Project 1: Warmup (LLVM)

I. Date & Objective: Record the date of each entry & Define the goal or objective of my development work on this particular date

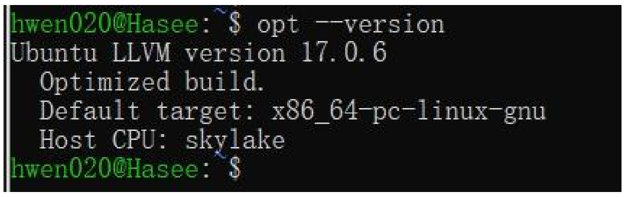
- Jan 18 Set up environment
- Jan 19 Perform experiments with a small test program which I write myself
- Jan 20-21 Report and Video Demonstration

II. Tasks Completed with Code Snippets: List the tasks I worked on during this session.

1. Install LLVM

```
wget https://apt.llvm.org/llvm.sh
chmod +x llvm.sh
sudo ./llvm.sh all
export PATH=$PATH:/lib/llvm-17/bin
echo "export PATH=\$PATH:/lib/llvm-17/bin/" >> ~/.profile
opt --version
```

Result:



```
hwen020@Hasee:~$ opt --version
Ubuntu LLVM version 17.0.6
  Optimized build.
  Default target: x86_64-pc-linux-gnu
  Host CPU: skylake
hwen020@Hasee:~$
```

2. Write test.c

```
nano test.c
```

Insert:

```
// test.c
#include <stdio.h>

int add(int a, int b) {
	return a + b;
}

int main() {
	int x = 5;
	int y = 10;
	int result = add(x, y);

	printf("The sum of %d and %d is: %d\n", x, y, result);

	return 0;
}
```

Press “ctrl+0+enter” to save the test.c file.

Press ”ctrl+X” to exit.

Press “Enter” to confirm file name.

3. Learn the basic uses of the following commands: clang,opt,llc,lli,llvm-link,llvm-as,llvm-dis

clang

Purpose: The Clang compiler is used to compile C, C++, and Objective-C source code into LLVM Intermediate Representation (IR) or directly into machine code.

Usage Example: clang source.c -o output_binary

opt

Purpose: The LLVM optimizer (opt) applies various optimization passes to LLVM IR code.

Usage Example: opt -O2 input.ll -o output.ll (Applies optimization level 2 to input LLVM IR).

llc

Purpose: The LLVM static compiler (llc) translates LLVM IR into machine-specific assembly code.

Usage Example: llc input.ll -o output.s (Translates LLVM IR to machine assembly code).

lli

Purpose: The LLVM interpreter (lli) executes LLVM IR directly without the need for compilation.

Usage Example: lli input.ll (Interprets and executes the LLVM IR).

llvm-link

Purpose: The LLVM linker (llvm-link) is used to link multiple LLVM modules into a single module.

Usage Example: llvm-link module1.ll module2.ll -o linked_module.ll (Links two LLVM modules).

llvm-as

Purpose: The LLVM assembler (llvm-as) converts a human-readable LLVM assembly file into LLVM bitcode.

Usage Example: llvm-as input.ll -o output.bc (Assembles LLVM assembly to LLVM bitcode).

llvm-dis

Purpose: The LLVM disassembler (llvm-dis) converts LLVM bitcode into human-readable LLVM assembly.

Usage Example: llvm-dis input.bc -o output.ll (Disassembles LLVM bitcode to LLVM assembly).

4. Translate between different code formats using the above commands.

source (.c) to binary (executable)

```
clang test.c -o test_binary
```

source (.c) to objective (.o)

```
clang -c test.c -o test.o
```

source (.c) to machine assembly (.s)

```
clang -S -mllvm --x86-asm-syntax=intel test.c -o test.s
```

source (.c) to LLVM bitcode (.bc)

```
clang -c -emit-llvm test.c -o test.bc
```

source (.c) to LLVM IR (.ll)

```
clang -S -emit-llvm test.c -o test.ll
```

LLVM IR (.ll) to LLVM bitcode (.bc)

```
llvm-as test.ll -o test.bc
```

LLVM bitcode (.bc) to LLVM IR (.ll)

```
llvm-dis test.bc -o test.ll
```

LLVM IR (.ll) to machine assembly (.s)

```
llc test.ll -o test.s
```

interpret the LLVM IR (which directly prints the output without compilation)

```
lli test.ll
```

Result files are included in the folder uploaded to google drive.

5. Generate the CFG(s) of test.c

Install graphviz

```
sudo apt install graphviz
```

Generate CFG

```
opt -passes=dot-cfg -disable-output test.ll
ls -la
dot -Tpng .add.dot -o add.png
```

opt -passes=dot-cfg -disable-output test.ll

opt is an LLVM optimization tool.

-passes=dot-cfg specifies the optimization to generate a Control Flow Graph (CFG).

-disable-output disables the actual output generation, so it won't create any output files.

test.ll is the input LLVM IR file.

The purpose of this command is to perform an optimization that generates a Control Flow Graph (CFG) but without producing any output files, allowing for inspection of the CFG for correctness.

ls -la

ls is a command to list files and directories.

-la option indicates listing all files and directories, including hidden ones, in long format.

This command is used to view detailed information about all files and directories in the current directory, including permissions, owners, sizes, etc.

dot -Tpng .add.dot -o add.png

This command uses the Graphviz tool's dot command to convert a graph description file in DOT format to a PNG image file. Here's a breakdown of the command:

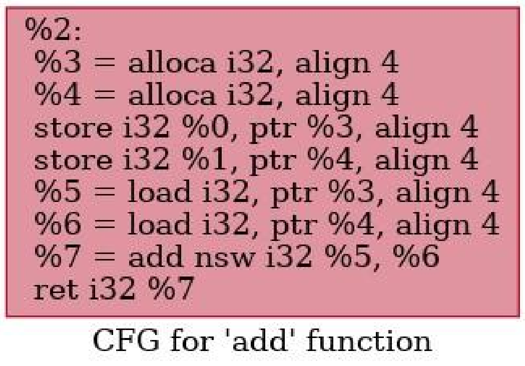
dot is a command-line tool from the Graphviz toolkit used for drawing graphs.

-Tpng: The -Tpng option specifies the output format as a PNG image.

.add.dot is the filename of the input file, which is a graph description file in DOT format.

-o add.png specifies the name of the output file, which is the generated PNG image file.

Therefore, the purpose of this command is to convert a DOT format graph description file named .add.dot into a PNG format image file and save the output as add.png as follow.

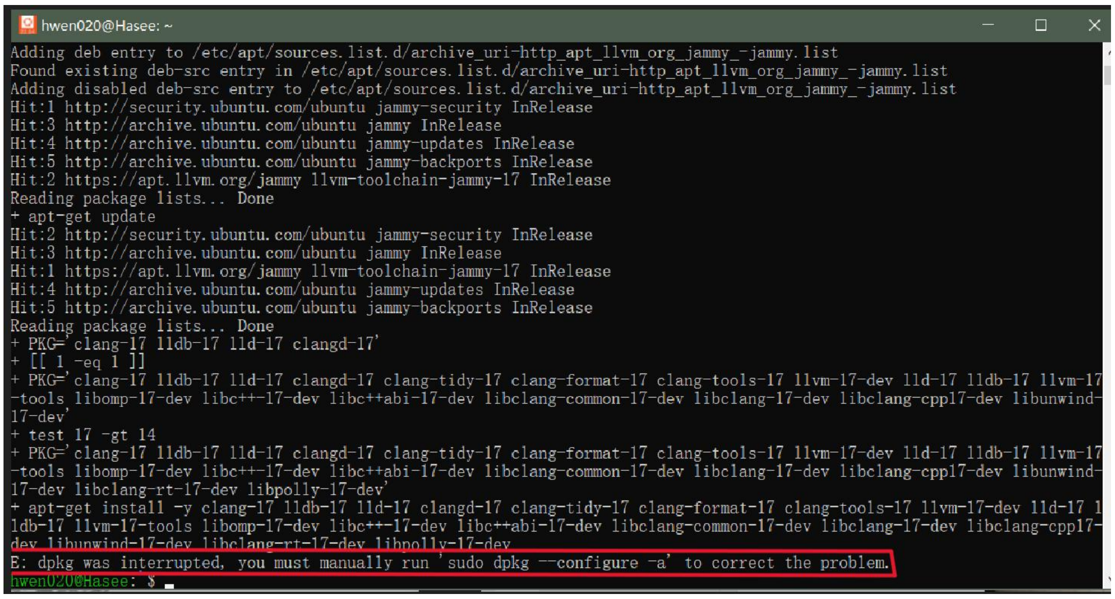


```
%2:
 %3 = alloca i32, align 4
 %4 = alloca i32, align 4
 store i32 %0, ptr %3, align 4
 store i32 %1, ptr %4, align 4
 %5 = load i32, ptr %3, align 4
 %6 = load i32, ptr %4, align 4
 %7 = add nsw i32 %5, %6
 ret i32 %7
```

III. Challenges Faced & Solutions/Workarounds Document any difficulties or challenges encountered and Describe how I addressed the challenges.

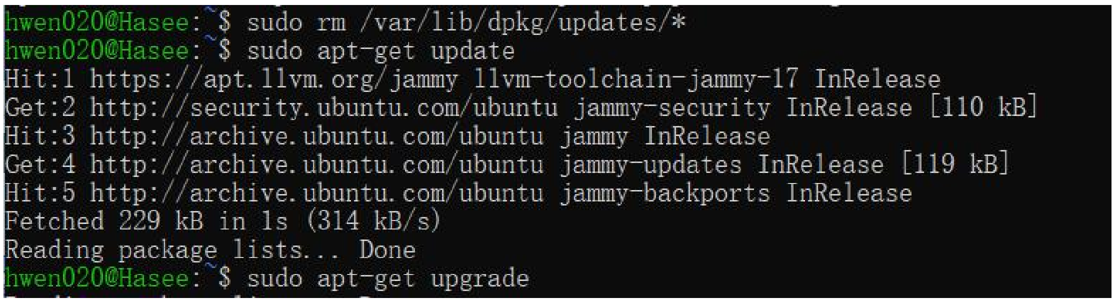
Problem encountered during LLVM installation

I kept receiving the following err as shown in the picture when using the command "sudo ./llvm.sh"



```
hwen020@Hasee: ~
Adding deb entry to /etc/apt/sources.list.d/archive_uri-http_apt_llvm_org_jammy_-jammy.list
Found existing deb-src entry in /etc/apt/sources.list.d/archive_uri-http_apt_llvm_org_jammy_-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-http_apt_llvm_org_jammy_-jammy.list
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:2 https://apt.llvm.org/jammy llvm-toolchain-jammy-17 InRelease
Reading package lists... Done
+ apt-get update
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:1 https://apt.llvm.org/jammy llvm-toolchain-jammy-17 InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
+ PKG='clang-17 lldb-17 lld-17 clangd-17'
+ [[ 1 -eq 1 ]]
+ PKG='clang-17 lldb-17 lld-17 clangd-17 clang-tidy-17 clang-format-17 clang-tools-17 llvm-17-dev lld-17 lldb-17 llvm-17
-tools libomp-17-dev libc++-17-dev libc++abi-17-dev libclang-common-17-dev libclang-17-dev libclang-cpp17-dev libunwind-
17-dev'
+ test 17 -gt 14
+ PKG='clang-17 lldb-17 lld-17 clangd-17 clang-tidy-17 clang-format-17 clang-tools-17 llvm-17-dev lld-17 lldb-17 llvm-17
-tools libomp-17-dev libc++-17-dev libc++abi-17-dev libclang-common-17-dev libclang-17-dev libclang-cpp17-dev libunwind-
17-dev libclang-rt-17-dev libpolly-17-dev'
+ apt-get install -y clang-17 lldb-17 lld-17 clangd-17 clang-tidy-17 clang-format-17 clang-tools-17 llvm-17-dev lld-17 l
ldb-17 llvm-17-tools libomp-17-dev libc++-17-dev libc++abi-17-dev libclang-common-17-dev libclang-17-dev libclang-cpp17-
dev libunwind-17-dev libclang-rt-17-dev libpolly-17-dev
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
hwen020@Hasee: $
```

After searching for reason why this error appears I found that it might be caused by damage of some files in the library. I used the three lines of code in the following picture and the problem was fixed so that I can run the rest commands to install LLVM.



```
hwen020@Hasee:~$ sudo rm /var/lib/dpkg/updates/*
hwen020@Hasee:~$ sudo apt-get update
Hit:1 https://apt.llvm.org/jammy llvm-toolchain-jammy-17 InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 229 kB in 1s (314 kB/s)
Reading package lists... Done
hwen020@Hasee:~$ sudo apt-get upgrade
```

IV. Learnings: Reflect on what I learned during this development process.

Command Line Tools Proficiency

Learning to use a variety of command line tools (clang, opt, llc, lli, llvm-link, llvm-as, llvm-dis, dot) enhances proficiency in the development and analysis of programs at a lower level.

Compilation and Translation Process

Understanding the compilation and translation process from source code to different intermediate and final formats (binary, object files, LLVM bitcode, LLVM IR, machine assembly) is fundamental. This knowledge provides insights into how code is transformed at each step.

LLVM Toolchain Workflow

The development process involves using the LLVM toolchain in a step-by-step manner, starting with compiling C code to binary and progressing to working with LLVM intermediate representations. It demonstrates the workflow for analyzing and optimizing code.

Intermediate Representation Understanding

Working with LLVM Bitcode and LLVM IR allows for a deeper understanding of intermediate representations. This knowledge is valuable for developers who want to optimize and analyze code at a higher level.

Control Flow Graph Generation:
Generating and visualizing Control Flow Graphs (CFGs) provides insights into the program's structure and how different parts interact. This is a powerful technique for program analysis.
Troubleshooting and Debugging:
Encountering and resolving errors during the development process helps in improving troubleshooting and debugging skills. Identifying and fixing issues with command-line tools and file formats enhances problem-solving abilities.
Documentation Exploration:
The need to refer to documentation (e.g., LLVM documentation, command-line help) to understand and address issues emphasizes the importance of exploring and understanding relevant documentation for development tools.