Lab 1: ELF Dependency Analyzer – `bldd`

Description

`bldd` (backward `ldd`) is a command-line tool that scans a directory for ELF binaries and identifies which shared libraries they depend on. It supports multiple architectures including `x86`, `x86_64`, `armv7`, and `aarch64`, and can generate reports in both **text** and **PDF** formats.

Implementation

The application is implemented in modular Python and is divided into the following components:

- `cli.py` – handles argument parsing and help output.
- `scanner.py` – scans directories for ELF binaries and gathers library usage.
- `elf_utils.py` – detects ELF file format and extracts architecture + dependencies.
- `reporter.py` – generates the final usage report in `.txt` or `.pdf`.
- `__main__.py` – the entry point that glues all components together.

Specification

- **Supported Architectures:** `x86`, `x86_64`, `armv7`, `aarch64`
- **Input Types:** Any directory containing ELF executables
- **Flags:**
  - `--libs`: Target libraries (e.g. `libc.so.6`)
  - `--dir`: Directory to scan
  - `--out`: Report output file (e.g. `report.txt`)
  - `--format`: Output format (`txt`)
- **Output:** Sorted list of libraries used across binaries grouped by architecture

---

Help Page

```
$ python3 -m bldd --help

usage: __main__.py [-h] [--libs [LIBS ...]] [--dir DIR] [--out OUT] [--
format {txt,pdf}]

bldd - backward ldd tool

options:
  -h, --help           show this help message and exit
  --libs [LIBS ...]    Shared libraries to track (e.g. libc.so.6)
  --dir DIR            Directory to scan (default: current)
  --out OUT            Output report filename
  --format {txt,pdf}   Output format (default: txt)
```

Requirements

Python Dependencies

- `argparse`
- `os`, `collections`
- `lief`

ELF Test File Generation

To test `bldd`, ELF files were generated for multiple architectures:

Build Environment Setup

```
sudo apt install gcc-i686-linux-gnu gcc-arm-linux-gnueabi gcc-aarch64-
linux-gnu
```

Directory Layout

```
test_bins/
├── x86/
│   └── hello_x86
├── x86_64/
│   └── hello_x86_64
├── armv7/
│   └── hello_armv7
├── aarch64/
│   └── hello_aarch64
```

Hello World Binary Creation

```
echo '#include <stdio.h>\nint main() { printf("Hello\\n"); return 0; }' >
hello.c

gcc hello.c -o test_bins/x86_64/hello_x86_64
i686-linux-gnu-gcc hello.c -o test_bins/x86/hello_x86
arm-linux-gnueabi-gcc hello.c -o test_bins/armv7/hello_armv7
aarch64-linux-gnu-gcc hello.c -o test_bins/aarch64/hello_aarch64
```

Output Example (TXT)

```
python3 -m bldd --dir test_bins --libs libc.so.6 --format txt --out
bldd_report.txt
```

output:

```
---------- ARCH.I386 ----------
libc.so.6 (1 execs)
-> test_bins/x86/hello_x86

---------- ARCH.X86_64 ----------
libc.so.6 (1 execs)
-> test_bins/x86_64/hello_x86_64

---------- ARCH.AARCH64 ----------
libc.so.6 (1 execs)
-> test_bins/aarch64/hello_aarch64

---------- ARCH.ARM ----------
libc.so.6 (1 execs)
-> test_bins/armv7/hello_armv7
```

Link to Github Lab sloution