

Project 0

Advanced Compilers 2024

TA: Juneyoung Park (juneyoung.park@snu.ac.kr)

May 9, 2024

Due: May 19, 2024

In this project, we will setup LLVM environment and plot control flow graph (CFG) using LLVM. Since changing default compiler executable or option can affect whole system, it's always good to separate your base environment from experimental settings. We will use docker container for such isolation.

1 Environment setups

1.1 Docker

Please refer to the official docker's installation guide at <https://docs.docker.com/engine/install>. One can add a user into 'docker' group to grant permissions.

```
1 | sudo usermod -aG docker $(whoami)
```

Otherwise, use docker commands after 'sudo'. One can check running containers by

```
1 | docker ps
```

Also, one can open the bash shell on the container via

```
1 | docker exec -it <container_name> bash
```

1.2 LLVM 17

TL;DR : Run '.container/deploy.sh' or 'setup.sh'.

Former script will launch a container named 'advcmp' and do most of tedious setups for you, and let you jump to [section 2](#). If multiple people have to do project on the same server, please configure 'container_name' at 'deploy.sh'. For those who want to use one's own native ubuntu, run 'setup.sh'. The script will register repository and install llvm-17 toolchain for you.

In this semester, we will use LLVM 17. (As of May 2024, 17 in stable, 18 in qualification, and 19 in development.)

1.2.1 Using package manager (Ubuntu 22.04)

LLVM officially supports apt repository. <https://apt.llvm.org/>

- Add following repository links at '/etc/apt/sources.list.d/llvm.list'.

```
1 | # latest
2 | deb http://apt.llvm.org/jammy/ llvm-toolchain-jammy main
3 | deb-src http://apt.llvm.org/jammy/ llvm-toolchain-jammy main
4 | # 18
5 | deb http://apt.llvm.org/jammy/ llvm-toolchain-jammy-18 main
6 | deb-src http://apt.llvm.org/jammy/ llvm-toolchain-jammy-18 main
```

```

7 | # 17
8 | deb http://apt.llvm.org/jammy/ llvm-toolchain-jammy-17 main
9 | deb-src http://apt.llvm.org/jammy/ llvm-toolchain-jammy-17 main

```

- Get repository's signature.

```

1 | wget -q0- https://apt.llvm.org/llvm-snapshot.gpg.key | sudo tee \
2 | /etc/apt/trusted.gpg.d/apt.llvm.org.asc

```

- Update and install.

```

1 | sudo apt update
2 | sudo apt install -y libllvm17 llvm-17 llvm-17-dev clang-17 ...

```

Installing 'libllvm17', 'llvm-17', and 'llvm-17-dev' is mandatory. Also, we recommend you to install any version of 'lld', 'lldb', or 'clang'.

- Test

```

1 | llvm-config-17 --help

```

- (Optional) Symbolic links

One has to append version number to call installed LLVM or clang compiler. (e.g., `llvm-config-17`, `clang-17`) By using 'update-alternatives' in Ubuntu, symbolic links without tailing version number can be easily generated.

Please run the script '`.container/script/update-alternatives-llvm.sh`' **inside** the container. If you used given script ('`.container/deploy.sh`') to launch a container, you don't need to run the script manually.

1.2.2 Build from source

You will need 'cmake', 'ninja-build' and C/C++ compiler for bootstrapping build. Also, we recommend using 'ccache' and 'lld' to speed up build. For more details, please refer to <https://llvm.org/docs/CMake.html>

- Fetch source

```

1 | git clone https://github.com/llvm/llvm-project.git
2 | git checkout -t llvmorg-17.0.6

```

- Configure cmake

```

1 | mkdir llvm-project/build
2 | cd llvm-project/build

1 | cmake -G Ninja ../llvm \
2 |   -DCMAKE_CXX_STANDARD=17 \
3 |   -DCMAKE_C_COMPILER=clang \
4 |   -DCMAKE_CXX_COMPILER=clang++ \
5 |   -DCMAKE_BUILD_TYPE=Release \
6 |   -DCMAKE_INSTALL_PREFIX="/tmp/llvm" \
7 |   -DLLVM_ENABLE_RTTI=ON \
8 |   -DLLVM_TARGETS_TO_BUILD="host" \
9 |   -DLLVM_ENABLE_ASSERTIONS=ON \
10 |  -DLLVM_BUILD_LLVM_DYLIB=ON \
11 |  -DLLVM_DYLIB_COMPONENTS="all" \
12 |  -DLLVM_USE_LINKER="lld" \
13 |  -DLLVM_CCACHE_BUILD=ON

```

You may change install directory by changing ‘-DCMAKE_INSTALL_PREFIX’, and compiler used by ‘-DCMAKE_C_COMPILER’ and ‘-DCMAKE_CXX_COMPILER’ options.

- Build and install

```
1 | ninja
2 | ninja install
```

- Configure library path

```
1 | echo "/tmp/llvm/lib" | sudo tee /etc/ld.so.conf.d/llvm.conf
2 | sudo ldconfig
```

2 Printing CFG

Workflow

1. Generate (emit) LLVM IR from the source.
2. Use ‘dot-cfg’ pass via ‘opt’.
3. Convert generated dot file into image (png).

2.1 Emit LLVM IR

```
1 | clang -S -emit-llvm -O0 -Xclang -disable-optnone \
2 | -fno-discard-value-names <source> -o <out.ll>
```

- -Xclang -disable-optnone :
Tells compiler to not generate ‘optnone’ attributes. ‘optnone’ attribute prohibits opt to access and modify the given function or operation.
- -fno-discard-value-names :
Make compiler to use value names written in the source code.
- The default file extension for LLVM IR is ‘.ll’.

2.2 Call dot-cfg pass

We will use ‘opt’ to call pass over the generated LLVM IR.

```
1 | opt -passes=dot-cfg <input.ll> -disable-output
```

dot files starting with ‘.(dot)’ will be generated for each function in the IR.

Note that those files are hidden in default (because their name starts with ‘.(dot)’!). Please use ‘ls’ with ‘-a’ option to see those files are in place.

2.3 Convert ‘dot’ into image

```
1 | sudo apt install graphviz
2 | dot -Tpng <dot_file> -o <output.png>
```

3 Submission

- Submit your CFG as ‘PR0_<YOUR_STUDENT_ID>.png’ (e.g., ‘PR0_2024-12345.png’) at eTL.

4 Example

```
#include <stdio.h>
```

```
int main() {
    const int x = 12;
    const int y = 20;

    int i, j;
    for (j = 0; j < y; ++j) {
        for (i = 0; i < x; ++i) {
            if (i > j)
                continue;
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

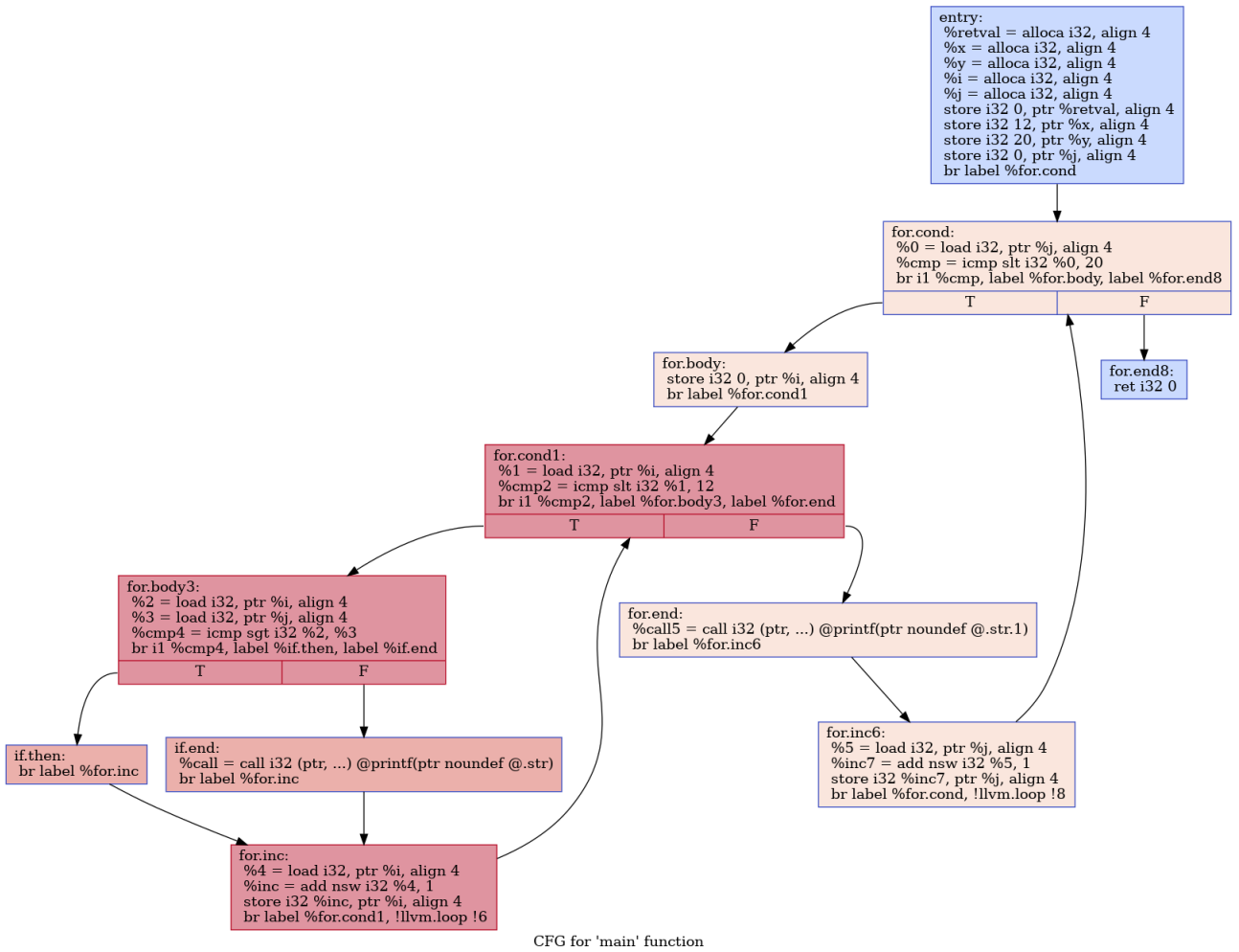


Figure 1: CFG example