第五章　GCC编译器

第五章：GCC编译器

目标:
本章使学员熟练掌握linux操作系统下GCC编译器的使用，通过本课的学习，学员应该掌握如下知识：

- ☑ 了解GCC编译选项
- ☑ 掌握如何利用GCC编译程序

学时：4 学时

教学方法：讲授ppt＋上机操作+实例演示

5.1 GNU gcc简介

- 目前Linux下最常用的C语言编译器是gcc（GNU Compiler Collection），它是GNU项目中符合ANSI C标准的编译系统．
- 是Linux 的基石,操作系统内核和大部分程序都是gcc 编译的,是Linux下最重要开发工具之一
- gcc 早期是c的编译器, 后来发展能支持c,c++和object C,它可以通过不同的前端模块来支持各种语言，如Java、Fortran、Pascal、Modula-3和Ada等。
- gcc 是一个交叉平台的编译器,目前支持几乎所有主流CPU的处理器平台.

5.1.1 gcc支持的文件格式

- gcc 支持源码格式
  - .c C源程序;
  - .C,.cc,.cxx,.cpp C++源程序;
  - .m Objective-C源程序;
  - .i 预处理后的C文件;
  - .ii 预处理后的C++文件;
  - .s 汇编语言源程序;
  - .S 汇编语言源程序;
  - .h 预处理器文件;
  - .o 目标文件(Object file)

注：gcc编译时是对输入文件扩展名是敏感的, 这一点跟大部Linux程序不一样。

5.1.2 gcc组成

- gcc 一般安装在 /usr/bin
- gcc 是一组编译工具的总称,包含如下工具
  - C 编译器 cc,cc1,cc1plus,gcc
  - C++编译器 c++,cc1plus,g++
  - 源码预处理程序 cpp,cpp0
  - 库文件 libgcc.a,libgcc_eh.a,libgcc_s.so,ibiberty.a,libstdc++,libsupc++.a

5.1.3 gcc的起步

Neuedu

- 生成一个helloworld程序
  - gcc hello.c -o hello #把hello.c编译成一个可执行程序 hello
  - gcc hello.c　#不指定输出名,生成一个a.out

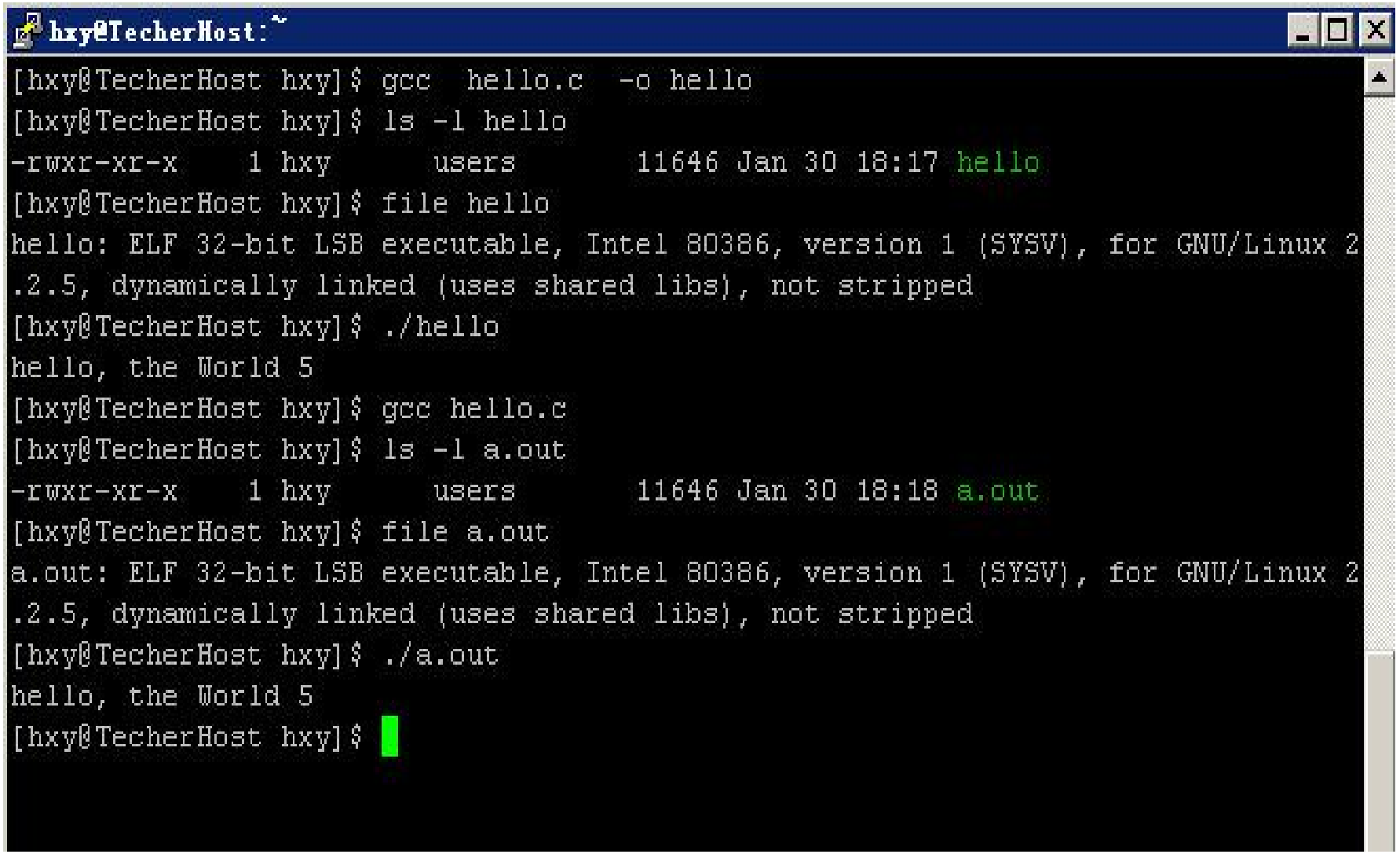
```
#include <stdio.h>

#define MY_NUMBER 5

int main(void)
{
    printf("Hello %d,the World!\n",MY_NUMBER);
    return 0;
}
```

5.1.3 gcc的起步

Neuedu



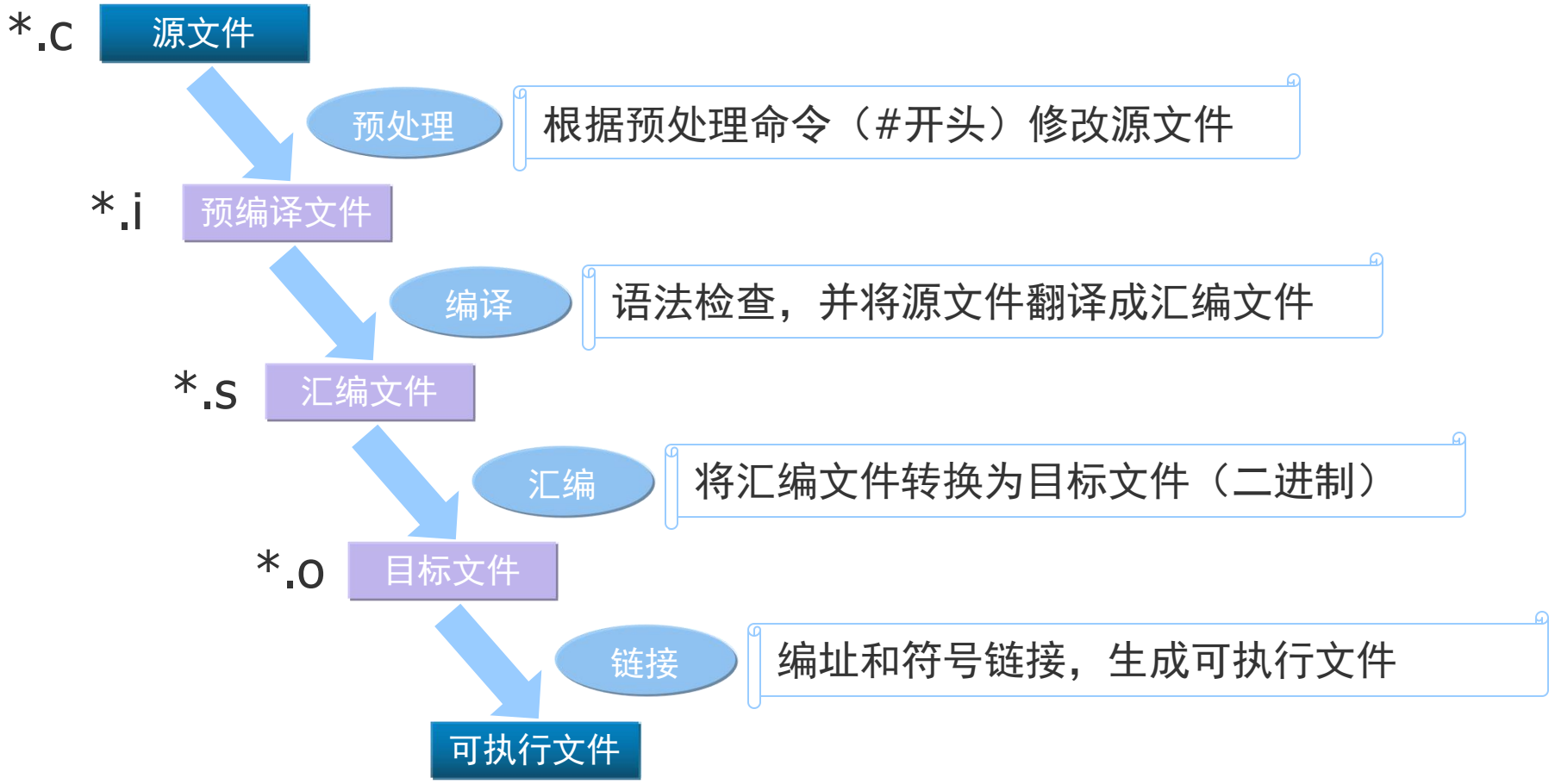
5.2 用gcc编译程序

- 任何一个可执行程序从源代码到可执行的二进制程序之中都要经过固定的几步
  - 预处理 （Pre-Processing）
    - 这一步完成对代码的预处理
  - 编译（Compiling）
    - 将源代码编译成汇编代码
  - 汇编（Assembling）
    - 将汇编代码汇编成目标文件
  - 链接（Linking）
    - 将目标代码和所需要库的链成一个完整的应用程序
- 集成开发环境(IDE)自动协助开发完成这几步,如VC++
- 在Linux 下,如果使用命令行开发工具(gcc,ld,ar)等,需要用户手工调用这一些命令来完成这几步骤.

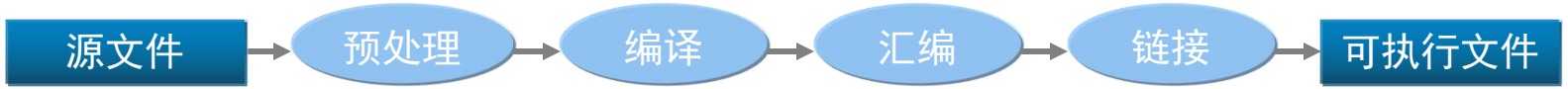
5.2.1 为什么要用gcc

- 随着Linux的GUI改进,也出现了越来越多的IDE开发环境，很多IDE基本上是基于gcc编译。
- 而且大部分项目,包括嵌入式开发,都提供gcc命令行开发模式。gcc是免费的。
- 大家都用gcc，易于发现问题。
- 用gcc开发是Linux和嵌入式开发的必须使用的工具，也是基本功之一。

5.2.2 hello编译过程分析



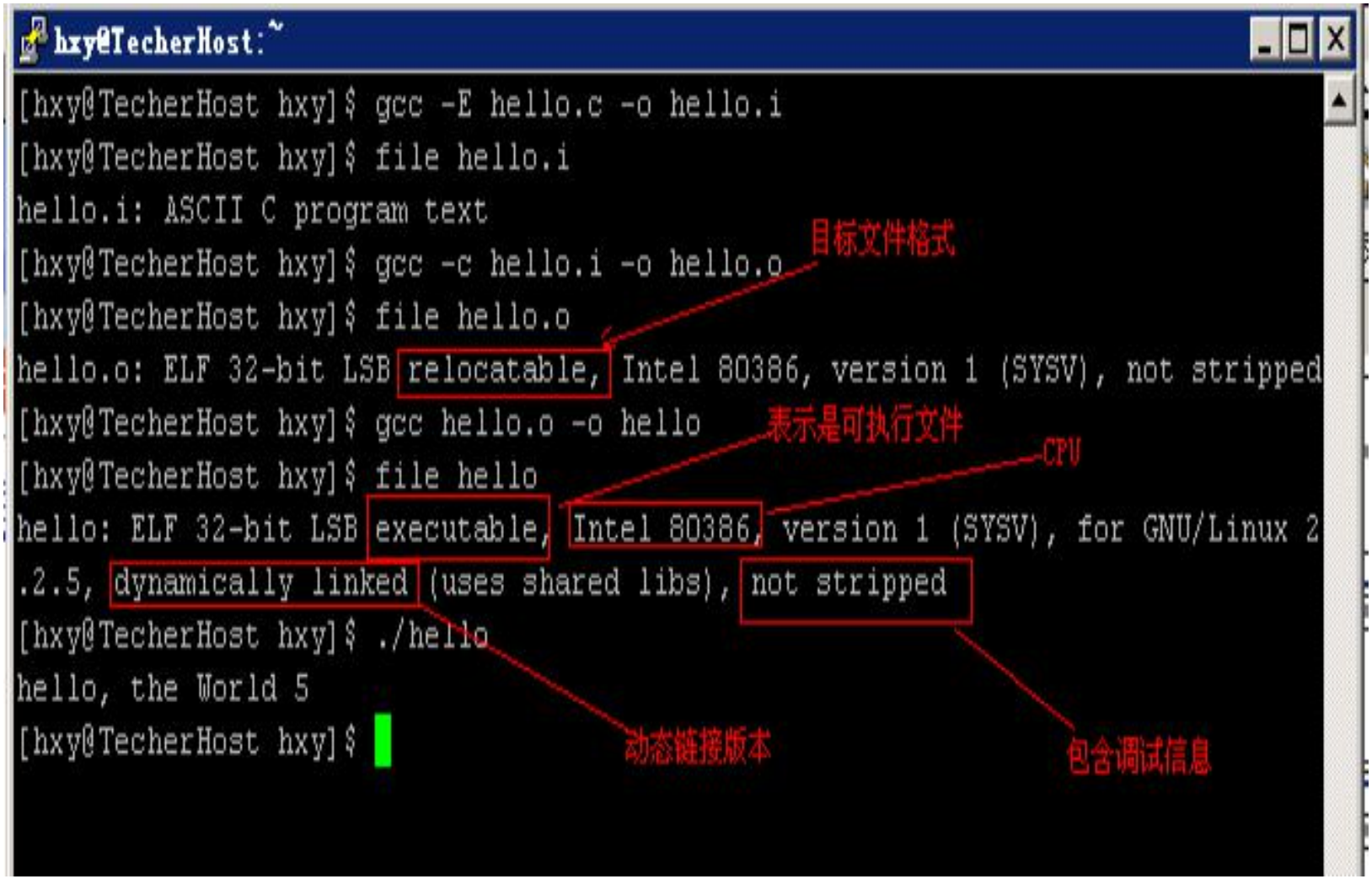
5.2.2 hello编译过程分析



- 预处理(Pre-Processing)
  # gcc -E hello.c -o hello.i
- 编译(Compiling)
  # gcc -S hello.i -o hello.s
- 汇编(Assembling)
  # gcc -c hello.s -o hello.o
- 链接(Linking)
  # gcc hello.o -o hello

注意: gcc的结果输出是后缀名不相关的.只与输出参数相关，这跟一般Linux程序是一样。例如gcc hello.c -o hello.o#虽然后缀名是.o，但实际是一个应用程序。

5.2.2 hello编译过程分析



5.2.3 用gcc构造程序(1)

Neuedu

- 如果可执行程序只由一个源代码文件构造出来，那么只需要用到如下形式
  - gcc hello.c -o hello
    - 表示将hello.c一次做完四步,构造出可执行程序 hello ,
  - gcc hello.c
    - 将hello.c构造一个可执行程序,有缺省名a.out,但不建议这样做.
  - gcc -c hello.c -o hello
    - 这一步骤是初学者常犯错误.以为等于在一次性构造应用程序hello
    - 但实际上这只是在编译c并生成一个目标文件hello,即便是没有.o的后缀.这个用file命令可以很容易查看,这个hello是无法执行.

5.2.3 用gcc构造程序(2)

- 使用多个源码的项目,如项目中包含2个以上的源代码,一般要先将源代码编译成目标代码.最后一次链接成可执行程序
- 也可以直接把两个文件在一句里编译,但建议不要这样做
  - gcc dl_list.c dl_file.c test_link.c -o test_link

```c
//main.c
#include "main.h"
int main()
{
    add();
    del();
    modify();
    return 0;
}
```

```c
//add.c
#include <stdio.h>
void add()
{
    printf("add\n");
}
```

```c
//del.c
#include <stdio.h>
void del()
{
    printf("del\n");
}
```

```c
//main.h
void add();
void del();
void modify();
```

```c
//modify.c
#include <stdio.h>
void modify()
{
    printf("modify\n");
}
```

思考:
如果main.h与源文件
不在同一目录会怎样

5.2.3 用gcc构造程序(3)

- 对于有头文件在多个目录时,需要在编译时多次使用用-I参数加入头文件所在目录.
  - 例如test_link.c需要用到 /usr/include, 当前目录(.),/home/hxy目录下的头文件.则如下编译
    gcc -I. -I/usr/include -I/home/hxy -c test_link.c

5.2.4 多文件gcc的处理

- 在采用模块化的设计思想进行软件开发时，通常整个程序是由多个源文件组成的，相应地也就形成了多个编译单元，使用GCC能够管理这些编译单元。假设有一个由foo1.c和foo2.c两个源文件组成的程序，为了对它们进行编译，并最终生成可执行程序foo，可以使用下面这条命令：
  - gcc foo1.c foo2.c -o foo
- 但在编译一个包含许多源文件的工程时，若只用一条GCC命令来完成编译是非常浪费时间的。假设项目中有100个源文件需要编译，并且每个源文件中都包含10000行代码，如果像上面那样仅用一条GCC命令来完成编译工作，那么GCC需要将每个源文件都重新编译一遍，然后再全部连接起来。很显然，这样浪费的时间相当多，尤其是当用户只是修改了其中某一个文件的时候，完全没有必要将每个文件都重新编译一遍，因为很多已经生成的目标文件是不会改变的。

5.2.4 多文件gcc的处理

- 一个大型项目,一个可执行程序可能拥有多个位于不同目录的头文件,多个源码文件,还可能链接一些静态库或动态库,这一些都需要用到gcc的一些扩展选项.
  - gcc的参数参见下一节
- 可能调用gcc很多次,如果完全手工编写,将是一个浩大的工程
  - 需要写一个类似Shell脚本的Makefile来调用gcc构造

练习

- 请开发一个程序,使用结构定义

```
struct student{
        int no; /* 学号 */
    char name[64]; /* 名称 */
        float  height; /* 身高 */
        };
```

要求创建3个学生定义,并给三个学生的三个成员都赋上值,
写一个show函数依次把所有学生信息显示，要求用vi编辑,并用gcc
在Linux下编译,测试通过。
提示：创建一个工程student目录，将结构体类型、show()函数原型
写到show.h，将show()定义写到show.c里，main()写到main.c 里，
.h文件放在include目录下，.c 放到source目录下。

5.3 gcc的选项

关于宏(*macro*)的选项

- **-D**macro
  - 定义宏*macro*,宏的内容定义为**printf**.
  - gcc test_m.c –DPRINTF=printf –o test_m
- **-D***macro*=*defn*
  - 定义宏DBG_NUM的内容为10
  - gcc test_m.c –DDBG_NUM=10 –o test_m

5.3 gcc的选项

gcc警告提示功能(2)

- **-Wall**
  - 打开所有编译警告
  - gcc -Wall illcode.c -o illcode
- **-Werror**
  - 视警告为错误;出现任何警告即放弃编译.
  - gcc -Wall -Werror illcode.c -o illcode
- -w
  - 禁止输出警告

```
[hxy@TecherHost gcc]$ gcc -Wall illcode.c -o illcode
illcode.c:4: warning: return type of `main' is not `int'
illcode.c: In function `main':
illcode.c:5: warning: unused variable `var'
[hxy@TecherHost gcc]$ gcc -Werror illcode.c -o illcode
cc1: warnings being treated as errors
illcode.c: In function `main':
illcode.c:4: warning: return type of `main' is not `int'
[hxy@TecherHost gcc]$
```

练习

- 将hello.c用gcc的-D选项参数生成两个版本的可执行程序，如果增加了DEBUG宏，则打印“DEGUG”，如果没有DEBUG宏，打印“RELEASE”。

```
[root@localhost root]# gcc hello.c -o hello
[root@localhost root]# ./hello
RELEASE
[root@localhost root]# gcc hello.c -DDEBUG -o hello
[root@localhost root]# ./hello
DEBUG
```