计算机系统结构 Lab01

练习1 gcc

使用编辑器打开并修改glory.c

```
nano glory.c
```

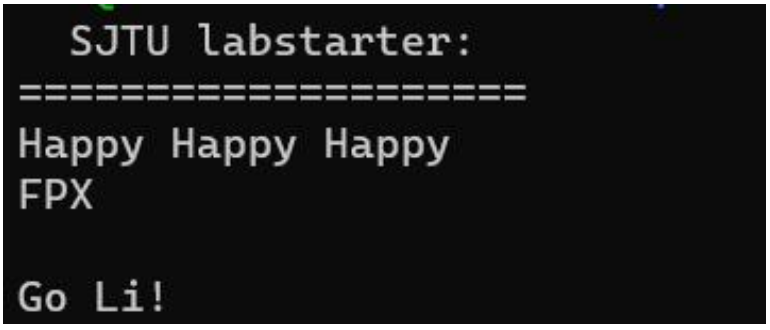
将宏定义修改为

```
#define V0 3
#define V1 3
#define V2 1
#define V3 3
```

保存修改，编译并运行

```
gcc glory.c -o glory
./glory
```

输出结果：



练习2 gdb

逐步调试程序，输出结果如下：

```
Reading symbols from hello...
(gdb) break main
Breakpoint 1 at 0x1169: file hello.c, line 8.
(gdb) run
Starting program: /home/mcx/code/lab0/hello

Breakpoint 1, main (argc=21845, argv=0x7ffff7fb2fc8 <__exit_funcs_lock>) at hello.c:8
8       int main(int argc, char** argv) {
(gdb) step
10              char *str = "hello, world!", ch;
(gdb) next
11              for (i = 0; i < strlen(str); i++)
```

1. **How do you pass command line arguments to a program when using gdb?**
   By using command `run arg1 "arg2" ...` , for exmple:

   ```
   (gdb) run hello world
   ```

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)?**
   By using command `break line-or-function if expr` , for example:

   ```
   (gdb) break 11 if a==1
   (gdb) break main if i==1
   ```

3. **How do you execute the next line of C code in the program after stopping at a breakpoint?**
   By using command `next` .
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?**
   By using command `step` .
5. **How do you resume the program after stopping at a breakpoint?**
   By using command `continue` .
6. **How can you see the value of a variable (or even an expression like 1+2) in gdb?**
   By using command `print [[OPTION]... --] [/FMT] [EXP]` , for example:

   ```
   (gdb)print a
   (gdb)print a+5
   ```

7. **How do you configure gdb so it prints the value of a variable after every step?**
   First, set a breakpoint. After the breakpoint being hit, use command `display[/FMT] EXP` , for example:

   ```
   (gdb)display a
   ```

8. **How do you print a list of all variables and their values in the current function?**
   By using command `info locals` and `info args`.
9. **How do you exit out of gdb?**
   By using command `quit`.

练习3 调试

编译完成后，进入gdb，设置断点并开始执行程序。

```
(gdb) break ll_equal
(gdb) run
```

触发断点之后，使用 `continue` 继续执行，当第二次进入 `ll_equal` 函数之后报错。
查看报错处的变量与参数。

```
(gdb) info locals
(gbd) info agrs
```

结果如下：

```
Program received signal SIGSEGV, Segmentation fault.
0x0000555555555185 in ll_equal (a=0x7fffffffe220, b=0x0) at ll_equal.c:11
11                      if (a->val != b->val)
(gdb) info locals
No locals.
(gdb) info args
a = 0x7fffffffe220
b = 0x0
```

可见第11行对空指针 `b` 进行了数据访问，因而造成非法内存操作。应在访问指针前先判断指针是否为空，修改后代码如下：

```
1  int ll_equal(const node* a, const node* b) {
2          while (a != NULL && b != NULL) {
3                  if (a->val != b->val)
4                          return 0;
5                  a = a->next;
6                  b = b->next;
7          }
8          /* lists are equal if a and b are both null */
9          return a == b;
10 }
```

练习4 Make初步

编译并执行，运行结果如下：

```
mcx@mcx-virtual-machine:~/code/lab0$ make wc
cc      wc.c    -o wc
mcx@mcx-virtual-machine:~/code/lab0$ ./wc wc.c
mcx@mcx-virtual-machine:~/code/lab0$ wc wc.c
  9  23 145 wc.c
```

两次运行结果不同的原因是 `./wc` 中的 `./` 代指当前目录，这条命令执行的程序为当前目录下的 `wc` 文件；而直接使用 `wc` 会执行位于系统path目录下的 `wc` 程序，执行 `which wc` 可以发现 `wc` 指代 `/usr/bin/wc` .
修改 `wc.c` 中 `wc` 函数，代码如下：

```
void wc(FILE *ofile, FILE *infile, char *inname) {
    char ch;
    int cCount = 0,wCount = 0,lCount = 0;
    int in_word = 0;
        if(!ofile) ofile = stdout;
        if(!infile) infile = stdin;
        if(!inname) inname = "";
    while(!feof(infile)){
        ch = fgetc(infile);
        if(ch!=EOF) cCount++;
        if(ch>=33&&ch<=126){
            if(!in_word){
                in_word = 1;
                wCount++;
            }
        }else in_word = 0;
        if(ch == '\n') lCount++;
    }
    fprintf(ofile,"%8d%8d%8d\t%s\n",lCount,wCount,cCount,inname);
}

```