프로그래밍

C Programming

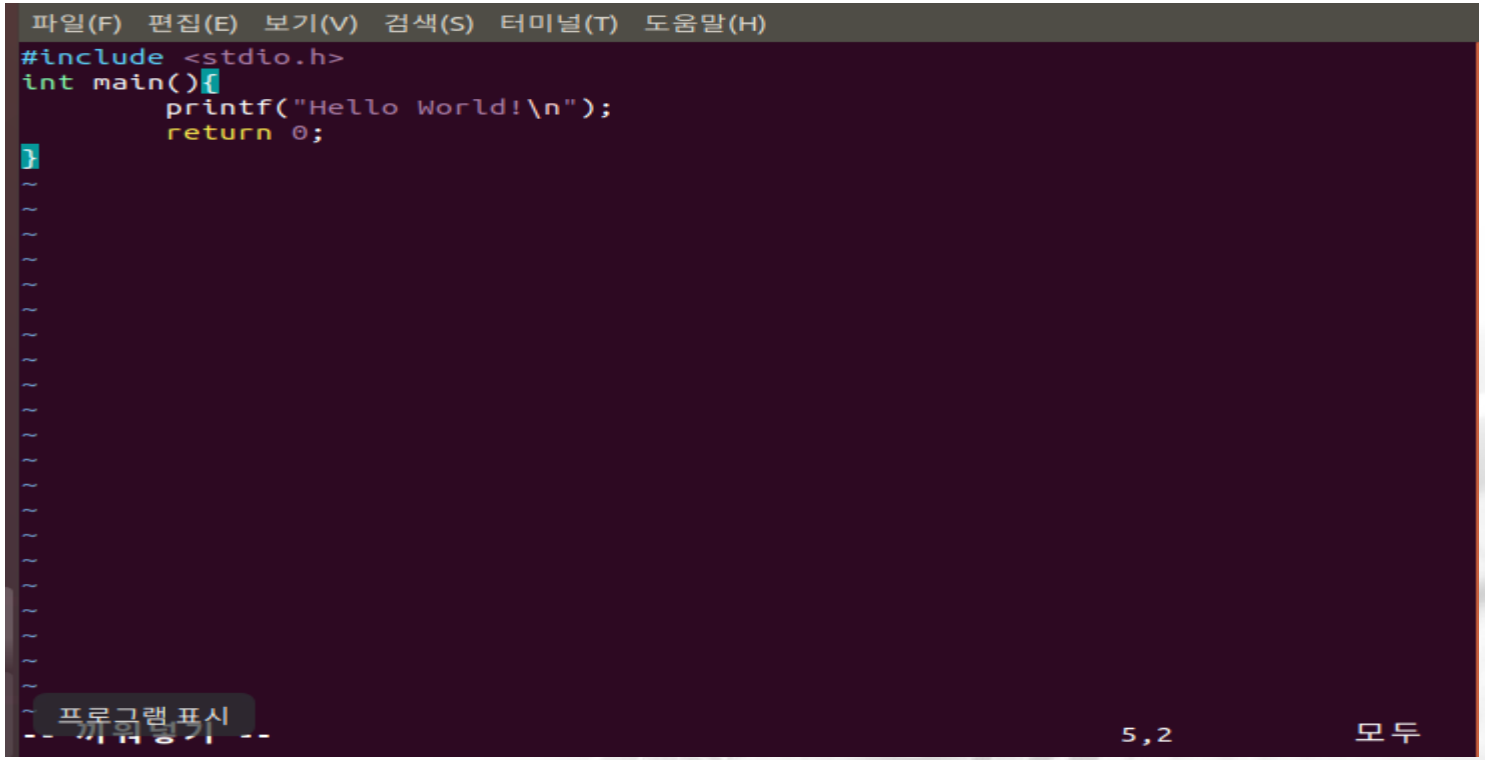
- GCC
  - ✓ 오픈 소스 개발자들이 리눅스 용 컴파일러를 개발하였는데 이것이 GCC(GNU Compiler Collection)
  - ✓ 우분투 배포판에는 기본적으로 GCC 가 설치되어 있는데 확인은 터미널을 열어서 gcc를 입력

```
park@UbuntuPark:~$ gcc
gcc: fatal error: no input files
compilation terminated.
park@UbuntuPark:~$
```

  - ✓ 설치
    - sudo apt-get install gcc

C Programming

- GCC
  - ✓ 소스 코드 작성
    - vim helloworld.c 를 입력하고 입력 모드로 전환(i)한 후 작성
    - esc 키를 입력해서 명령 모드로 전환한 후 :wq 를 입력하고 종료



C Programming

- GCC
  - ✓ 컴파일
    - gcc -o helloworld helloworld.c
  - ✓ 실행
    - ./helloworld

```
park@UbuntuPark:~$ vim helloworld.c
park@UbuntuPark:~$ gcc -o helloworld helloworld.c
park@UbuntuPark:~$ ./helloworld
Hello World!
park@UbuntuPark:~$
```

C Programming

- make
  - ✓ 실제 패키지는 많은 파일로 복잡하게 구성되어 있기 때문에 gcc로 일일이 컴파일하여 하나의 실행 파일로 만드는 것은 매우 번거로운 작업인데 이를 간단하게 해결할 수 있도록 해주는 것이 make 명령
  - ✓ make 명령은 makefile(또는 Makefile)에 설정된 정보를 읽어서 여러 소스 파일을 컴파일하고 링크하여 최종 실행 파일을 만듬
  - ✓ 소스로 배포되는 많은 오픈 소스 소프트웨어는 소스 코드와 함께 makefile을 배포
  - ✓ make 명령 설치
    - make 명령이 설치되어 있는지 확인하고 만약 make 명령이 없다면 설치

```
adam@adam:~$ make
Command 'make' not found, but can be installed with:
sudo apt install make          # version 4.3-4.1build1, or
sudo apt install make-guile  # version 4.3-4.1build1
adam@adam:~$ sudo apt install make
```

C Programming

- make
  - ✓ 실습
    - one.c

```
#include <stdio.h>

extern int two();

int main() {
        printf("Go to Module Two—-₩n");
        two();
        printf("End of Module One.₩n");
}
```

C Programming

- make
  - ✓ 실습
    - two.c

```
#include <stdio.h>
int two() {
        printf("In Module Two--₩n");
        printf("--- This is a Module Two.₩n");
        printf("End of Module Two.₩n");
}
```

C Programming

- make
  - ✓ 실습
    - makefile(작성 시 들여쓰기는 탭 사용)

```
TARGET=one
OBJECTS=one.o two.o
${TARGET} : ${OBJECTS}
	gcc -o ${TARGET} ${OBJECTS}
one.o : one.c
	gcc -c one.c
two.o : two.c
	gcc -c two.c
```

C Programming

- make
  - ✓ 실습
    - ● 실행

```
adam@adam:~$ make
gcc -c one.c
gcc -c two.c
gcc -o one one.o two.o
adam@adam:~$ ./one
Go to Module Two—-
In Module Two--
--- This is a Module Two.
End of Module Two.
End of Module One.
```

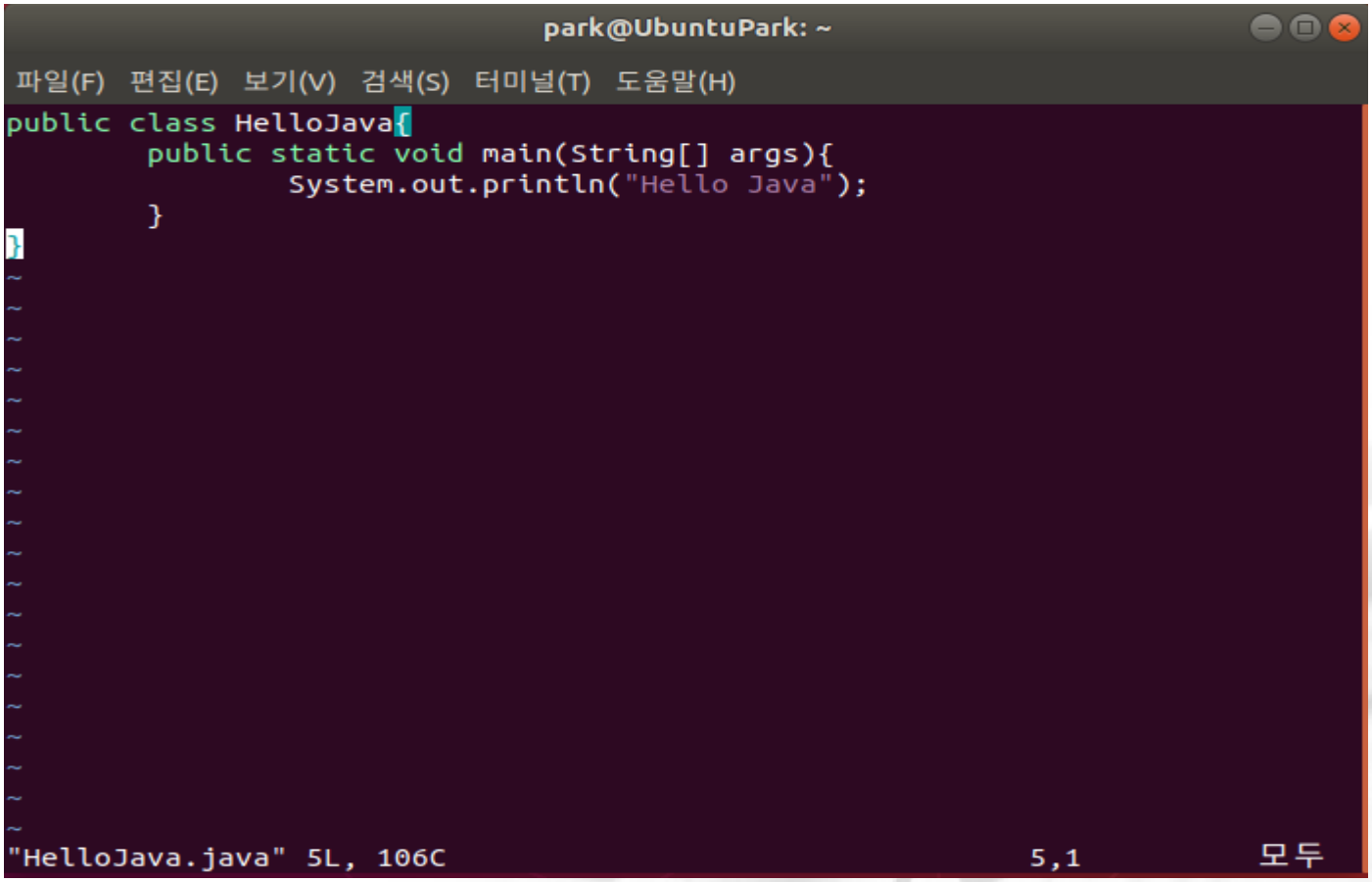
Java Programming

- ❖ Java 설치
  - ✓ 설치
    - sudo apt update # 패키지 목록 업데이트
    - sudo apt install openjdk-21-jdk # OpenJDK 21 설치
  - ✓ 설치 확인
    - java -version
    - javac -version

```
park@UbuntuPark:~$ java -version
openjdk version "1.8.0_265"
OpenJDK Runtime Environment (build 1.8.0_265-8u265-b01-0ubuntu2~18.04-b01)
OpenJDK 64-Bit Server VM (build 25.265-b01, mixed mode)
park@UbuntuPark:~$ javac -version
javac 1.8.0_265
park@UbuntuPark:~$
```

Java Programming

- Java 설치
  - ✓ 소스 코드 작성 : vim HelloJava.java



Java Programming

- Java 설치
  - ✓ 컴파일: javac HelloJava.java
  - ✓ 실행: java HelloJava

```
park@UbuntuPark:~$ vim HelloJava.java
park@UbuntuPark:~$ javac HelloJava.java
park@UbuntuPark:~$ java HelloJava
Hello Java
```

Python Programming

- ❖ 파이썬
  - ✓ 설치
    - 이미 설치 되어 있음: python3 --version
    - 업데이트: sudo apt-get upgrade python3
    - venv 모듈 설치: sudo apt install python3-venv

Python Programming

- ❖ 파이썬
  - ✓ 코드 작성 및 실행
    - 디렉토리 생성: mkdir pythontest
    - 디렉토리 이동: cd pythontest
    - 가상환경 생성: python3 -m venv myenv
    - 가상환경 활성화: source myenv/bin/activate
    - 코드 작성: main.py

```
# 학생 성적 리스트
scores = [85, 92, 78, 90, 88]
# 80점 이상인 성적만 필터링 (List Comprehension)
high_scores = [s for s in scores if s >= 80]
# 평균 계산
average = sum(scores) / len(scores)
print(f"고득점 리스트: {high_scores}")
print(f"전체 평균: {average:.2f}")
```

    - 실행: python3 main.py
    - 라이브러리 내보내기: pip freeze > requirements.txt

Node.js Programming

- 노드 설치
  - ✓ NVM 설치 스크립트 실행

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

  - ✓ 현재 셸에 NVM 적용

```
source ~/.bashrc
```

  - ✓ 최신 LTS 버전 설치

```
nvm install --lts
```

  - ✓ 설치 확인

```
node -v
npm -v
```

Node.js Programming

- 프로젝트 생성 및 실행
  - ✓ 작업 디렉토리 생성

    mkdir hello-node

    cd hello-node
  - ✓ 프로젝트 초기화 (기본값으로 생성)

    npm init –y

Node.js Programming

❖ 프로젝트 생성 및 실행

✓ 샘플 코드 작성(app.js)

```
// app.js
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain; charset=utf-8');
  res.end('안녕, Ubuntu Node.js 서버입니다!₩n');
});

server.listen(port, hostname, () => {
  console.log(`서버가 실행 중입니다: http://${hostname}:${port}/`);
});
```

Node.js Programming

- 프로젝트 생성 및 실행
  - ✓ 실행

    node app.js
  - ✓ 다른 터미널에서 127.0.0.1:3000 접속

    curl http://127.0.0.1:3000

Go Programming

- 설치
  - ✓ 기존에 설치된 go가 있다면 삭제

    sudo rm -rf /usr/local/go
  - ✓ x86_64인 경우 (Standard Intel/AMD)
    - Go 다운로드 (버전 숫자는 변경될 수 있음)

    curl -OL https://golang.org/dl/go1.22.0.linux-amd64.tar.gz
    - 압축 해제

    sudo tar -C /usr/local -xzf go1.22.0.linux-amd64.tar.gz
  - ✓ aarch64 (또는 arm64)인 경우
    - Go 다운로드 (버전 숫자는 변경될 수 있음)

    curl -OL https://golang.org/dl/go1.22.0.linux-arm64.tar.gz
    - 압축 해제

    sudo tar -C /usr/local -xzf go1.22.0.linux-arm64.tar.gz

Go Programming

- 설치
  - ✓ 어느 디렉토리에서나 go 명령어를 사용할 수 있도록 환경 변수를 설정
    - ~/.profile 파일 끝에 경로 추가

```
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.profile
echo 'export PATH=$PATH:$(go env GOPATH)/bin' >> ~/.profile
```

    - 변경사항 적용

```
source ~/.profile
```

  - ✓ 확인

```
go version
```

Go Programming

- 프로젝트 생성 및 실행
  - ✓ 작업 디렉토리 생성

    mkdir hello-go

    cd hello-go
  - ✓ hello-go라는 이름으로 모듈을 시작

    go mod init hello-go

Go Programming

❖ 프로젝트 생성 및 실행

✓ 소스 코드 작성(main.go)

```
package main

import "fmt"
func main() {
    fmt.Println("안녕하세요, Go 언어의 세계에 오신 것을 환영합니다!")

    // 간단한 산술 연산 예제
    a := 10
    b := 20
    sum := a + b
    fmt.Printf("%d + %d = %d₩n", a, b, sum)
}
```

✓ 실행

```
go run main.go
```

Rust Programming

- 설치
  - ✓ 시스템 아키텍처를 자동으로 감지하여 적절한 버전을 설치
    - curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
    - 명령을 수행하면 화면에 선택할 수 있는 선택지가 나오는데 1번 선택

    1) Proceed with standard installation (default - just press enter)

    2) Customize installation

    3) Cancel installation
  - ✓ 환경 변수 적용
    - source $HOME/.cargo/env
  - ✓ 설치 확인
    - rustc --version

Rust Programming

❖ 프로젝트 생성 및 실행

✓ 프로젝트 생성

cargo new hello_rust

✓ 소스 코드 확인

- cd hello_rust
- nano src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

✓ 실행

cargo run

Rust Programming

❖ 프로젝트 생성 및 실행

✓ 소스 코드 수정 - nano src/main.rs

```
fn main() {
    println!("안녕하세요, Rust의 세계에 오신 것을 환영합니다!");

    // 간단한 변수 사용 예제
    let a = 10;
    let b = 20;
    println!("{} + {} = {}", a, b, a + b);
}
```

✓ 실행

```
cargo run
```