Lab 02 Answer Sheet

1. GCC

完成以下操作，每一个操作限用一条指令，且不能更改文件位置：

- 将 `./main.c` 编译为 `./main.o`（仅编译）

  > 指令：gcc -c -o main.o main.c -I include

- 将 `./v1/dog.c` 编译为 `./v1/dog.o`（仅编译）

  > 指令：gcc -c -o v1/dog.o v1/dog.c -I include

- 将 `./v2/dog.c` 编译为 `./v2/dog.o`（仅编译）

  > 指令：gcc -c -o v2/dog.o v2/dog.c -I include

- 将 `./v1/dog.o` 与 `./main.o` 链接为 `./dog1`

  > 指令：gcc -o dog1 v1/dog.o main.o

- 将 `./v2/dog.o` 与 `./main.o` 链接为 `./dog2`

  > 指令：gcc -o dog2 v2/dog.o main.o

- 执行 `./dog1`

  > 指令：dog1
  >
  > 输出结果：I'm a dog, my name is P-A-I-M-O-N.

- 执行 `./dog2`

  > 指令：dog2
  >
  > 输出结果：I'm a really really beautiful dog, my holy name is P-A-I-M-O-N.

2. 静态和动态库的操作

静态库

- 执行下面的命令

  ```
  ar cr libdog.a ./v1/dog.o
  gcc -o main main.o -L. -ldog
  ```

  说明上述两个命令完成了什么事?

> 1）创建静态库
>
> 2）链接静态库生成可执行文件

- 查看 `main` 文件大小，并记录

> 16752 B

- 执行 `./main`
- 执行以下命令：

```
ar cr libdog.a ./v2/dog.o
./main
```

和上面执行的结果有不同之处吗？

> 没有。

动态库

- 执行下面的命令

```
gcc -c -fPIC v1/dog.c -o v1/dog.o -I include
gcc -c -fPIC v2/dog.c -o v2/dog.o -I include
gcc -shared -fPIC -o libdog.so v1/dog.o
gcc main.c libdog.so -o main -I include
```

- 查看 `main` 文件大小，并记录

> 16672 B

- 执行 `./main` 前，需要将库文件路径设置为当前路径：

```
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH
```

- 执行 `./main`
- 执行以下命令：

```
gcc -shared -fPIC -o libdog.so v2/dog.o
./main
```

和上面执行的结果有不同之处吗？

> 有，输出了 v2/dog.c 中实现的 bark。

- 和静态库操作结果进行对比，同时分析 `main` 文件的大小，你能得出什么结论？

> 静态库会把库文件中的函数放入可执行文件中，而动态库不会。静态库在链接时装载，动态库在运行时装载。

3. GDB

（2）在 main 函数处设置断点（写出命令）

```
b main
```

（7）在程序第 10 行加入断点（写出命令）

```
b 10
```

（10）使用 print 命令打印 i 和 factorial 的值（写出命令）

```
print i
print factorial
```

说明源程序中存在的错误。

> 没有初始化 factorial 的值。

你更喜欢使用 printf 还是 gdb 调试？为什么？

> 具有可视化图形界面的操作系统上喜欢 printf，因为操作方便，门槛低。
>
> 不具有可视化图形界面的操作系统上喜欢 gdb，因为能方便地获得各个变量的值。

4. make

`./Makefile` 文件内容：

```
main: main.o fun1.o fun2.o
        gcc -o main main.o fun1.o fun2.o -Llib -ldy

fun1.o: src/fun1.c include/fun1.h
        gcc -c -o fun1.o src/fun1.c -I include

fun2.o: src/fun2.c include/fun2.h
        gcc -c -o fun2.o src/fun2.c -I include

main.o: src/main.c include/dylib.h include/fun1.h include/fun2.h
        gcc -c -o main.o src/main.c -I include

clean:
        rm main main.o fun1.o fun2.o
```

`main` 可执行程序输出了什么？

```
hello world
this is fun1
this is fun2
HIDE AND SEEK, FIND YOU NEXT WEEK!
```