대학사업단
SW중심대학
과학기술정보통신부
IITP 정보통신기획평가원