

实验报告

一、实验目的

本实验的目的是了解工业界常用的编译器 GCC 和 LLVM, 熟悉编译器的安装 和使用过程, 观察编译器工作过程中生成的中间文件的格式和内容, 了解编译器的优化效果, 为编译器的学习和构造奠定基础。

二、实验内容

本实验主要的内容为在 Linux 平台上安装和运行工业界常用的编译器 GCC 和 LLVM, 如果系统中没有安装, 则需要首先安装编译器, 安装完成后编写简单的测试程序, 使用编译器编译, 并观察中间输出结果。

三、实验步骤

1、编译器安装

本实验在 Linux 系统下进行。通过使用相应的命令, 使用 Linux 系统的程序管理器, 即可快速完成对实验环境的搭建。使用 Visual Studio code 作为实验代码的编辑器。通过使用相关的命令, 使用 apt-get 包管理器完成实验所需的 gcc、LLVM、clang 环境安装。相关环境的安装命令如下

1. gcc 环境 : `sudo apt-get install build-essential`
2. LLVM 环境: `sudo apt-get install llvm`
3. clang 环境: `sudo apt-get install clang`

在执行上述语句后, 即可完成对实验环境的搭建。通过相关安装完成提示, 可以查看安装的编译环境版本。

2、编写测试程序

以实验 1 中, 对数组排序的代码作为本次的主要实验代码。由于实验中要求完成对单文件、多文件的代码进行编写, 因此在实验 1 的基础上, 将 qsort 函数抽取至 qsort.h、qsort.c 文件中, 作为编译多个文件的实验代码。对于查看编译器中间处理结果的测试代码, 为了便于观察, 使用简单的”Hello, world“程序作为测试程序。这里分别针对 gcc、llvm 编译器将测试程序命名为 hello.c、test.c。对于性能测试方面, 使用运行时间较长的程序进行。同样选取实验 1 中, 对数组排序的代码作为实验代码。

3、实验操作

对于 gcc 编译器, 按照如下步骤完成实验操作:

1. 查看编译器的版本: `gcc --version`
2. 使用编译器编译单个文件: `gcc main.c -o main`

3. 使用编译器编译链接多个文件: `gcc qsort.c main.c -o main`
4. 查看预处理结果: `gcc -E hello.c -o hello.i`
5. 查看语法分析树: `gcc -fdump-tree-all hello.c`
6. 查看中间代码生成结果: `gcc -fdump-rtl-all hello.c`
7. 查看生成的目标代码(汇编代码): `gcc -S hello.c -o hello.s`

对于 LLVM 编译器,按如下步骤完成:

1. 查看编译器的版本: `clang --version`、`llc -version`
2. 使用编译器编译单个文件: `clang main.c -o main`
3. 使用编译器编译链接多个文件: `clang qsort.c main.c -o main`
4. 查看编译流程和阶段: `clang -ccc-print-phases test.c -c`
5. 查看词法分析结果: `clang test.c -Xclang -dump-tokens`
6. 查看词法分析结果 2: `clang test.c -Xclang -dump-raw-tokens`
7. 查看语义分析结果: `clang test.c -Xclang -ast-dump`
8. 查看语义分析结果 2: `clang test.c -Xclang -ast-view`
9. 查看编译优化的结果: `clang test.c -S -mllvm -print-after-all`
10. 查看生成的目标代码结果: Target code generation: `clang test.c -S`

在上述实验进行过程中,对编译器的运行结果进行保存、截图,并在每一步结束后对执行结果进行分析检查。

使用不同的编译参数,对两个编译器输入程序进行优化编译。对每一个程序,重复运行 5 次并计时。

对于 gcc 编译器的四个优化编译指令:

0. `gcc main.c -O0 -o gcc_0`
1. `gcc main.c -O1 -o gcc_1`
2. `gcc main.c -O2 -o gcc_2`
3. `gcc main.c -O3 -o gcc_3`

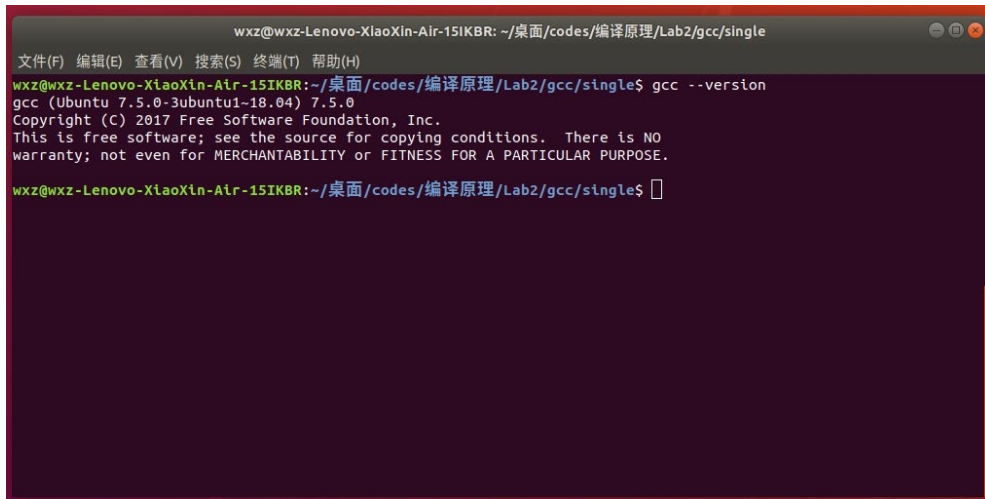
对于 LLVM 编译器的四个优化编译指令:

0. `clang main.c -O0 -o llvm_0`
1. `clang main.c -O1 -o llvm_1`
2. `clang main.c -O2 -o llvm_2`
3. `clang main.c -O3 -o llvm_3`

通过对比 GCC、LLVM 在不同优化等级下的程序运行时间,评估程序运行效率。

四、GCC 运行结果分析

1、查看编译器版本



```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/gcc/single
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/single$ gcc --version
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/single$
```

图 1 查看 gcc 编译器版本

上图为使用命令查看 gcc 编译器版本的结果。本次实验使用的 gcc 编译器版本为：gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

2、使用编译器编译单个文件

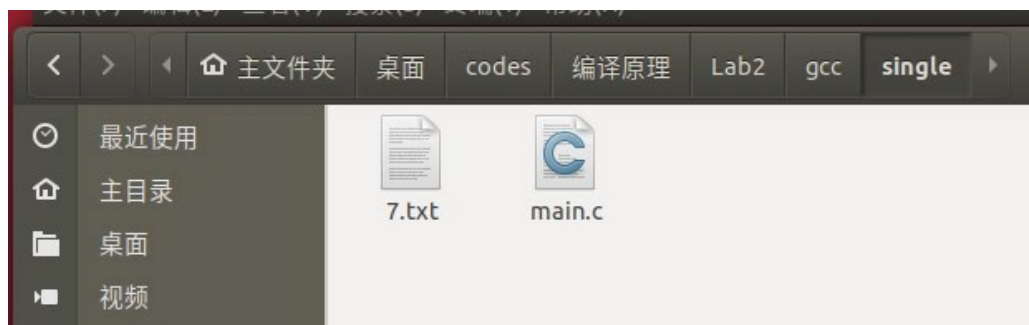
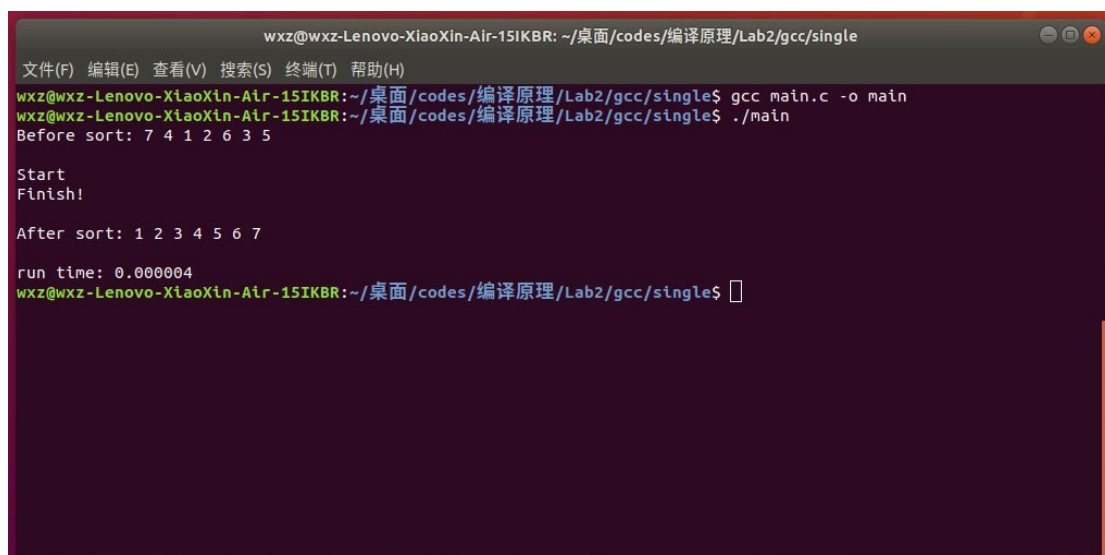


图 2 gcc 单个文件文件夹



```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/gcc/single
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/single$ gcc main.c -o main
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/single$ ./main
Before sort: 7 4 1 2 6 3 5

Start
Finish!

After sort: 1 2 3 4 5 6 7

run time: 0.000004
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/single$
```

图 3 gcc 编译单个文件

如图，使用 gcc 编译器编译程序后，使用 ./main 运行程序。程序正常运行，并输出了排序后的数组。程序运行正常，输出符合预期。

3、使用编译器编译多个文件

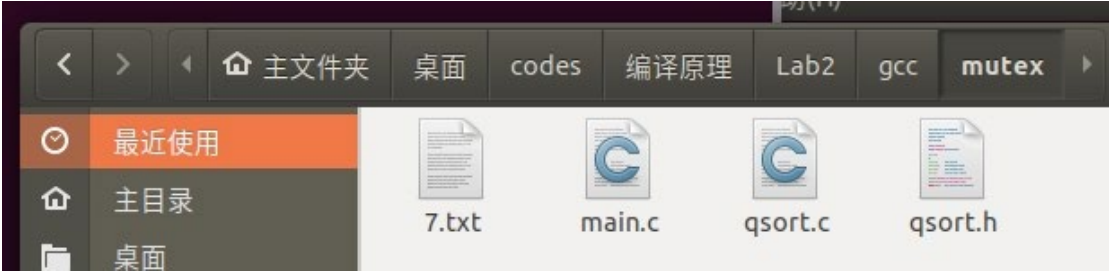


图 4 gcc 多个文件文件夹

```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/gcc/mutex
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/mutex$ gcc main.c qsort.c -o main
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/mutex$ ./main
Before sort: 7 4 1 2 6 3 5

Start
Finish!

After sort: 1 2 3 4 5 6 7

run time: 0.000005
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc/mutex$
```

图 5 gcc 编译多个文件

多个文件的程序结构如图所示，包含 main.c、qsort.h、qsort.c 三个文件。7.txt 为排序用数据文件。如图，使用 gcc 编译器编译多个文件后，使用 ./main 运行程序。程序正常运行，并输出了排序后的数组。程序运行正常，输出符合预期。

4、查看预处理结果

```
hello.i x
Lab2 > gcc > C hello.i > ...
506 extern int fileno(FILE *__stream) __attribute__((__nothrow__, __leaf__));
507
508 extern int fileno_unlocked(FILE *__stream) __attribute__((__nothrow__, __leaf__));
509 # 800 "/usr/include/stdio.h" 3 4
510 extern FILE *popen(const char *__command, const char *__modes);
511
512 extern int pclose(FILE *__stream);
513
514 extern char *ctermid(char *__s) __attribute__((__nothrow__, __leaf__));
515 # 840 "/usr/include/stdio.h" 3 4
516 extern void flockfile(FILE *__stream) __attribute__((__nothrow__, __leaf__));
517
518 extern int ftrylockfile(FILE *__stream) __attribute__((__nothrow__, __leaf__));
519
520 extern void funlockfile(FILE *__stream) __attribute__((__nothrow__, __leaf__));
521 # 868 "/usr/include/stdio.h" 3 4
522
523 # 2 "hello.c" 2
524
525 # 3 "hello.c"
526 int main( ) {
527     printf("Hello,world!\n");
528 }
529
```

图 6 gcc 预处理结果

执行命令: `gcc -E hello.c -o hello.i` 后, 编译器将预处理后的文件存储至 `hello.i` 中。经与源文件对比, 源 `hello.c` 文件中, `#include<stdio.h>` 的头文件被替换为一大段代码, 而主程序部分被完整的保留。

5、查看语法分析树

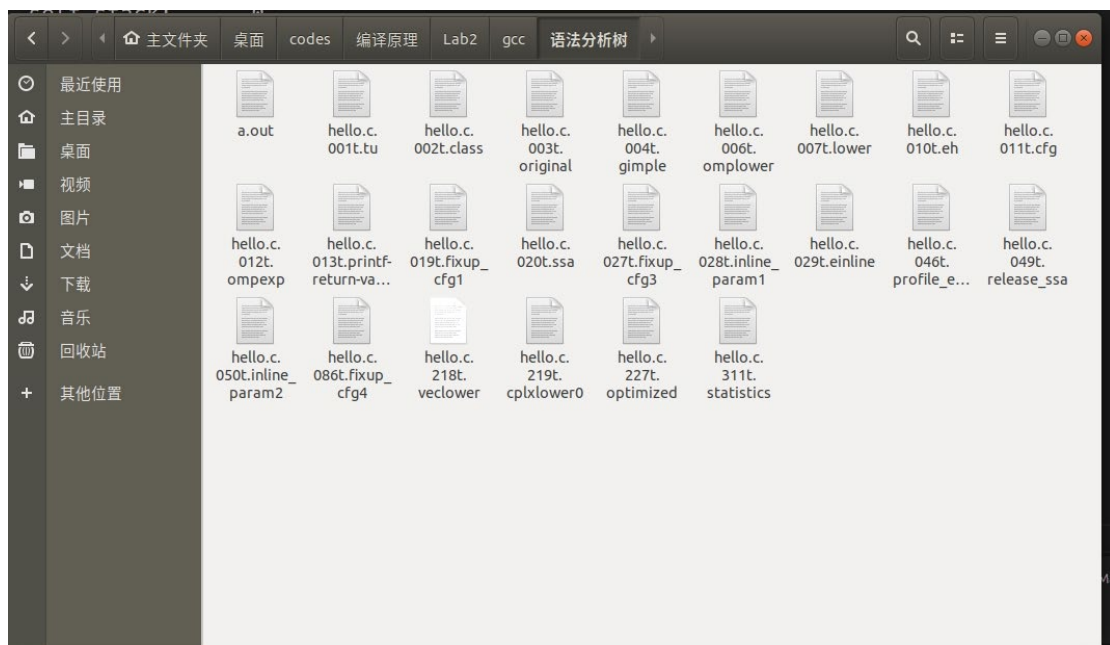


图 7 gcc 语法分析树

执行命令：`gcc -fdump-tree-all hello.c` 后，编译器将语法分析结果输出至文件夹。这里包含 24 个文件，其中 23 个为语法分析树文件，1 个为编译后的可执行文件。

6、查看中间代码生成结果

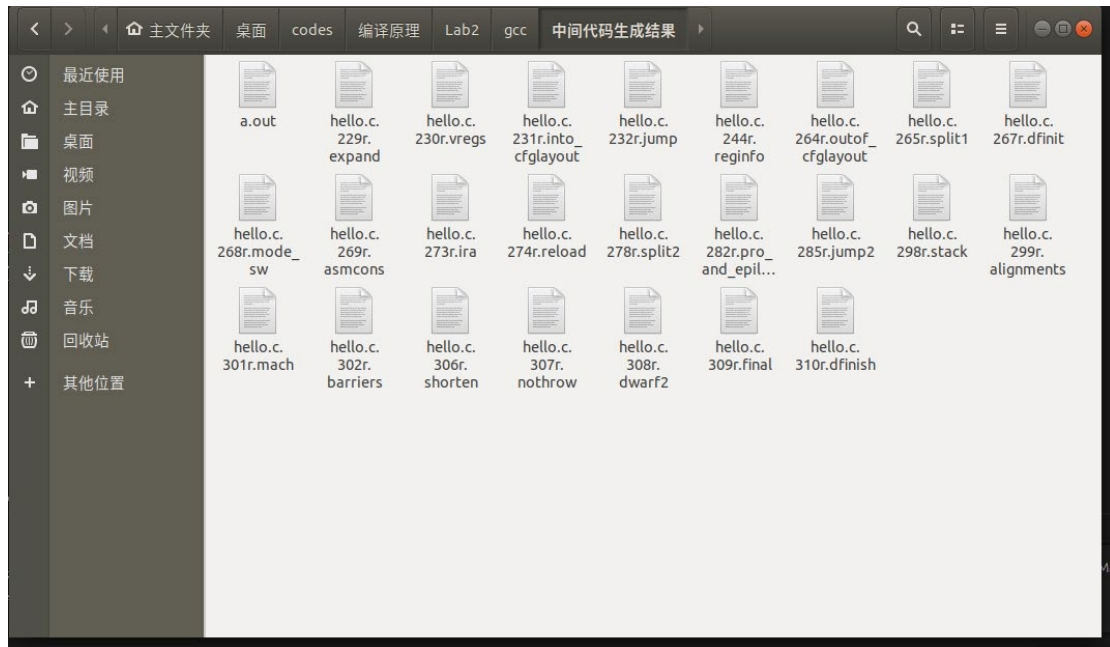


图 8 gcc 中间代码生成结果

执行命令：`gcc -fdump-rtl-all hello.c` 后，编译器将所有中间代码输出至文件夹。这里包含 25 个文件，其中 24 个为中间代码文件，1 个为编译后的可执行文件。

7、查看生成的目标代码（汇编代码）

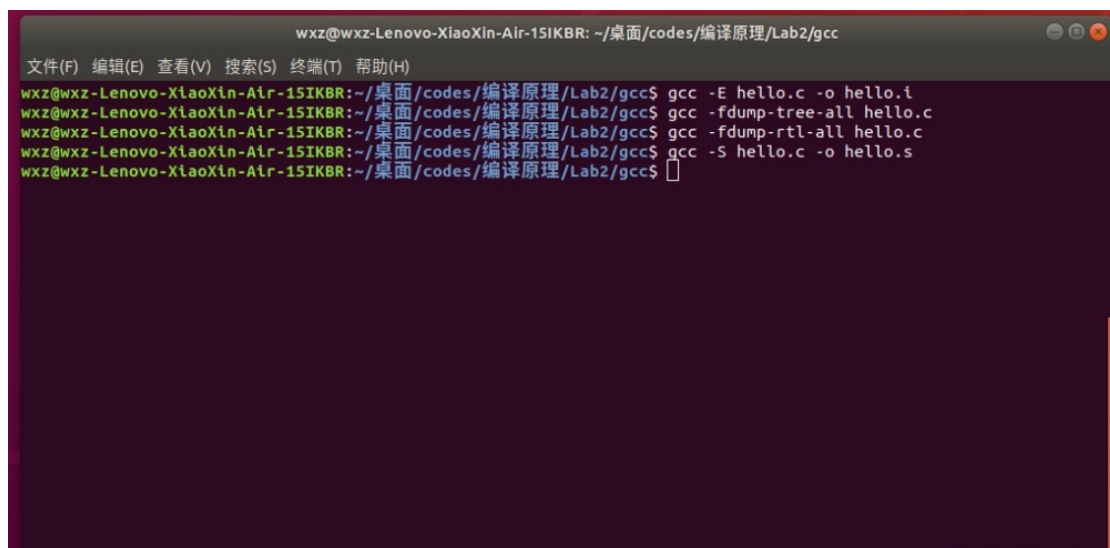
```

hello.s x
Lab2 > gcc > hello.s
1      .file    "hello.c"
2      .text
3      .section .rodata
4      .LC0:
5      .string "Hello,world!"
6      .text
7      .globl  main
8      .type   main, @function
9      main:
10     .LFB0:
11     .cfi_startproc
12     pushq   %rbp
13     .cfi_def_cfa_offset 16
14     .cfi_offset 6, -16
15     movq    %rsp, %rbp
16     .cfi_def_cfa_register 6
17     leaq    .LC0(%rip), %rdi
18     call    puts@PLT
19     movl    $0, %eax
20     popq    %rbp
21     .cfi_def_cfa 7, 8
22     ret
23     .cfi_endproc
24     .LFE0:
25     .size   main, .-main
26     .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"
27     .section .note.GNU-stack,"",@progbits
28

```

图9 gcc 目标代码

执行命令：`gcc -S hello.c -o hello.s` 后，编译器将最终生成的汇编代码输出至文件夹。



```

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/gcc
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc$ gcc -E hello.c -o hello.i
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc$ gcc -fdump-tree-all hello.c
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc$ gcc -fdump-rtl-all hello.c
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc$ gcc -S hello.c -o hello.s
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/gcc$

```

图10 gcc 查看编译中间结果

五、LLVM 运行结果分析

1、查看编译器版本

A terminal window titled 'wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm/single'. The terminal shows the output of the 'clang --version' and 'llc -version' commands. The output for clang is 'clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)' with target 'x86_64-pc-linux-gnu', thread model 'posix', and installed dir '/usr/bin'. The output for llc is 'LLVM (http://llvm.org/): LLVM version 6.0.0', 'Optimized build.', 'Default target: x86_64-pc-linux-gnu', 'Host CPU: skylake', and a list of registered targets including aarch64, aarch64_be, amdgc, arm, arm64, armeb, bpf, bpfel, bpfel, and hexagon.

```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm/single
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/single$ clang --version
clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/single$ llc -version
LLVM (http://llvm.org/):
  LLVM version 6.0.0

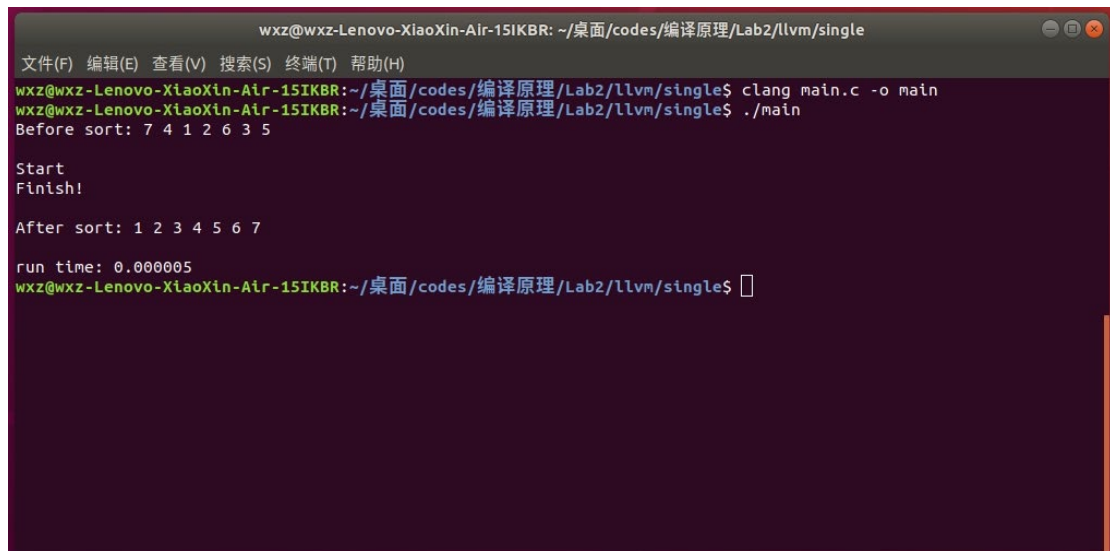
  Optimized build.
  Default target: x86_64-pc-linux-gnu
  Host CPU: skylake

  Registered Targets:
    aarch64      - AArch64 (little endian)
    aarch64_be   - AArch64 (big endian)
    amdgc        - AMD GCN GPUs
    arm          - ARM
    arm64        - ARM64 (little endian)
    armeb        - ARM (big endian)
    bpf          - BPF (host endian)
    bpfel        - BPF (big endian)
    bpfel        - BPF (little endian)
    hexagon      - Hexagon
```

图 11 llvm 查看编译器版本

上图为使用命令查看编译器版本的结果。本次实验使用的 LLVM 编译环境中，clang 版本：clang version 6.0.0-1ubuntu2，LLVM 版本：LLVM version 6.0.0

2、使用编译器编译单个文件

A terminal window titled 'wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm/single'. The terminal shows the output of the 'clang main.c -o main' and './main' commands. The output for clang is 'clang main.c -o main'. The output for ./main is 'Before sort: 7 4 1 2 6 3 5', 'Start', 'Finish!', 'After sort: 1 2 3 4 5 6 7', and 'run time: 0.000005'.

```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm/single
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/single$ clang main.c -o main
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/single$ ./main
Before sort: 7 4 1 2 6 3 5

Start
Finish!

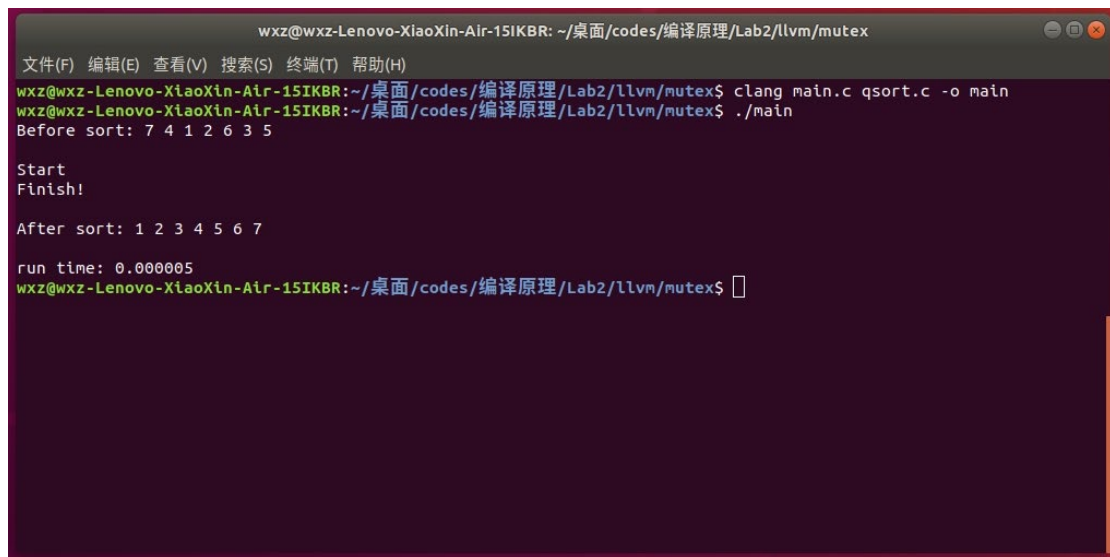
After sort: 1 2 3 4 5 6 7

run time: 0.000005
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/single$
```

图 12 llvm 编译单个文件

使用命令：clang main.c -o main 可以完成对单个文件的编译。从上图可以看到，程序正常运行，输出结果符合预期。

3、使用编译器编译链接多个文件



```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm/mutex
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/mutex$ clang main.c qsort.c -o main
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/mutex$ ./main
Before sort: 7 4 1 2 6 3 5

Start
Finish!

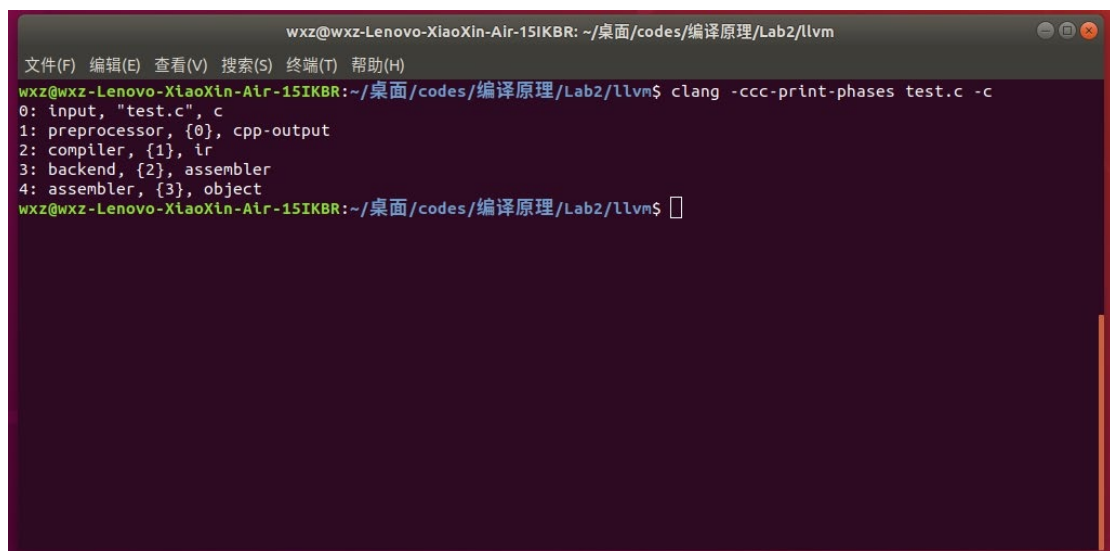
After sort: 1 2 3 4 5 6 7

run time: 0.000005
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm/mutex$
```

图 13 llvm 编译多个文件

多个文件的程序结构如图所示，包含 main.c、qsort.h、qsort.c 三个文件。7.txt 为排序用数据文件。如图，使用 llvm 编译器编译多个文件后，使用 ./main 运行程序。程序正常运行，并输出了排序后的数组。程序运行正常，输出符合预期。

4、查看编译流程和阶段



```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang -ccc-print-phases test.c -c
0: input, "test.c", c
1: preprocessor, {0}, cpp-output
2: compiler, {1}, ir
3: backend, {2}, assembler
4: assembler, {3}, object
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$
```

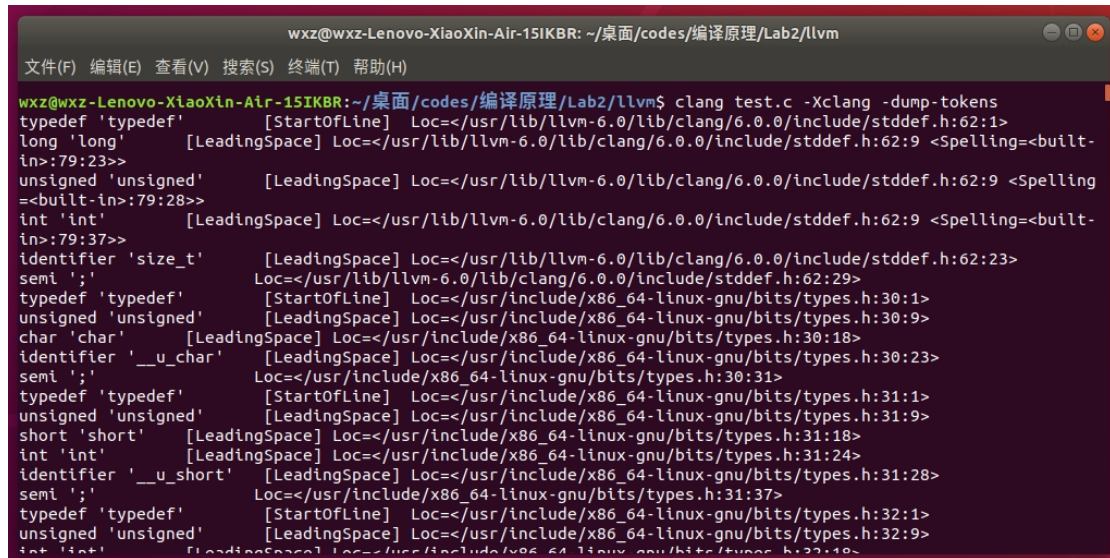
图 13 llvm 查看编译流程和阶段

执行命令：clang -ccc-print-phases test.c -c 后，在终端窗口输出程序编译执行的各个阶段，分别为：

1. 读取文件

2. 预处理器
3. 编译程序
4. 后端
5. 汇编程序

5、查看词法分析结果



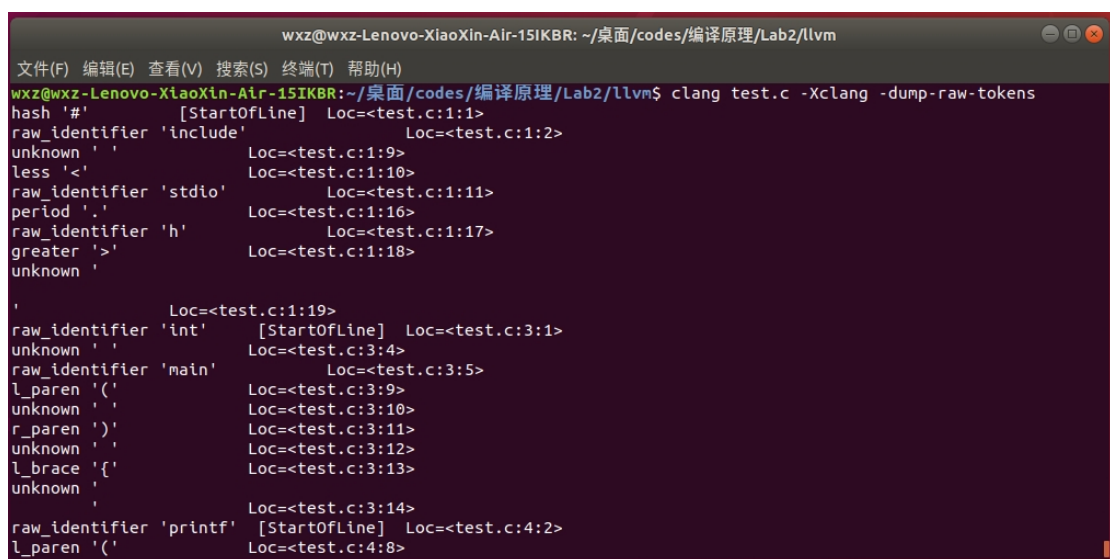
```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang test.c -Xclang -dump-tokens
typedef 'typedef' [StartOfLine] Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:1>
long 'long' [LeadingSpace] Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:9 <Spelling=<built-
in>;79:23>
unsigned 'unsigned' [LeadingSpace] Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:9 <Spelling
=<built-in>;79:28>
int 'int' [LeadingSpace] Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:9 <Spelling=<built-
in>;79:37>
identifier 'size_t' [LeadingSpace] Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:23>
semi ';' Loc=</usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:29>
typedef 'typedef' [StartOfLine] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:30:1>
unsigned 'unsigned' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:30:9>
char 'char' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:30:18>
identifier '__u_char' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:30:23>
semi ';' Loc=</usr/include/x86_64-linux-gnu/bits/types.h:30:31>
typedef 'typedef' [StartOfLine] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:1>
unsigned 'unsigned' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:9>
short 'short' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:18>
int 'int' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:24>
identifier '__u_short' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:28>
semi ';' Loc=</usr/include/x86_64-linux-gnu/bits/types.h:31:37>
typedef 'typedef' [StartOfLine] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:32:1>
unsigned 'unsigned' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:32:9>
int 'int' [LeadingSpace] Loc=</usr/include/x86_64-linux-gnu/bits/types.h:32:18>
```

图 14 llvm 查看词法分析结果

执行命令: `clang test.c -Xclang -dump-tokens` 后, 在终端界面显示对程序执行此法分析后的结果。

6、查看词法分析结果 2



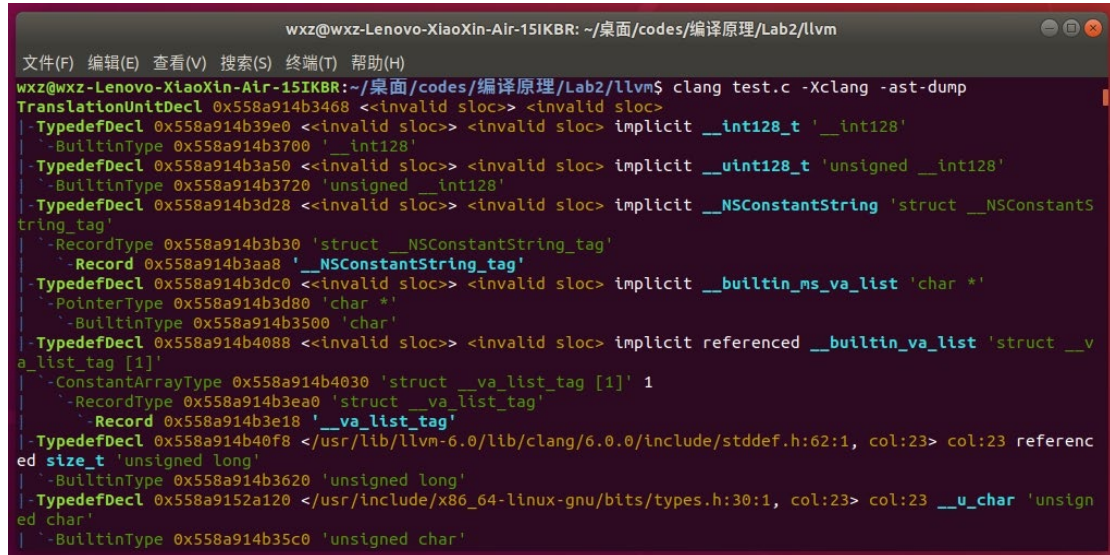
```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang test.c -Xclang -dump-raw-tokens
hash '#' [StartOfLine] Loc=<test.c:1:1>
raw_identifier 'include' Loc=<test.c:1:2>
unknown ' ' Loc=<test.c:1:9>
less '<' Loc=<test.c:1:10>
raw_identifier 'stdio' Loc=<test.c:1:11>
period '.' Loc=<test.c:1:16>
raw_identifier 'h' Loc=<test.c:1:17>
greater '>' Loc=<test.c:1:18>
unknown ' ' Loc=<test.c:1:19>
raw_identifier 'int' [StartOfLine] Loc=<test.c:3:1>
unknown ' ' Loc=<test.c:3:4>
raw_identifier 'main' Loc=<test.c:3:5>
l_paren '(' Loc=<test.c:3:9>
unknown ' ' Loc=<test.c:3:10>
r_paren ')' Loc=<test.c:3:11>
unknown ' ' Loc=<test.c:3:12>
l_brace '{' Loc=<test.c:3:13>
unknown ' ' Loc=<test.c:3:14>
raw_identifier 'printf' [StartOfLine] Loc=<test.c:4:2>
l_paren '(' Loc=<test.c:4:8>
```

图 15 llvm 查看词法分析结果 2

执行命令：clang test.c -Xclang -dump-raw-tokens 后，在终端界面显示对程序执行词法分析后的结果。此部分的词法分析可以清晰看到，针对 test.c 源文件中的每一个语法成分做了明确的属性标注和位置信息标注。每个语法成份的属性、所属文件、行号、列号均做了清晰标注。

7、查看语义分析结果

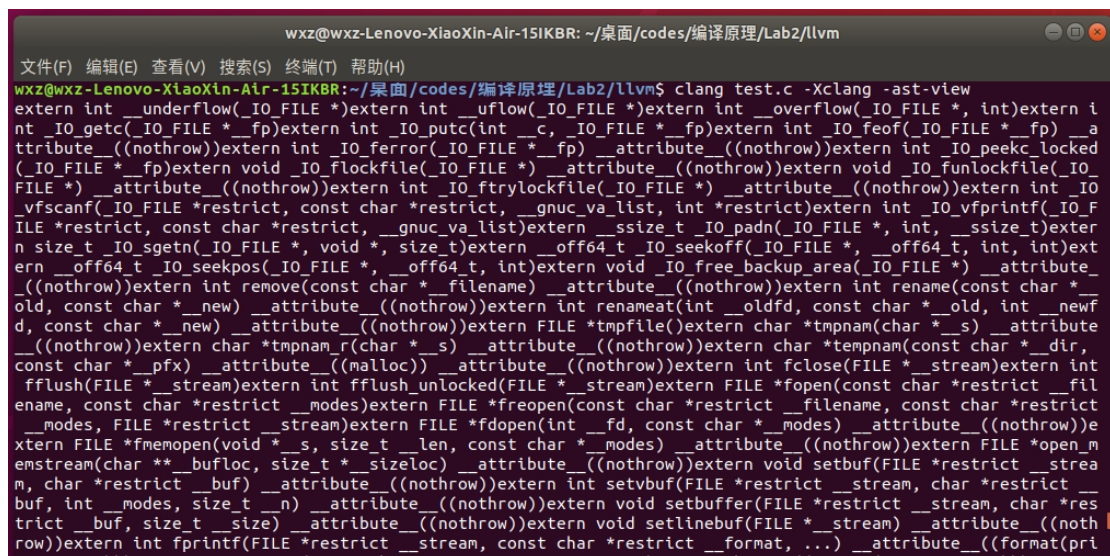


```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang test.c -Xclang -ast-dump
TranslationUnitDecl 0x558a914b3468 <<invalid sloc>> <invalid sloc>
| -TypeDefDecl 0x558a914b39e0 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
| | -BuiltinType 0x558a914b3700 '__int128'
| -TypeDefDecl 0x558a914b3a50 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
| | -BuiltinType 0x558a914b3720 'unsigned __int128'
| -TypeDefDecl 0x558a914b3d28 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_tag'
| | -RecordType 0x558a914b3b30 'struct __NSConstantString_tag'
| | | -Record 0x558a914b3aa8 '__NSConstantString_tag'
| -TypeDefDecl 0x558a914b3dc0 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
| | -PointerType 0x558a914b3d80 'char *'
| | -BuiltinType 0x558a914b3500 'char'
| -TypeDefDecl 0x558a914b4088 <<invalid sloc>> <invalid sloc> implicit referenced __builtin_va_list 'struct __va_list_tag [1]'
| | -ConstantArrayType 0x558a914b4030 'struct __va_list_tag [1]' 1
| | | -RecordType 0x558a914b3ea0 'struct __va_list_tag'
| | | | -Record 0x558a914b3e18 '__va_list_tag'
| -TypeDefDecl 0x558a914b40f8 </usr/lib/llvm-6.0/lib/clang/6.0.0/include/stddef.h:62:1, col:23> col:23 referenced size_t 'unsigned long'
| | -BuiltinType 0x558a914b3620 'unsigned long'
| -TypeDefDecl 0x558a9152a120 </usr/include/x86_64-linux-gnu/bits/types.h:30:1, col:23> col:23 __u_char 'unsigned char'
| | -BuiltinType 0x558a914b35c0 'unsigned char'
```

图 16 llvm 查看语义分析结果

执行命令：clang test.c -Xclang -ast-view 后，编译器在终端输出对程序执行语义分析的结果。

8、查看语义分析结果 2

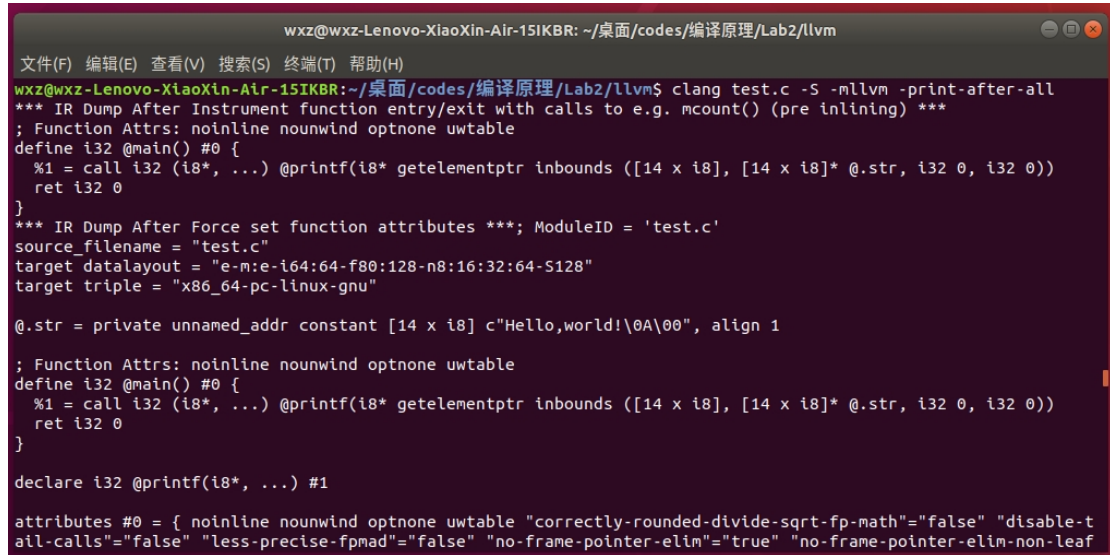


```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang test.c -Xclang -ast-view
extern int __underflow(_IO_FILE *)extern int __uflow(_IO_FILE *)extern int __overflow(_IO_FILE *, int)extern int __getc(_IO_FILE * __fp)extern int __putc(int __c, _IO_FILE * __fp)extern int __feof(_IO_FILE * __fp) __attribute__((nothrow))extern int __ferror(_IO_FILE * __fp) __attribute__((nothrow))extern int __peekc_locked(_IO_FILE * __fp)extern void __flockfile(_IO_FILE *) __attribute__((nothrow))extern void __funlockfile(_IO_FILE *) __attribute__((nothrow))extern int __ftrylockfile(_IO_FILE *) __attribute__((nothrow))extern int __vscanf(_IO_FILE * __restrict, const char * __restrict, __gnuc_va_list, int * __restrict)extern int __vfprintf(_IO_FILE * __restrict, const char * __restrict, __gnuc_va_list)extern __ssize_t __IO_pdn(_IO_FILE *, int, __ssize_t)extern size_t __IO_sgetn(_IO_FILE *, void *, size_t)extern __off64_t __IO_seekoff(_IO_FILE *, __off64_t, int, int)extern __off64_t __IO_seekpos(_IO_FILE *, __off64_t, int)extern void __IO_free_backup_area(_IO_FILE *) __attribute__((nothrow))extern int remove(const char * __filename) __attribute__((nothrow))extern int rename(const char * __old, const char * __new) __attribute__((nothrow))extern int renameat(int __oldfd, const char * __old, int __newfd, const char * __new) __attribute__((nothrow))extern FILE * __tmpfile()extern char * __tmpnam(char * __s) __attribute__((nothrow))extern char * __tmpnam_r(char * __s) __attribute__((nothrow))extern char * __tmpnam(const char * __dir, const char * __pfx) __attribute__((malloc)) __attribute__((nothrow))extern int fclose(FILE * __stream)extern int fflush(FILE * __stream)extern int fflush_unlocked(FILE * __stream)extern FILE * __fopen(const char * __restrict __filename, const char * __restrict __modes)extern FILE * __freopen(const char * __restrict __filename, const char * __restrict __modes, FILE * __stream)extern FILE * __fdopen(int __fd, const char * __modes) __attribute__((nothrow))extern FILE * __fmemopen(void * __s, size_t __len, const char * __modes) __attribute__((nothrow))extern FILE * __open_memstream(char ** __bufloc, size_t * __sizeloc) __attribute__((nothrow))extern void setbuf(FILE * __restrict __stream, char * __restrict __buf) __attribute__((nothrow))extern int setvbuf(FILE * __restrict __stream, char * __restrict __buf, int __modes, size_t __n) __attribute__((nothrow))extern void setbuffer(FILE * __restrict __stream, char * __restrict __buf, size_t __size) __attribute__((nothrow))extern void setlinebuf(FILE * __stream) __attribute__((nothrow))extern int fprintf(FILE * __restrict __stream, const char * __restrict __format, ...) __attribute__((format(pri
```

图 17 llvm 查看语义分析结果 2

执行命令：clang test.c -Xclang -ast-view 后，编译器在终端输出对程序执行语义分析的结果。

9、查看编译优化的结果



```
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/llvm
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/llvm$ clang test.c -S -mllvm -print-after-all
*** IR Dump After Instrument function entry/exit with calls to e.g. mcount() (pre inlining) ***
; Function Attrs: noline nounwind optnone uwtable
define i32 @main() #0 {
    %1 = call i32 @i32 (i8*, ...) @printf(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @.str, i32 0, i32 0))
    ret i32 0
}
*** IR Dump After Force set function attributes ***; ModuleID = 'test.c'
source_filename = "test.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [14 x i8] c"Hello,world!\0A\00", align 1

; Function Attrs: noline nounwind optnone uwtable
define i32 @main() #0 {
    %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @.str, i32 0, i32 0))
    ret i32 0
}

declare i32 @printf(i8*, ...) #1

attributes #0 = { noline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
```

图 18 llvm 查看编译优化的结果

执行命令：clang test.c -S mllvm -print-after-all 后，编译器输出代码优化后的结果。

10、查看生成的目标代码结果


```

test.s x
编译原理 > Lab2 > llvm > test.s
1  .text
2  .file "test.c"
3  .globl main # -- Begin function main
4  .p2align 4, 0x90
5  .type main,@function
6  ~ main: # @main
7  .cfi_startproc
8  ~ # %bb.0:
9  pushq %rbp
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 subq $16, %rsp
15 movabsq $.L.str, %rdi
16 movb $0, %al
17 callq printf
18 xorl %ecx, %ecx
19 movl %eax, -4(%rbp) # 4-byte Spill
20 movl %ecx, %eax
21 addq $16, %rsp
22 popq %rbp
23 retq
24 ~ .Lfunc_end0:
25 .size main, .Lfunc_end0-main
26 ~ .cfi_endproc
27 # -- End function
28 .type .L.str,@object # @.str
29 .section .rodata.str1.1,"aMS",@progbits,1
30 ~ .L.str:
31 .asciz "Hello,world!\n"
32 .size .L.str, 14
33

```

图 19 llvm 查看生成的目标代码结果

执行命令：clang test.c -S 后，编译器会将生成的汇编代码保存至当前文件夹，并命名为 test.s。

六、GCC 与 LLVM 对比分析

1、特点对比

从使用上来看，gcc 与 LLVM 并没有明显的区别。通过使用简单的命令，即可完成对单文件、多文件程序的编译，指定相应优化等级、输出目录等详细编译参数。

在编译器工作方式方面，gcc 编译器在预处理之后，执行语法树分析，并形成大量语法分析树文件。在之后，执行中间代码生成，并进行多项代码优化操作。因此，在此分布也会形成大量的中间代码。最后，执行代码生成，形成最终的汇编代码。

而对于 LLVM 编译器，在预处理后，编译器先后进行词法分析、语法分析、语义分析和中间代码生成。这部分工作由 LLVM 的前端 clang 进行完成。之后由 LLVM 后端执行代码优化、生成目标代码等操作，并形成最终的目标汇编代码。

2、性能对比

```

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/speed
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ gcc main.c -O0 -o gcc_0
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ gcc main.c -O1 -o gcc_1
main.c: In function 'main':
main.c:35:2: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &num);
main.c:37:3: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &a[i]);
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ gcc main.c -O2 -o gcc_2
main.c: In function 'main':
main.c:35:2: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &num);
main.c:37:3: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &a[i]);
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ gcc main.c -O3 -o gcc_3
main.c: In function 'main':
main.c:35:2: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &num);
main.c:37:3: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
fscanf(fp, "%d", &a[i]);
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$

```

图 20 gcc 不同优化等级编译

上图为使用 gcc 编译器，进行不同优化等级优化编译的程序编译命令。

经测试，对 5000000 规模的数据进行排序，在-O0、-O1、-O2、-O3 四个优化等级下，排序时间分别为：3.215、1.470、1.181、1.299。显然，优化-O2 是执行速度最快的优化等级，它的执行速度甚至快于-O3。

表1 gcc编译器不同优化等级程序运行时间

优化等级	1	2	3	4	5	平均
-O0	3.203999	3.180167	3.191086	3.223592	3.273963	3.215
-O1	1.45504	1.4775	1.458391	1.496569	1.463941	1.470
-O2	1.166163	1.20466	1.187566	1.177173	1.169733	1.181
-O3	1.293478	1.3135	1.292953	1.307134	1.288223	1.299

```

wxz@wxz-Lenovo-XiaoXin-Air-15IKBR: ~/桌面/codes/编译原理/Lab2/speed
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ clang main.c -O0 -o llvm_0
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ clang main.c -O1 -o llvm_1
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ clang main.c -O2 -o llvm_2
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$ clang main.c -O3 -o llvm_3
wxz@wxz-Lenovo-XiaoXin-Air-15IKBR:~/桌面/codes/编译原理/Lab2/speed$

```

图 22 LLVM 不同优化等级编译

上图为使用 LLVM 编译器，进行不同等级优化编译的程序编译命令。

经测试，对 5000000 规模的数据进行排序，在 -O0、-O1、-O2、-O3 四个优化等级下，排序时间分别为：3.043、1.083、1.089、1.089。优化 -O1 是执行速度最快的优化等级，它的执行速度也比 -O3 快。

表2 LLVM编译器不同优化等级程序运行时间

优化等级	1	2	3	4	5	平均
-O0	3.099347	3.034016	3.031169	3.025571	3.023625	3.043
-O1	1.080175	1.084706	1.080149	1.08499	1.086492	1.083
-O2	1.089277	1.089641	1.087473	1.089208	1.086991	1.089
-O3	1.084862	1.094768	1.097186	1.081378	1.086087	1.089

从上面的对比可以看出，gcc、LLVM 在 -O0 的最低优化等级下的性能存在一定的差距。而在高优化等级下，gcc 的程序运行最短时间约 1.181s，LLVM 的程序运行最短时间约为 1.083s 左右。这说明，gcc、LLVM 在性能方面存在一定的差距。LLVM 编译器生成的代码性能会稍优于 gcc 生成的代码。

同时在对最终生成的汇编语言进行对比后发现，高优先等级下，生成的代码行数远多于低优先级代码。这可能导致一些过度的优化，从而使得在某些情况下程序性能的下降。在两款编译器中，均存在高优化等级下生成的程序性能并非最快的程序这一显现，在 gcc 中这一现象尤为明显。

七、心得体会

本次实验中，我使用 gcc、LLVM 两种 C 语言编译环境，对于单文件、多文件的 c 语言程序进行了编译，并对两种编译器对单文件程序的编译过程作了细致分析。可以看到，两种编译器在程序编译方面有更多的差异，在编译过程及编译器架构上均有不同。gcc 是一款较为完善的标准编译器，在工业界有较为广泛的应用。而 LLVM 则是一种编译器的框架，对于不同的语言，使用不同的前端生成相同的中间代码；而对于不同的及其平台，则使用不同的后端，生成运行在不同平台上运行的代码。在本次编译中使用的 clang，就是 LLVM 的前端。在性能方面，经查阅相关资料，LLVM 的性能、安全性等都较 gcc 有一定优势。在具体其它细节方面，也有较多的差距。

经过本次实验，我对 gcc、LLVM 两种编译器有了更深入的了解。对于编译器的工作过程和中间文件有了更多的了解和认识。对这两款编译器的架构和特点也有了进一步的了解。在以后的学习中，我要进一步了解和學習以上两款经典编译器的特点和原理，为编译器的学习和构造奠定基础。