

课程实践性作业准备练习

首先解压缩实验用的文件：

```
unzip lab0.zip
```

本实验帮助你学习 linux 环境下的编译和调试工具。

知识点包括

- gcc
- gdb
- make

练习 1 gcc

学习简单的编译指令，并复习 C 语言中的宏定义。

修改 glory.c 文件中的四个宏（V0~V3）的值，使用以下命令编译和运行程序：

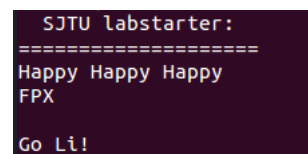
编译

```
$ gcc -o glory glory.c
```

运行

```
$ ./glory
```

使得程序输出为：



```
SJTU labstarter:
=====
Happy Happy Happy
FPX
Go Li!
```

练习 2 GDB

GDB 调试器是 GCC 编译器的兄弟。做为自由软件领域几乎是唯一的调试器，它秉承了 Unix 类操作系统的一贯风格，采用纯命令行操作，有点儿类似 dos 下的 debug。

关于它的使用方法请看 GDB 使用手册。这个网址

<http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html> 可以帮助你理解 GDB.

另外，可以学习实验楼的 [《GDB 简明教程》](https://www.shiyanlou.com/courses/496)，通过动手实验学习 Linux 上 GDB 调试 C 语言程序的基本技巧。

使用以下命令可以将可执行文件中用于调试的信息保存下来，供 gdb 使用：

编译

```
$ gcc -g -o hello hello.c
```

调试

```
$ gdb hello
```

逐步调试程序：

1. setting a breakpoint at main
2. giving gdb's run command
3. using gdb's single-step command

在实验报告中回答以下问题:

1. How do you pass command line arguments to a program when using gdb?
2. How do you set a breakpoint which only occurs when a set of conditions is true (e.g. when certain variables are a certain value)?
3. How do you execute the next line of C code in the program after stopping at a breakpoint?
4. If the next line of code is a function call, you'll execute the whole function call at once if you use your answer to #3. How do you tell GDB that you want to debug the code inside the function instead?
5. How do you resume the program after stopping at a breakpoint?
6. How can you see the value of a variable (or even an expression like $1+2$) in gdb?
7. How do you configure gdb so it prints the value of a variable after every step?
8. How do you print a list of all variables and their values in the current function?
9. How do you exit out of gdb?

练习 3 调试

编译并执行 ll_equal.c

编译

\$ gcc -g -o ll_equal ll_equal.c

执行

\$./ll_equal

运行结果如下:

```
equal test 1 result = 1
段错误 (核心已转储)
```

使用 gdb 在 ll_equal() 函数中设置断点, 用 gdb 调试, 找出程序中的错误, 并修正过来。
在实验报告中把正确的函数实现写出来。

练习 4 Make 初步

本实验的目的:

- Make 初步
- 掌握 main 函数中 argc, argv 参数的用法。
- 文件的基本操作

编译

\$ make wc

再试试:

\$./wc wc.c

\$ wc wc.c

这两次输出的内容为何不同：(Hint: 运行 `which wc`)

修改 `wc.c`, so that it implements word count according to the specification of ``man wc``, except that it does not need to support any flags and only needs to support a single input file, (or STDIN if none is specified). Beware that `wc` in OS X behaves differently from `wc` in Ubuntu. We will expect you to follow the behavior of `wc` in Ubuntu.

功能：修改 `wc.c`, 使其具有类似于 `wc` in Ubuntu 的功能。统计一个指定文件的行数、单词数、字符数。该程序不需要支持 `wc` in Ubuntu 中的 option 选项，如 `--bytes`, `--chars` 等选项。在不指定文件名时，统计标准输入（ubuntu 中 `ctrl-d` 表示 EOF）的行数、单词数、字符数。

```
void wc(FILE *ofile, FILE *infile, char *iname) {  
}
```

在实验报告中把你的 `wc.c` 文件中的关键模块：`wc` 函数写出来。

`wc` 函数的三个参数分别代表: 1: 要统计的文件, 缺省时是标准输入, 2: 结果输出到的文件, 缺省时是标准输出。3: 一个字符串, 代表要统计的文件名。

具体代码框架请查看 `main()` 函数。

`make` 是一个工具程序 (Utility software), 经由读取一个名字为“`Makefile`”的文件, 自动化建构软件。它构建的目标被称为“`target`”; 与此同时, 它也检查文件的依赖关系, 如果需要的话, 它还会调用一些外部软件来完成任务。它的依赖关系检查规则非常简单, 主要根据依赖文件的修改时间进行判断。

对于大型、复杂的 `project`, 多个程序文件之间存在依赖和调用关系, 程序员很难通过一条简单的 `gcc` 编译指令就能够生成整个 `project` 的可执行代码。我们往往会通过写 `Makefile` 来来确定 `target` 文件的依赖关系, 并把生成这个 `target` 的相关命令传给 `shell` 去执行。

有了 `Makefile`, 就可以通过相关命令 `make`, 编译源代码, 生成结果代码, 然后把结果代码连接起来生成可执行文件或者库文件。

这个博客是一个简单的 `Makefile` 例子, 帮你起步
(<http://blog.chinaunix.net/uid-25838286-id-3204219.html>)

这里有一个非常棒的 `Makefile` 教程: (<http://wiki.wlug.org.nz/MakefileHowto>).

官方手册在这儿: (<http://www.gnu.org/software/make/manual/make.html>).

B 站上的视频, 也能帮你了解它的来龙去脉:

<https://www.bilibili.com/video/BV1B4411F7EK?from=search&seid=7439875720245299496>