# 补充实验：LINUX 下C语言使用、编译与调试实验

**一、实验目的：**

1. 练习并掌握Linux提供的vi编辑器来编译C程序
2. 学会利用gcc、gdb编译、调试C程序
3. 学会使用make工具

**二、实验内容**

1. 编写C语言程序，用gcc编译并观察编译后的结果，运行生成的可执行文件。
2. 利用gdb调试程序。
3. 学习编写makefile，并进行编译。

**三、实验步骤**

1. **文件编辑器vi**

进入vi，直接执行vi编辑程序。

例：# vi test.c

显示器出现vi的编辑窗口，同时vi会将文件复制一份至缓冲区（buffer）。vi先对缓冲区的文件进行编辑，保留在磁盘中的文件则不变。编辑完成后，使用者可决定是否要取代原来旧有的文件。

1) vi的工作模式

（1）输入模式

输入以下命令即可进入vi输入模式：

|  |  |
| --- | --- |
| a | 在光标之后加入资料 |
| A | 在该行之末加入资料 |
| i | 在光标之前加入资料 |
| I | 在该行之首加入资料 |
| o | 新增一行于该行之下 |
| O | 新增一行于该行之上 |

（2）命令模式

在输入模式下，按ESC可切换到命令模式。命令模式下，可选用下列指令离开vi：

|  |  |
| --- | --- |
| ：q! | 离开vi，并放弃刚在缓冲区内编辑的内容 |
| ：wq | 将缓冲区内的资料写入磁盘中，并离开vi |
| ZZ | 同wq |
| ：x | 同wq |
| ：w | 将缓冲区内的资料写入磁盘中，但并不离开vi |
| ：q | 离开vi，若文件被修改过，则要被要求确认是否放弃修改的内容，此指令可与：w配合使用 |

（3）命令模式下光标的移动

|  |  |
| --- | --- |
| h | 左移一个字符 |
| j | 下移一个字符 |
| k | 上移一个字符 |
| l | 右移一个字符 |
| 0（零） | 移至该行的行首 |
| $ | 移至该行的行尾 |
| ^ | 移至该行的第一个字符处 |
| H | 移至窗口的第一行 |
| M | 移至窗口中间那一行 |
| L | 移至窗口的最后一行 |
| G | 移至该文件的最后一行 |
| W, w | 下一个单词 (W 忽略符号) |
| B, b | 上一个单词 (B 忽略符号) |

(4) 命令模式下的编辑命令

|  |  |
| --- | --- |
| dd | 删除当前光标所在行 |
| yy | 复制当前光标所在行 |
| p | 将复制的内容粘贴在光标所在的位置后 |
| P | 将复制的内容粘贴在光标所在的位置前 |
| x | 删除当前光标字符 |
| X | 删除当前光标之前字符 |
| u | 撤消 |
| · | 重做 |

1. **GNU C编译器**

1) 使用gcc

通常后跟一些选项和文件名来使用gcc编译器。gcc命令的基本用法如下：

gcc [options] [filenames]

命令行选项指定的编译过程中的具体操作

2) gcc常用选项

当不用任何选项编译一个程序时，gcc将建立（假定编译成功）一个名为a.out的可执行文件。

选项含义：

-o FILE 指定输出文件名，在编译为目标代码时，这一选项不是必须的。如果FILE 没

有指定，默认文件名是a.out．

例如，

# gcc test.c

编译成功后，当前目录下就产生了一个a.out文件。

也可用-o选项来为即将产生的可执行文件指定一个文件名来代替a.out。

例如：

#gcc –o count count.c

此时得到的可执行文件就不再是a.out，而是count。

-c GCC 仅把源代码编译为目标代码。默认时GCC 建立的目标代码文件有一个.o 的

扩展名。

-E 对文件进行预处理

-S 对文件进行编译，生成汇编代码。

-O 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行得更快。

-g 在可执行程序中包含标准调试信息。

-Wall 允许发出GCC 能提供的所有有用的警告，也可以用-W(warning)来标识指定的

警告。

-l *name* 链接静态库

-L *dir* 库文件的搜索路径

3) 执行文件

格式： ./可执行文件名

例：

# ./a.out

# ./count

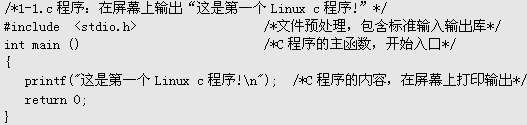
4) 示例程序

1. 设计一个程序，要求在屏幕上输出“这是第一个Linux c程序!”。

步骤 1:设计编辑源程序代码

使用文本编辑器vim，在终端中输入：

[root@localhost root]#vim 1-1.c

输入完成后存盘：按ESC键→输入“:wq”回车 

步骤 2:编译程序

查看当前目录下是否有1-1.c文件，输入命令:

[root@localhost root]#ls

1-1.c

编译：

[root@localhost root]#gcc 1-1.c -o 1-1

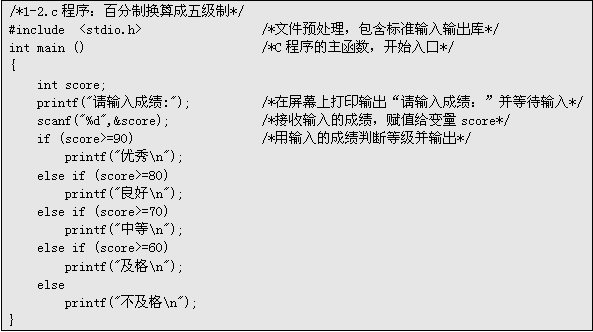
步骤 3:运行程序

[root@localhost root]#**ls**

1-1.c 1-1

[root@localhost root]#**./1-1**

这是第一个Linux c程序!

1. 设计一个程序，要求把输入的百分制的成绩转换成五级制输出。即输入成绩大于等于90分，显示“优秀”；若成绩介于80~90分，显示“良好”；若成绩介于70~80分，显示“中等”；若成绩介于60~70分，显示“及格”；若成绩小于60分，显示“不及格”。

**步骤 1:**编辑源程序代码

[root@localhost root]#**vim 1-2.c**

**步骤 2:**用gcc编译程序

[root@localhost root]#**gcc 1-2.c –o 1-2**

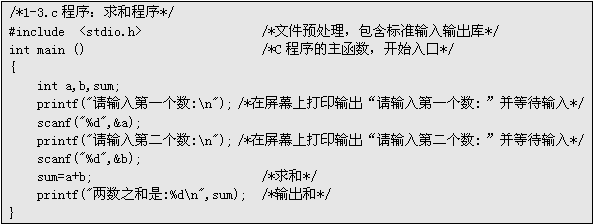
**步骤 3:**运行程序

[root@localhost root]**#./1-2**

1. 设计一个程序，要求输入两个整数，求和输出。通过使用gcc的参数，控制gcc的编译过程，了解gcc的编译过程，进一步认识gcc的灵活性。

**步骤 1:**编辑源程序代码

[root@localhost root]#**vim 1-3.c**



**步骤 2:**预处理阶段

[root@localhost root]#**gcc 1-3.c –o 1-3.i –E**

[root@localhost root]#**vim 1-3.i**

**步骤 3:编译阶段**

[root@localhost root]**#gcc 1-3.i –o 1-3.s –Ｓ**

[root@localhost root]**#vim 1-3.s**

**步骤 4:**汇编阶段

[root@localhost root]#**gcc 1-3.s –o 1-3.o –c**

**步骤 5:**链接阶段

[root@localhost root]#**gcc 1-3.o –o 1-3**

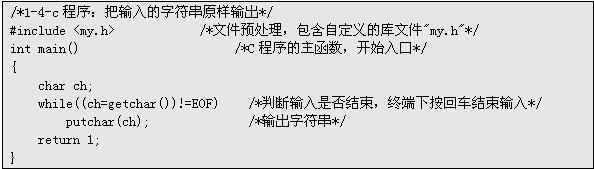
Linux系统把printf和scanf函数的实现，都放在了libc.so.6的库文件中。在没有参数指定时，gcc到系统默认的路径“/usr/lib”下查找，链接到libc.so.6库函数中去，这样就有了printf和scanf函数的实现部分。把程序中一些函数的实现，这是链接阶段的工作。

完成链接后，gcc就可以生成可执行程序文件

1. 设计一个程序，要求把输入的字符串原样输出，程序中的头文件自己定义，源程序文件为“1-4.c”，自定义的头文件为“my.h”，放在目录“/root”下。

**步骤 1:**设计编辑源程序代码1-4.c

[root@localhost root]#**vim 1-4.c**



**步骤 2:**设计编辑自定义的头文件my.h

[root@localhost root]#**vim my.h**

**步骤 3:**正常编译1-4.c文件：

[root@localhost root]#**gcc 1-4.c –o 1-4**

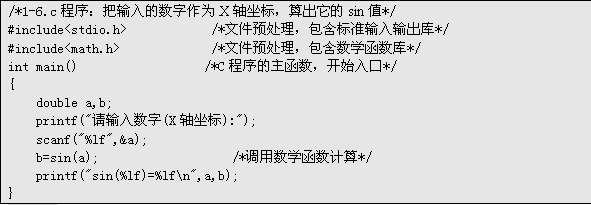
编译器提示出错。

**步骤 4:**加“-I dir”参数编译：

[root@localhost root]#**gcc 1-4.c –o 1-4 –I /root**

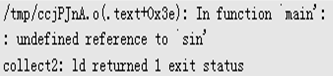
1. 设计一个程序，要求把输入的数字作为X轴坐标，算出它的sin值。

**步骤 1:**编辑源程序代码

[root@localhost root]#**vim 1-6.c**

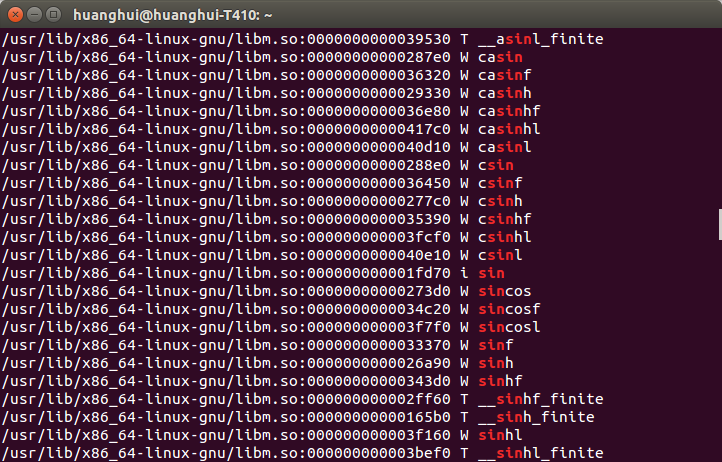
**步骤 2:**用gcc编译程序

[root@localhost root]#**gcc 1-6.c –o 1-6**

结果发现编译器报错：

原因是需要指定函数的具体路径，要查找函数sin，输入：

[root@localhost root]# **nm -Do /usr/lib/x86\_64-linux-gnu/\*.so|grep sin**



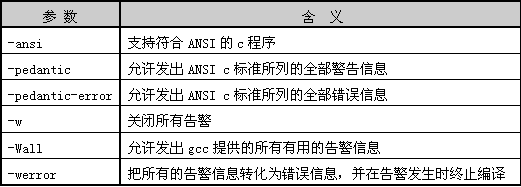
在/lib/libm.so:000000000001fd70 i sin中除去函数库头lib，所余下的符号为“m”，在编译时用字符“l”与余下的符号“m”相连接成“lm”，在编译时加上此参数就能正确地通过编译，即：

[root@localhost root]# **gcc 1-6.c –o 1-6 -lm**

* **步骤 3:**运行程序

[root@localhost root]#**./1-6**

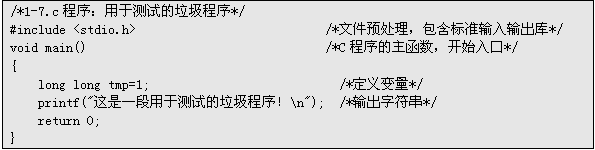
## gcc的常用告警和出错参数



**1-7：** 设计一个程序，要求打印“这是一段用于 测试的垃圾程序！”，里面包含一些非标准语法。

**步骤 1:**设计编辑源程序代码

[root@localhost root]#**vim 1-7.c**



**步骤 2:关闭所有告警**

**[root@localhost root]#gcc 1-7.c –o 1-7 –w**

**步骤 3:显示不符合ANSI c标准语法的告警信息**

**[root@localhost root]#gcc 1-7.c –o 1-7 –ansi**

**步骤 4: 允许发出ANSI c标准所列的全部警告信息**

**[root@localhost root]#gcc 1-7.c –o 1-7 –pedantic**

**步骤 5:允许发出gcc提供的所有有用的告警信息**

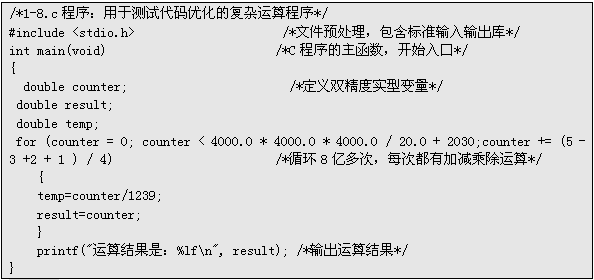
**[root@localhost root]#gcc 1-7.c –o 1-7 –Wall**

## 优化参数

1. 代码优化指的是编译器通过分析源代码，找出其中 尚未达到最优的部分，然后对其重新进行组合，目 的是改善程序的执行性能。
2. gcc提供的代码优化功能非常强大，它通过编译参数 “-On”来控制优化代码的生成，其中n是一个代表 优化级别的整数。
3. 通常来说，数字越大优化的等级越高，同时也就意 味着程序的运行速度越快。

**1-8：**设计一个程序，要求循环8亿次左右，每次都有一些可以优化的加减乘除运算。比较gcc的编译参数“-On”优化程序前后的运行速度。

**步骤 1:**编辑源程序代码：



**步骤 2:**不加任何优化参数进行编译。

[root@localhost root]#**gcc 1-8.c –o 1-8**

**步骤 3:用time命令大致统计出该程序在运行时所需要的时间。**

**[root@localhost root]#time ./1-8**

**步骤 4:加“-O2”优化参数进行编译。**

**[root@localhost root]#gcc 1-8.c –o 1-8 –O2**

**步骤 5:再统计优化后的程序运行时所需要的时间。**

**[root@localhost root]#time ./1-8**

**步骤 6:**对比两次执行的输出结果。

优化虽然能够给程序带来更好的执行性能，但在 一些场合中应该避免优化代码。

* 1. 程序开发的时候。
  2. 资源受限的时候。
  3. 跟踪调试的时候。

1. **gdb调试工具**

1) 调试编译代码

为了使gdb正常工作，必须使你的程序在编译时包含调试信息。调试信息里包含你程序里的每个变量的类型和在可执行文件里的地址映射以及源代码的行号。gdb利用这些信息使源代码和机器码相关联。

在编译时用 –g 选项打开调试选项。

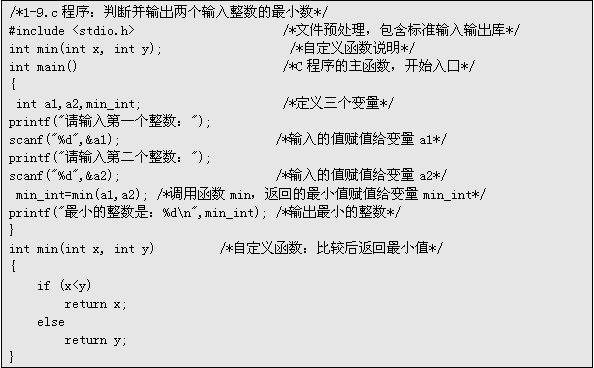
2) gdb基本命令

|  |  |
| --- | --- |
| 命 令 | 描 述 |
| file | 装入欲调试的可执行文件 |
| kill | 终止正在调试的程序 |
| list | 列出产生执行文件的源代码部分 |
| next | 执行一行源代码但不进入函数内部 |
| step | 执行一行源代码并进入函数内部 |
| run | 执行当前被调试的程序 |
| quit | 终止gdb |
| watch | 监视一个变量的值而不管它何时被改变 |
| break | 在代码里设置断点，使程序执行到这里时被挂起 |
| make | 不退出gdb就可以重新产生可执行文件 |
| shell | 不离开gdb就执行UNIX shell 命令 |

3)使用gdb调试示例程序

**设计一个程序，要求输入两个整数，判断并输出其中的最小数。**

**步骤 1:**编辑源程序代码：

**步骤 2:** 用gcc编译程序

[root@localhost root]#**gcc 1-9.c –o 1-9 –g**

**步骤 3:** 进入gdb调试环境

[root@localhost root]#**gdb 1-9**

**步骤 4:** 用gdb调试程序

(1)查看源文件

在gdb中输入“l”(list)就可以查看程序源代码，一次显示10行;

(2)设置断点

在gdb中设置断点命令是“b”(break)，后面跟行号或者函数名。

如：(gdb) b 10

(3)查看断点信息

用命令“info b”(info break)查看断点信息。

(4)运行程序：输入“r”(run)开始运行程序。

(5)查看变量值

程序运行到断点处会自动暂停，输入“p 变量名”

调试程序时，可能需要修改变量值，程序运行到断点处时，输入“set 变量=设定值”，例如给变量“a2” 赋值11，输入“set a2=11”。

gdb在显示变量值时都会在对应值前加“$n”标记，它是当前变量值的引用标记，以后想再引用此变量，可以直接使用“$n”，提高了调试效率 。

(6)单步运行

在断点处输入 “n”(next)或者“s”(step) 。它们之间的区别在于：若有函数调用时，“s”会进入该函数而“n”不会进入该函数。

(7)继续运行程序

输入“c”(continue)命令恢复程序的正常运行，把剩余的程序执行完，并显示执行结果。

(8)退出gdb环境：输入“q”(quit)命令。

1. **make的使用**

（1）用vi编辑以下程序，程序清单：

main.c

function1.h

function1.c

function2.h

function2.c

//main.c

#include "function1.h"

#include "function2.h"

int main(int argc, char \*\*argv)

{

function1\_print("hello");

function2\_print("world");

return 0;

}

//function1.h

void function1\_print(char \*str);

//function1.c

#include "function1.h"

void function1\_print(char \*str)

{

printf("This is function1 print %s\n", str);

}

//function2.h

void function2\_print(char \*str);

//function2.c

#include "function2.h"

void function2\_print(char \*str)

{

printf("This is function2 print %s\n", str);

}

实验要求：

（1）画出各个源程序、目标文件以及最终的目标文件之间的依赖关系图。

（2）编辑makefile文件

（3）利用make命令进行上述程序的编译，生成可执行代码并运行。

（4）修改其中一个源文件，重新make，察看编译过程。

（5）通过使用makefile变量和隐含规则，对makefile文件进行简化

1. **写出详细的试验过程及其试验结果。**