

Compiler (컴파일러) 개론 개요

2015년 2학기
충남대학교 컴퓨터공학과
조은선

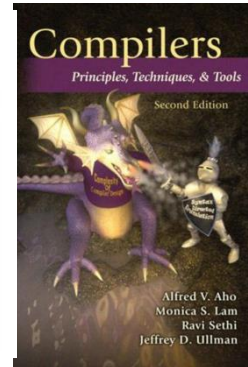
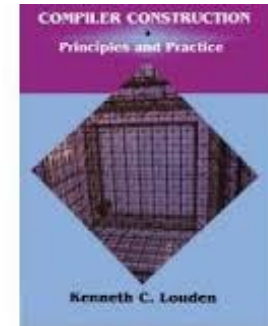
이번 시간에 할 것

- 과목 설명
 - 수업 설명
 - 컴파일러란 무엇인지
 - 간단한 컴파일러의 처리과정 Overview
 - 왜 배우는지
 - ...

과목 설명-Overview

2015년 2학기 컴파일러 과목 개요

- 시간: 월, 수
- 장소: 공5404
- 대상: 컴퓨터공학과 3학년
- 교재:
 - 컴파일러 입문 (개정판), 오세만저, 정익사
 - Compiler Construction-Principles and Practice, K.C. Louden
 - Modern Compiler Implementation in Java (Basic Technique), Andrew W. Appel
 - Compilers: Principles, Techniques, and Tools, Jeffrey D Ullman
 - 기타 : 웹에 있는 관련자료들, 형식언어, 중간언어, 최적화, 머신코드 관련자료들



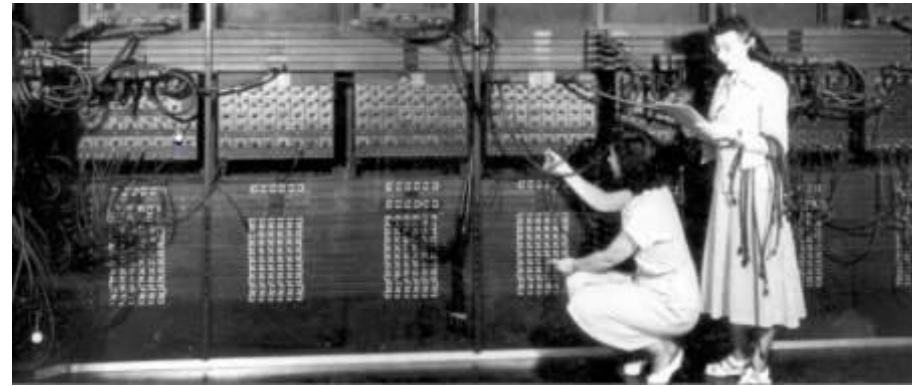
과목의 목표 (계획서)

1. 프로그래머의 의미전달을 담당하는 컴파일러에 대해 학습
2. 프로그래밍 언어 및 처리과정의 본질을 이해
3. 궁극적으로는 창의적인 프로그래밍을 지향
4. 작지 않은 크기의 프로그램 대한 체계적인 구조를 경험해 봄으로써 프로그래밍 실력을 한 단계 향상

컴파일러란?

컴퓨터에게 말 걸기

- 우선, **컴퓨터 말**을 쓰면
 - 사람 : '0010'! (덧셈해!)
 - 컴퓨터 : '101'.. (싫은데..)
- 불편하다:
프로그램 = 0/1과의 전쟁



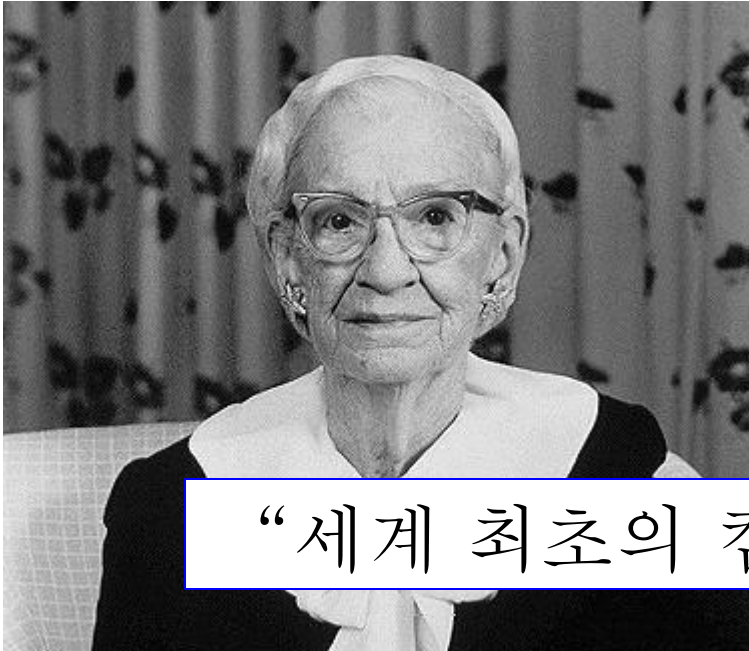
<ENIAC 코딩 - 스위치로 on/off>

- **어셈블리어**
 - 사람은 직접 2진수를 쓰지 않고 기호로 일단 표기
 - **ADD -> 0010** (즉, 덧셈)
 - **SUB -> 0101** (즉, 뺄셈)
 - 번역은? 어셈블러
 - 번역만 전담하는 소프트웨어
 - 그런데 CPU 다른 거 끼우면 다시 짜야 한다.
 - 예) X86용, ARM 용...

고급 언어

- 0/1 신경도 안 쓰고,
- 기계마다 다르지도 않고,
- 영어와 아주 비슷한 언어
- C, FOTRAN, BASIC, PASCAL, Java, C#, ... (종류가 많다)
- 번역하는 프로그램이 조금(!) 복잡한 일을 해야 한다..
 - 이 때의 번역기를 ‘컴파일러’라고 한다.

그레이스 호퍼 (Grace Hopper, 1906 ~ 1992)



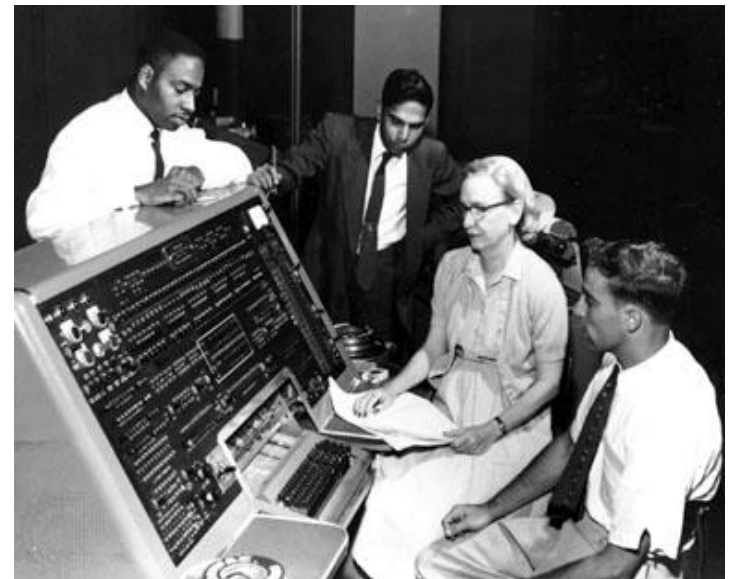
- 미국 할머니..
- 미 Vassar 대학 수학과 부교수
- 해군 제독
- 세계제2차대전 참전
- 미 병기국 계산팀 배치
- 프로그래머, 시스템엔지니어

“세계 최초의 컴파일러 제작자!” 용어의 창

- COBOL 탄생에 큰 영향을 줌
- ‘**컴파일러**’라는 용어를 처음 사용. 그리고 ...

컴파일러 탄생의 배경

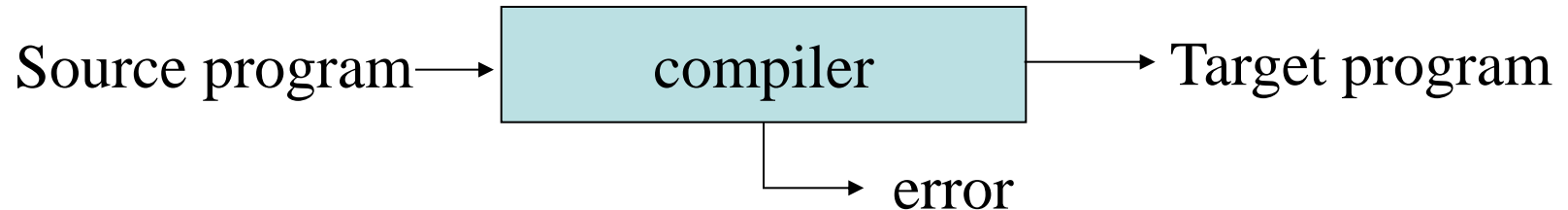
- **1950년대** 고급프로그래밍언어가 나타나던 시기
 - 지금도 유명한 것들: FORTRAN, LISP, COBOL, ALGOL ..
- **1952년** 그레이스 호퍼가 자신이 개발한 UNIVAC용 프로그래밍 언어 A-0를 기계어로 번역하는 컴파일러개발
 - 영어로 코딩할 수 없을까 고민함
 - 한사람이 6개월걸리는 미분해석기 프로그래밍을 18분만에 해결
- **1957년** 최적화 기능이 탑재된 포트란 컴파일러 등장
IBM의 존 배커스 개발



<UNIVAC 코딩- 오른쪽 두번째가 호퍼>

컴파일러의 정의

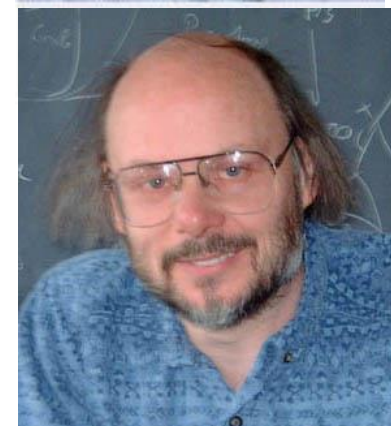
- 한 언어로 쓰여진 프로그램(source program) 을 읽어서
- 다른 언어로 된 의미가 같은 프로그램 (target program)으로 번역해주는
- 프로그램



Note) 저급언어로 번역하면 *compiler*,
동일 언어로 옮기면 *rewriter*,
고급언어로 번역하면 *decompiler*.

컴파일러를 만들었던 주요 인물들

- 데니스 리치 (→→→)
 - 1972년경 Sun사 근무시절 Unix를 만들다보니 어셈블리어 쓰기 귀찮아서 아예 언어를 만들었는데 그게 C
- C의 전작은 B!
 - (by 켄 톰슨 →)
- 브야네 스트롭스트롭
 - 1987년경 C++ 만듦 (→→)
- 리차드스톨만 (→)
 - 1987년 GNU 프로젝트의 컴파일러로 gcc 제작



컴파일러를 만들었던 주요 인물들 (계속)



- 제임스 고슬링(←)
1994년 경 Java 언어 발명, 컴파일러
와 virtual machine 개발
- 앤더스 헤일스버그 (→)
 - 2000년 C# 개발팀 주요 인물
 - 그 이전 이미 Turbo Pascal, Delphi 등 개발



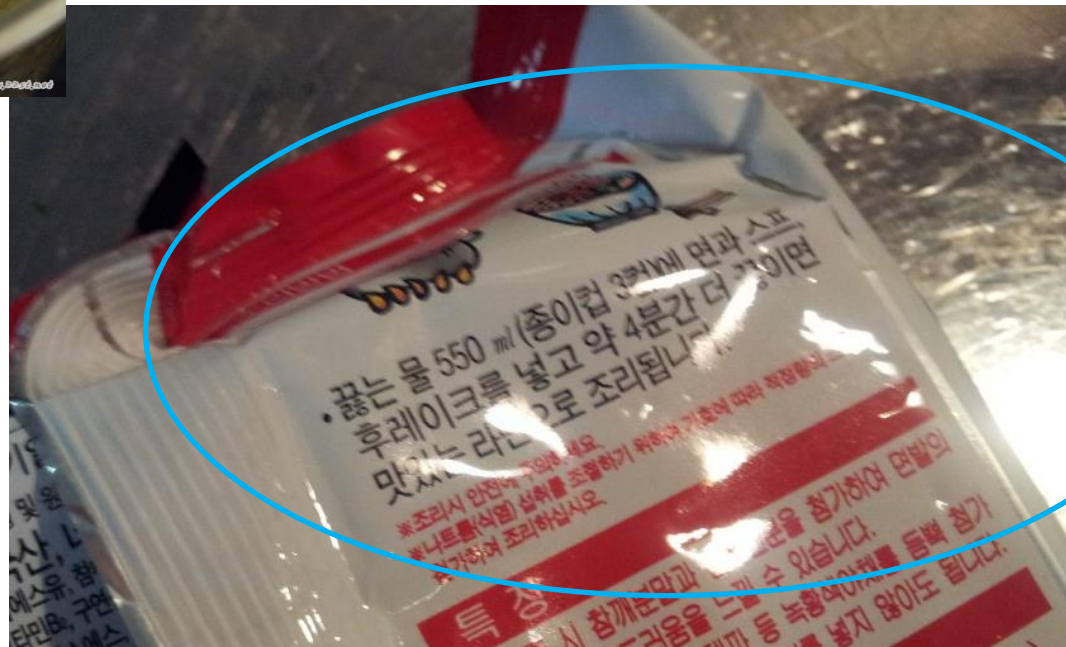
- ??????? (→)
 - 그 다음?



가능성...

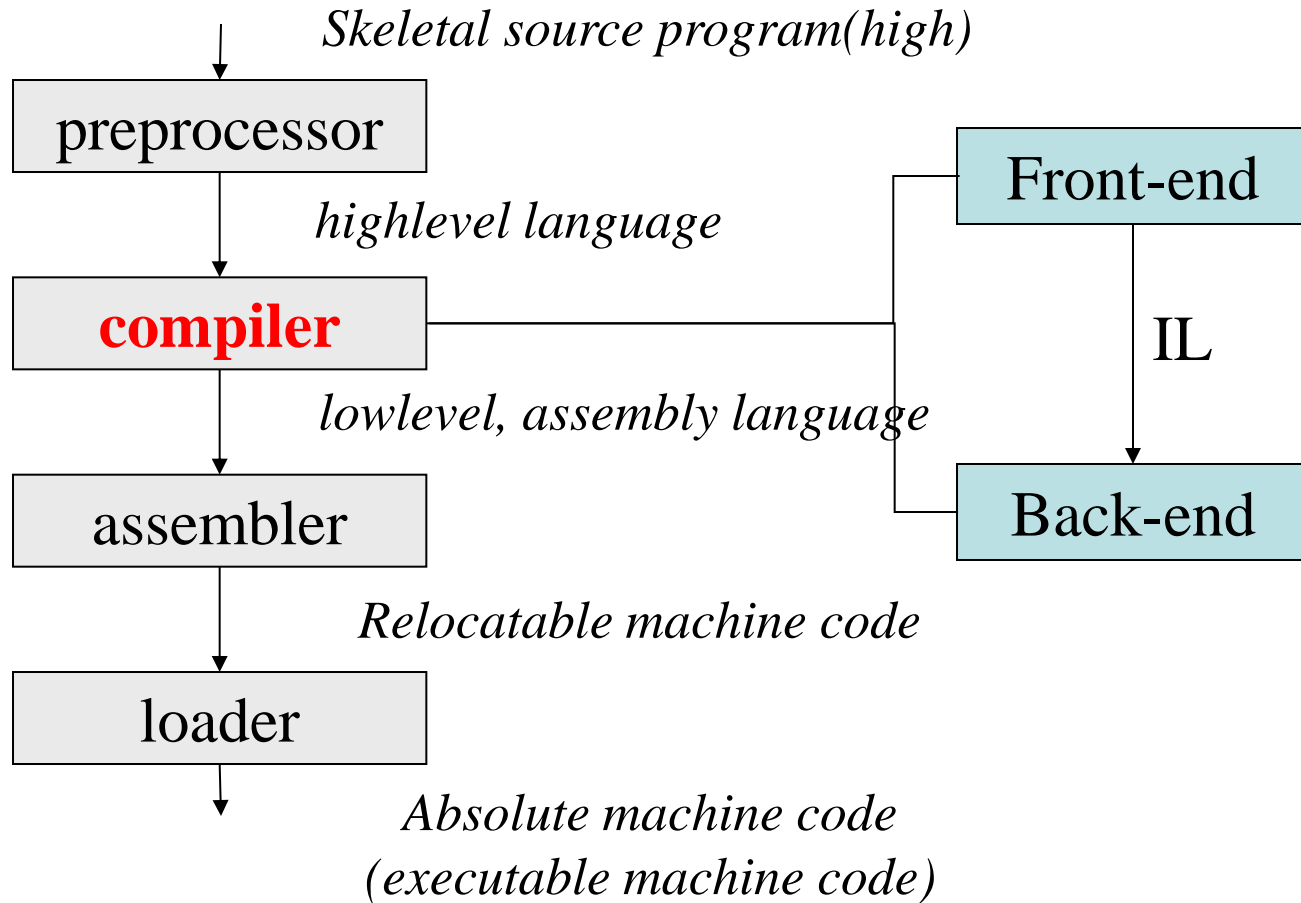
[복습] 컴파일러와 인터프리터의 차이

- 컴파일러식으로 라면 끓이기
: 끓이는 방법을 다 읽고 외어서
- 인터프리터식으로 라면 끓이기
: 끓이는 방법을 한 줄씩 읽고 실행

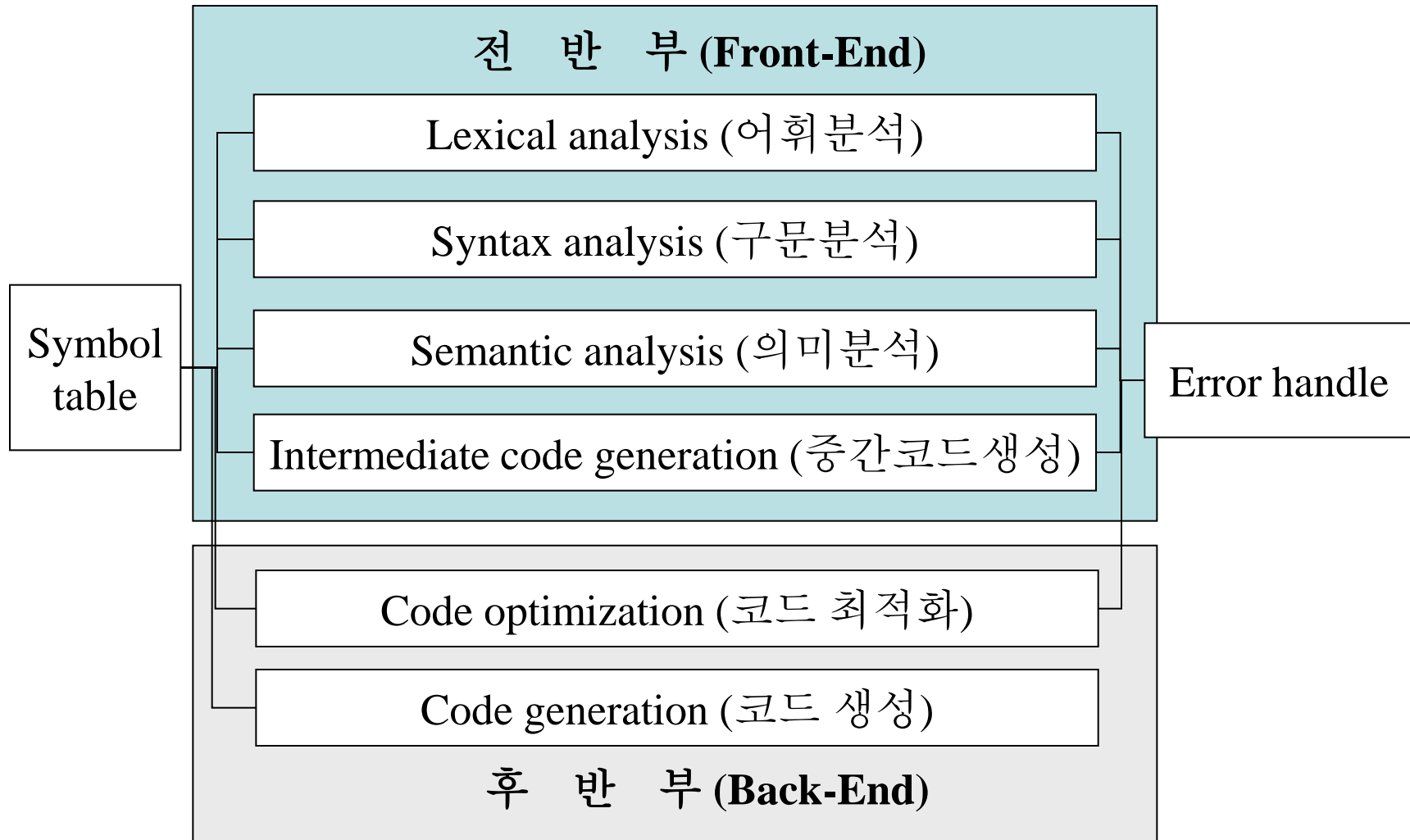


컴파일러의 처리 과정

자세한 프로그램 처리과정



컴파일러의 각 단계

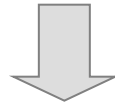


- Lexical analysis (어휘분석)
 - source program을 읽어서 문법의 최소 단위인 token (토큰) 을 생성하는 일
 - 토큰 (token): 키워드, 연산자, 식별자, 상수 ...
 - $A := B + 3 ;$ (token의 개수 : 6개)
 - A, B (variable),
 - $:=$ (assignment symbol)
 - $+$ (plus operator)
 - 3 (numeric)
 - $;$ (delimiter)

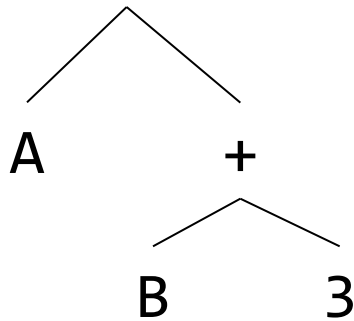
- Syntax analysis (구문분석)

- Token을 읽어 오류를 검색하고 올바른 문장에 대한 구문구조를 만든다.
- 어떻게? 주로 트리 형태

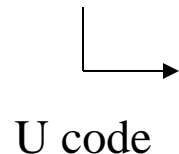
A := B + 3 ;



:= 추상구문트리



- Semantic analysis (의미분석)
 - Type checking(형 검사)
 - 각 연산자가 원시 언어의 정의에 맞는 피연산자를 가지는가를 검사
- Intermediate code generation (중간코드생성)
 - 구문구조를 이용하여 코드 생성 또는 문법규칙에 의해 생성
 - 예) **A := B + 3;**



load	1	2
loc	3	
add		
str	1	1

- Code optimazation (코드 최적화)
 - 코드 분석 후 시간, 공간 등 최적화
 - 선택적 단계 (그러나, 공간적, 시간적 효율화를 위해 필수적)
- Code generation (코드 생성)
 - 목적코드 생성
 - assemble language, machine code

프로그램을 완전히 분해, 분석하게 됨

position := initial + rate * 60

Lexical analyzer

id1 := id2 + id3 * 60

Syntax analyzer

id1
 :=
 id2 + id3 * 60

Semantic analyzer

id1
 :=
 id2 + id3 * inttoreal
 60

Intermediate code generator

temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3

Code optimizer

temp1 := id3 * 60.0
id1 := id2 + temp1

Code generator

MOVF id3, R1
MULTF #60.0, R1
MOVF id2, R2
ADDF R2, R1
MOVF R1, id1

Symbol Table

1	position	...
2	initial	...
3	rate	...
4		

다음에서 오류가 발생된다면, 어휘분석, 구문분석, 의미분석 중 어느 단계에서 발생했을까?

```
int a;  
a = 1.0;
```

```
int a; b  
b = a;
```

```
{ int a;  
  a = 1;  
}  
{ a = 2;  
}
```

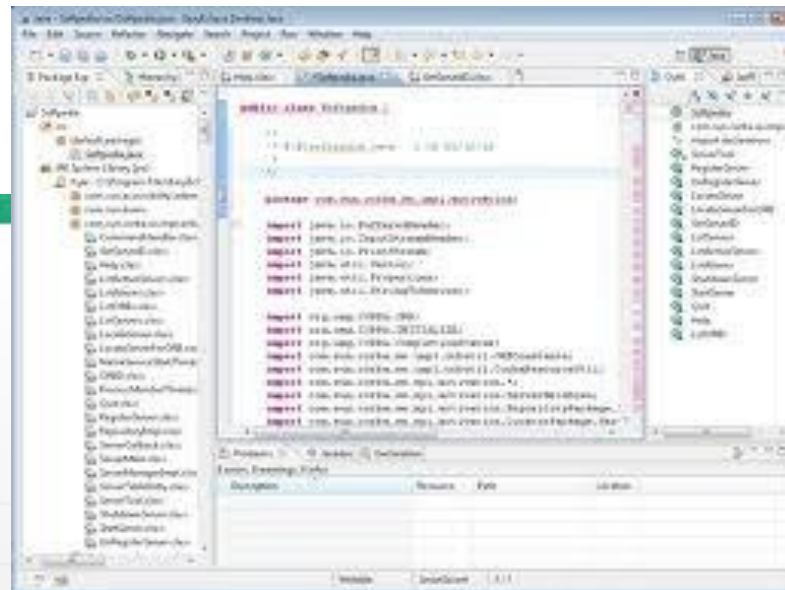
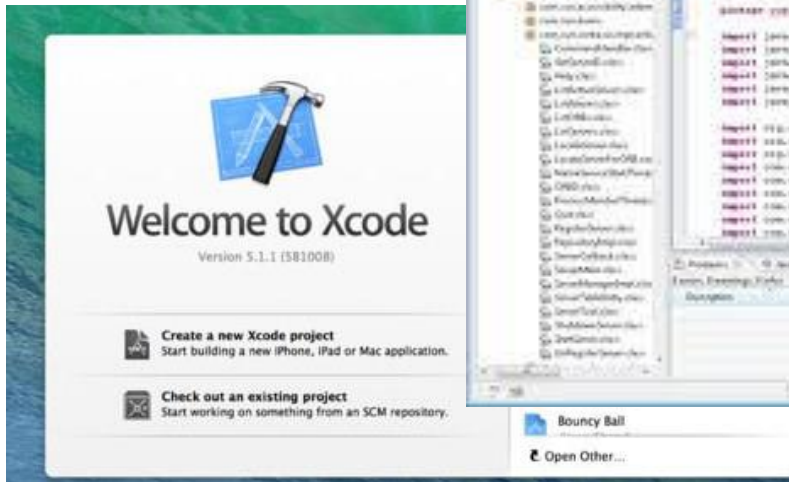
```
in a;  
a = 1;
```

```
int foo(int a)  
{  
  foo = 3;  
}
```

컴파일러 기술의 응용

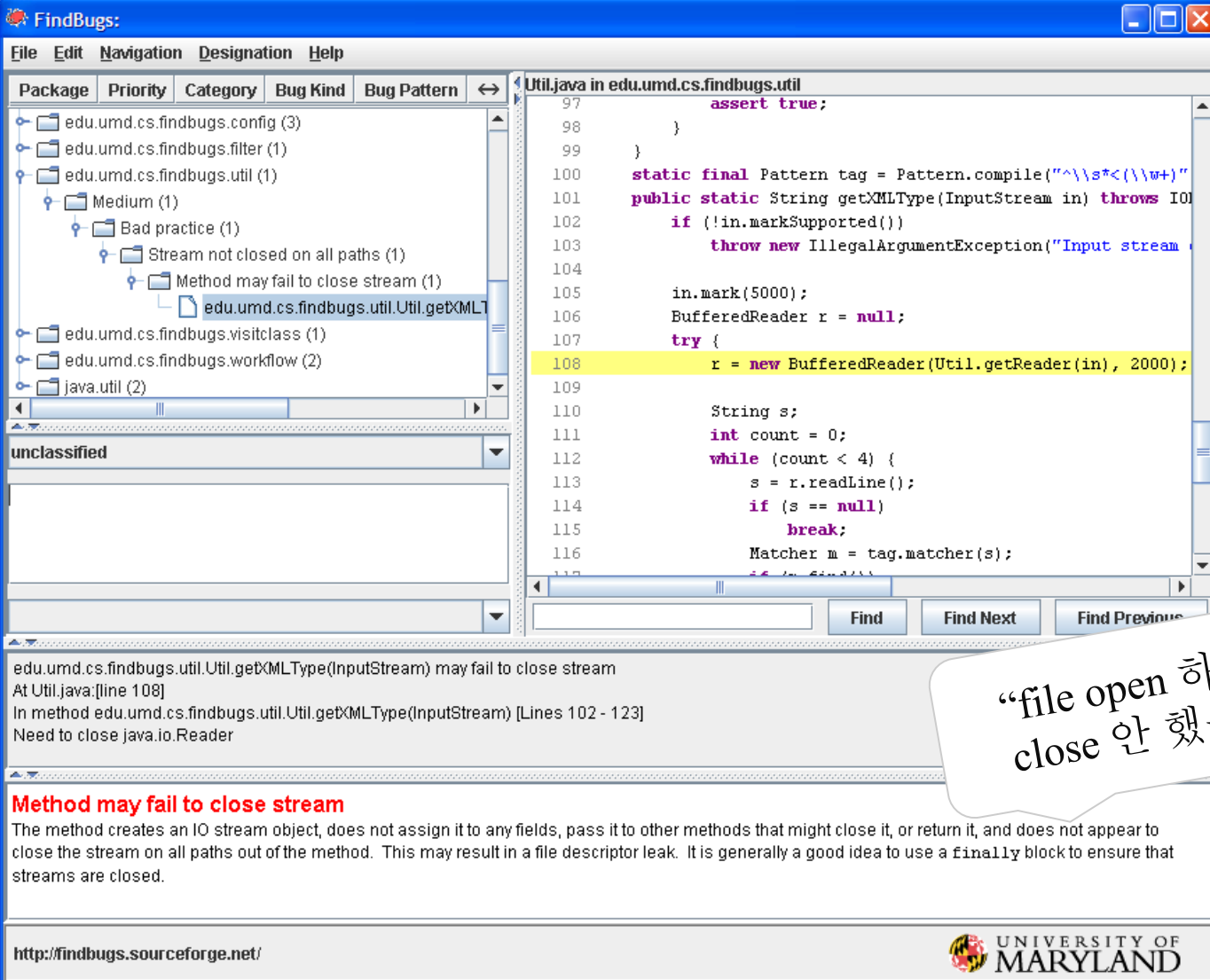
컴파일러 기술의 응용 1-IDE

- 컴파일러 자체도 만들 수 있지만
- 편리한 IDE를 제작하는데에도 중요한 기술임



컴파일러 기술의 응용 2-버그잡는 도구

FindBug-<http://findbugs.sourceforge.net/>



The screenshot shows the FindBugs application window. The left pane displays a project tree with the following structure:

- edu.umd.cs.findbugs.config (3)
- edu.umd.cs.findbugs.filter (1)
- edu.umd.cs.findbugs.util (1)
 - Medium (1)
 - Bad practice (1)
 - Stream not closed on all paths (1)
 - Method may fail to close stream (1)
 - edu.umd.cs.findbugs.util.Util.getXMLType
- edu.umd.cs.findbugs.visitclass (1)
- edu.umd.cs.findbugs.workflow (2)
- java.util (2)

The right pane shows the source code of `Util.java` in `edu.umd.cs.findbugs.util`. The code is as follows:

```
97     assert true;
98     }
99     }
100    static final Pattern tag = Pattern.compile("^\\s*<\\w+");
101    public static String getXMLType(InputStream in) throws IOException {
102        if (!in.markSupported())
103            throw new IllegalArgumentException("Input stream does not support mark");
104
105        in.mark(5000);
106        BufferedReader r = null;
107        try {
108            r = new BufferedReader(Util.getReader(in), 2000);
109
110            String s;
111            int count = 0;
112            while (count < 4) {
113                s = r.readLine();
114                if (s == null)
115                    break;
116                Matcher m = tag.matcher(s);
117                if (m.matches())
118                    return m.group(1);
119            }
120        } finally {
121            if (r != null)
122                r.close();
123        }
124    }
125 }
```

The bug report at the bottom of the window is as follows:

edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) may fail to close stream
At Util.java:[line 108]
In method edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) [Lines 102 - 123]
Need to close java.io.Reader

Method may fail to close stream

The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a `finally` block to ensure that streams are closed.

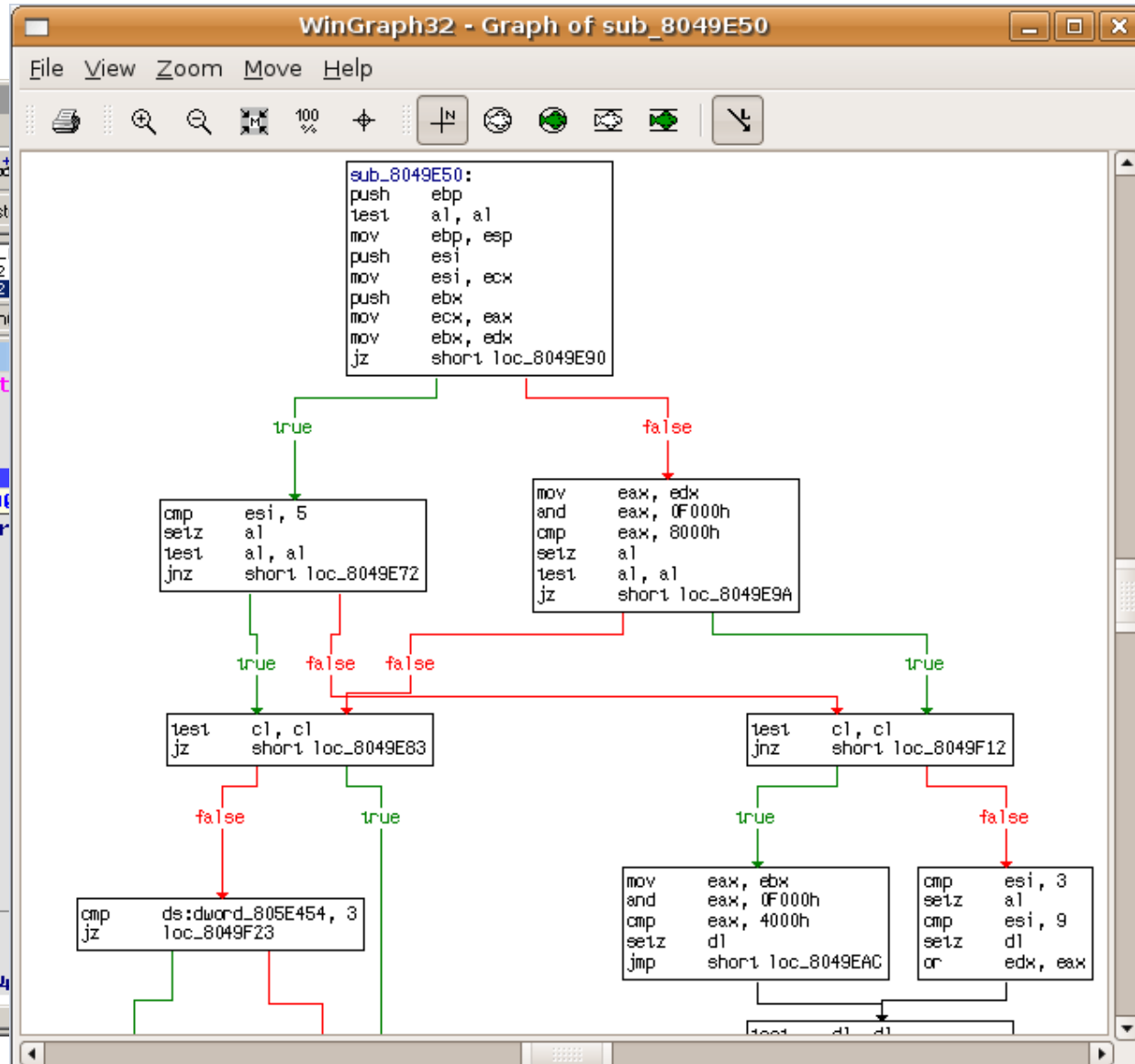
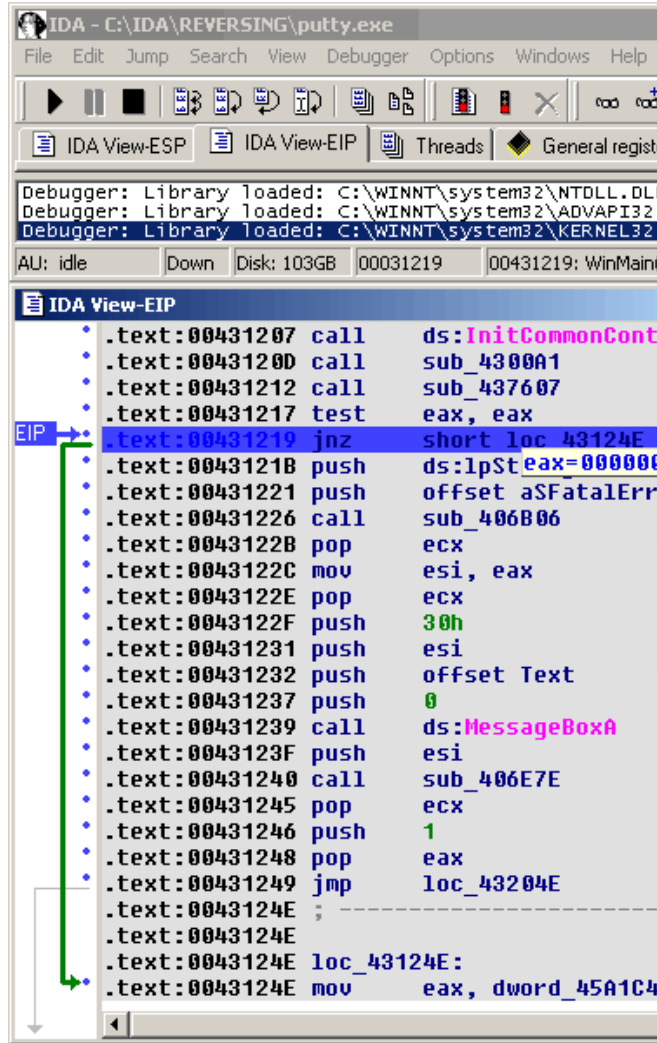
<http://findbugs.sourceforge.net/>

UNIVERSITY OF MARYLAND

“file open 하고
close 안 했음”

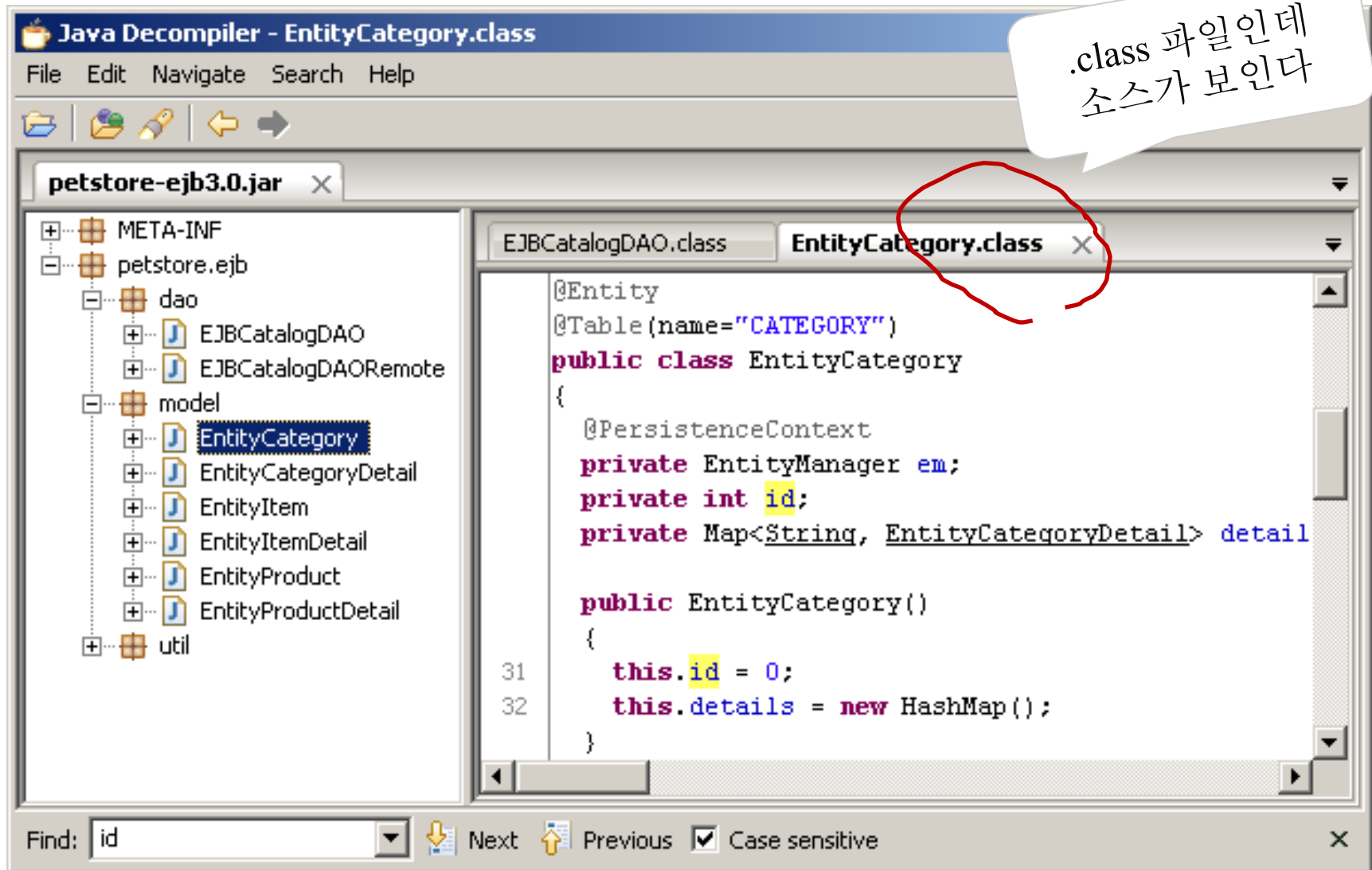
컴파일러 기술의 응용 3-바이너리 분석

IDAPro- <https://www.hex-rays.com/>



컴파일러 기술의 응용 4-디컴파일러

JD Decompiler – <http://jd.benow.ca/>



컴파일러 기술의 응용 5-국내 전문회사들



PRODUCTS

SPARROW



국내 최초·최고 기술력의 실행의미 기반 정적 분석도구
SPARROW로 무결점 소프트웨어 개발의 꿈을 실현합니다.


Print

02) 300-9101

SPARROW는 정적 프로그램 분석(Static Program Analysis)이론을 이용하여, 개발이 완료되기도 전에 프로그램의 치명적인 오류를
검출하는 소스코드 분석 도구입니다.

www.etnews.com/20140311000124



<코드마인드는 정보유출 자동분석도구 '코드마인드 프라이버시'로 안드로이드앱 3만개를 조사했다. 전체 24%에서 정보가 유출되는 것으로 드러났다.>

전자신문이 코드분석전문기업 코드마인드(대표 신승철)에 의뢰해 2월부터 3월까지 구글플레이에 등록된 안드로이드 앱 3만개를 분석한 결과, 24.3%에서 정보가 유출되는 것으로 나타났다. 국내

교과목으로서의 컴파일러

관련 교과목, 필수 기술

- 기본적인 Java 프로그래밍 능력
- 새로운 tool 설치 구동 능력
- 간단한 어셈블리어 이해 (미리 공부해오라는 것은 아님. 이해할 수 있는 능력만 있으면 됨.)
- tree, hash table, graph 등의 자료구조에 대한 기초적 이해
- 프로그래밍언어론? 계산이론?
 - 유리하긴 함, 그러나 그게 전부는 절대(!) 아님

Compiler를 과목으로 배우는 이유 (from U. of Washington)

- 좀 더 실력 있는 프로그래머가 되기 위해
 - 프로그래밍 언어와 하드웨어 사이의 관계를 잘 이해하고
 - 내 프로그램이 실제 구현되는 방식을 이해하고
 - 디버깅도 더 잘하고, 내 코드 자체에 대해서도 더 잘 이해하고
- 여러 군데서 기술이 쓰이므로 (parsing, software tools, database query engines, text processing, s/w and h/w checking tools, multicore compiler, domain specific compiler...)
- 수학적 이론과 실제 사용되는 것이 밀접하게 연결되면서 어려운 문제를 해결해야 하는 대표적 분야
 - parsing, analysis, optimization...
- 컴퓨터 분야의 각 주요 아이디어들을 집대성한 분야

홈 » Forums » 공부 » 프로그래밍 QnA

컴파일러 제작

글쓴이: **jeromechoi** 작성 일시: 일, 2008/02/18
프로그래밍 QnA

컴파일러의 매력에 빠져 살고 있습니다;;;
Louden씨의 **Compiler Construction**를 보며
현 해 보았습니다만 이제 본격적으로 빠지고 싶은
back-end(어느 번역본에선 후위부라고 설명해
다;;;에 관해 좀 더 확실히 알고 싶은데
죄다... 복잡하므로 생략하겠습니다라고 하네요.

제가 원하는 최종 목표는 **Virtual machine**에서
진!정!한! 컴파일러 제작인데.... **target machine**

그래서 혹시 서적이거나 그런 관련 자료가 있으시면
니당~♡

컴파일러를 만들어 볼까..

글쓴이: **나빌레라** 작성 일시: 금, 2009/02/06 - 2:15오전

운영체제를 만든지 일년이 넘었다.
일년동안 그걸 가지고 책도 썼고 조만간 출판될 예정이다.

어디선가 본 이야기이다.
시스템 프로그래밍에는 세 마리 용(드레곤)이 있다고 한다.

운영체제
컴파일러
데이터베이스

이 세 마리 용을 다 잡으면 드레곤 슬레이어가 된다고 한다.

운영체제는 하나 잡았으니, 이제 컴파일러 차례다.

사실 컴파일러를 만들어볼 계획은 지난 겨울 운영체제를 완성하는 순간
다.

다만 어떻게 해야 할지 감을 잡지 못하고 있었을 뿐.

컴퓨터 내공도 쌓긴 해야겠

t++ 사람들이 모이는 김에 **compiler**부터 시작하자.

이왕이면, 자원이 좋아야겠다.

워싱턴대 동영상 강의가 좋겠다.

http://dada.cs.washington.edu/dl/course_index.html 들 중 최신

참고: 프로그래머가 이력서 읽는법

Another resume tip

by Joel Spolsky

Friday, January 02, 2009

Are you a software developer applying to a small company?

Here's a tip from someone who has read thousands of resumes. When you're applying to a startup, or a software company with less than, say, 100 employees, you may want to highlight the Range Out Code parts of your experience, while parts of your experience.

When a startup CTO sees things like:

- Responsible for \$30
- Architected new ER
- Managed team of 25
- Optimized business

they think, "Spare me, the somebody running around and optimize and architect need someone who isn't a code." Here's the stuff CTO want to see on a resume:

Joel on Software

Getting Your Résumé Read

by Joel Spolsky

Monday, January 26, 2004

I've been going through a big pile of applications for the summer internship positions at Fog Creek Software, and, I don't know say this, some of them are really, really bad. This is not to say applicants are stupid or unqualified, although they might be. I'm going to find out, because when I have lots of excellent applications and only two open positions, there's really no need to waste time

Rhino on Rails

Rich Programmer Focus

That Old Marshmallow

Tuesday, September 2, 2003

Ten Tips for a (Slightly) Less

How a programmer reads your resume



personal* opinions, not Google that most resume screeners don't build their own software. Microsoft or Google — with miced screeners disagree on it my own opinions. These tips are not results. Your mileage may vary. Do not use nding in a pool of water. Do not tap on the on not feed the tips. Etc

듣을까 말까 매우 고민하던 과목인데 후배들에게도 적극 추천하고 싶습니다. 정말 컴공이 아니면 절대 어디서 쉽게 접할 수 없는 과목인 것 같습니다.

컴파일러란 단어가 단순히 사용만 하는 것으로 막연했었는데 이것들이 내부적으로 동작하는 것을 배울 수 있어서 좋았습니다.

모든 언어의 기본원리를 탐구하는 내용이라 의미도 모르고 그냥 사용했던 문장들을 새롭게 알 수 있었다

강의 자체나 교과목의 중요성이나 절대 빼놓을 수 없는 과목, 감사합니다 ^^

컴파일러라는 과목에 대해서 더 많이 배우고 싶은 생각이 드는 과목이었습니다.

과제가 많아서 힘든긴 했지만 역시 과제가 많아야 배우는 게 많은 것 같습니다.

프로그래머라면 한번쯤은 들어봐야 할 중요한 과목이라 생각합니다. 감사합니다

계산이론을 들어서 조금 수월할 줄 알았는데 생각보다 어려운 점이 많았습니다

강의 중간중간 예제가 많아 강의를 듣고 바로 적용해보고 확인하는 작업이 수업시간 안에 이루어지니까 다시 복습을 할 때도 큰 무리 없이 복습을 할 수 있다.

수업시간에 문제를 풀어보는 것이 좋습니다. 잘 이해했는지 확인해볼 수 있었습니다.

과제를 하면서 lex와 yacc등 여러가지 새로운 것을 접할 수 있어서 재미있었습니다..

수업은 좀 빠르게 나가셨지만 많이 배우고 학기를 마감하는 것 같습니다. 실습을 완벽히 따라가지는 못했지만, 책에 소스가 많이 있어서 아예 힘들어서 손을 놓지는 않았던 것 같습니다.

과제에서 따라가기 힘든 부분들 특히 C 코드의 부분에서 어려움을 겪었지만, 유익하고 유용한 부분을 다룰 수 있어서 좋은 수업이었던 것 같다.

많이 힘든 것이 사실이었습니다. 하지만 그만큼 얻은 것도 정말 많은 강의였다고 생각합니다.

교수님 말씀으로는 앞에 배우는 것이 더 쉽다고 하셨는데, 제겐 뒤부분이 더 어려웠던 것 같습니다.

강의자료가 적극적으로 참여할 수 있도록 되어있습니다.

ㅠ 아직 마지막 과제 하고 있는데 ㅠ울 것 같아요

그냥 사용하기만 하던 컴파일러에 대해 자세히 배울 수 있는 시간이었다.

과제가 너무 어려워요 ㅠ 하지만 이론만 하는 것 보다는 많은 것을 얻을 수 있는 것 같습니다.

e-learn으로 다시 들을 수 있다는 것도 참 좋은 것 같습니다.

본인이 열정만 있다면 해당 과목 뿐만이 아닌 더 많은 영역을 익힐 수 있는 수업이었습니다.

컴파일러는 인터프리터와는 확실히 다르게 어려웠지만 재미있었습니다.

아침수업이 힘들었어요ㅕㅕ.

수업을 들은 후 (선배들..)

과제 (계획)

- 과제 (계획)

1	가칭 W 소스 파일을 C 코드로 바꾸는 프로그램 구현
2	ANTLR를 이용하여 Mini C 프로그램의 어휘분석기, 구문분석기를 구현
3	ANTLR를 이용하여 만든 Mini C 파서에 AST를 만드는 코드를 추가
4	Mini C코드를 입력받아 U-코드로 변환 시켜주는 translator 작성

- 구 Project (2014)

Mini C to U-코드 translator를 수정, 개선한다.

- 예) x86, 64, ARM 어셈블리로 변환, gcc GIMPLE이나 LLVM BIT 코드 등 U-코드가 아닌 다른 중간코드로 변환하고 최적화해보기, 타입검사 추가 ...

성적 산정

- 시험 70%
 - 중간/기말고사
 - 퀴즈: 1회 이상
 - 시험 중 부정행위와 유사한 행위만 보여도 0점 처리
- 과제물 등: 20%
 - 5회 이상 부여 예정
 - 마지막 과제는 설계 프로젝트 (공학인증 설계 과목)
 - 과제 사용 언어 : Java, c
 - Copy는 엄중 처벌
- 출석/발표/기타: 10%
 - 주의: ¼ 이상 출석하지 못하면 F

기타

- 과목 웹 사이트 :
 - <http://e-learn.cnu.ac.kr> 강의 슬라이드, 토론, Q&A 등
(cyber 강의 → 보면 정말 이익!)
 - <http://www.cs.washington.edu/education/courses/csep501/09au/> (미 워싱턴대학 동영상강의 → 좀 이론적인 부분을 보완하고 영어실력을 기르려면..)
- 담당교수연구실 :
 - 공학5호관 5524호 (구내전화 6857) 혹은 5502호 PLAS 연구실(구내전화 7450)
 - 이메일 eschough@cnu.ac.kr

첫 번째 주제

- Lexical analysis (어휘 분석)