

知识点 1【内存的分区】

主要强调的是运行中的进程内存分区。

在 32 位系统中，系统会为每一个进程分配 4G 的空间（虚拟的）。



知识点 2【各类变量】（背）

1、普通局部变量

定义形式：在{}里面定义的变量为普通局部变量。

```
void test01()  
{  
    int a = 0;  
    {  
        int b = 0;  
    }  
}
```

作用范围：在{}里面有效

生命周期：{}复合语句结束 局部变量被释放。

内存区域：栈区

注意事项：

1、局部变量不初始化 内容不确定

2、局部变量如果同名 就近原则。

```
void test01 ()
{
    int a = 10;
    {
        int a = 20;
        printf("%d\n",a) ;//20
    }
}
```

2、普通全局变量

定义形式：在函数外定义的变量。

```
#include <stdio.h>
int data;//全局变量
void test01 ()
{
}
```

作用范围：当前源文件以及其他源文件都有效

生命周期：整个进程（运行的程序结束后 才释放）

内存区域：全局区

注意事项：

- 1、全局变量不初始化 内容为 0
- 2、如果其他源文件 要使用全局变量 必须在使用处加 extern 声明。
- 3、全局变量和局部变量同名优先选择局部变量

3、静态局部变量

定义形式：在{}里面定义 加 static 修饰

```
void test01 ()  
{  
    static int data = 20;  
    printf ("%d\n", data);  
}
```

作用范围：{}里面有效

生命周期：整个进程（运行的程序结束后 才释放）

内存区域：全局区

注意事项：

- 1、静态局部变量不初始化 内容为 0
- 2、静态局部变量 只会定义一次

4、静态全局变量

定义形式：全局变量前加 static 修饰

作用范围：只在当前源文件有效。

生命周期：整个进程（运行的程序结束后 才释放）

内存区域：全局区

注意事项：

- 1、不初始化为 0
- 2、只在当前源文件有效

5、全局函数

定义的函数默认为全局函数。

只要在其他源文件加 extern 声明 就可以在其他源文件中使用。

00_code.c	00_fun.c
<pre>#include <stdio.h> extern 说明func来之其他源文件 extern void func(); void test01() { func(); } int main(int argc, char *argv[]) { test01(); }</pre>	<pre>#include <stdio.h> void func() { printf("00_fun.c中的函数\n"); } 全局函数</pre>

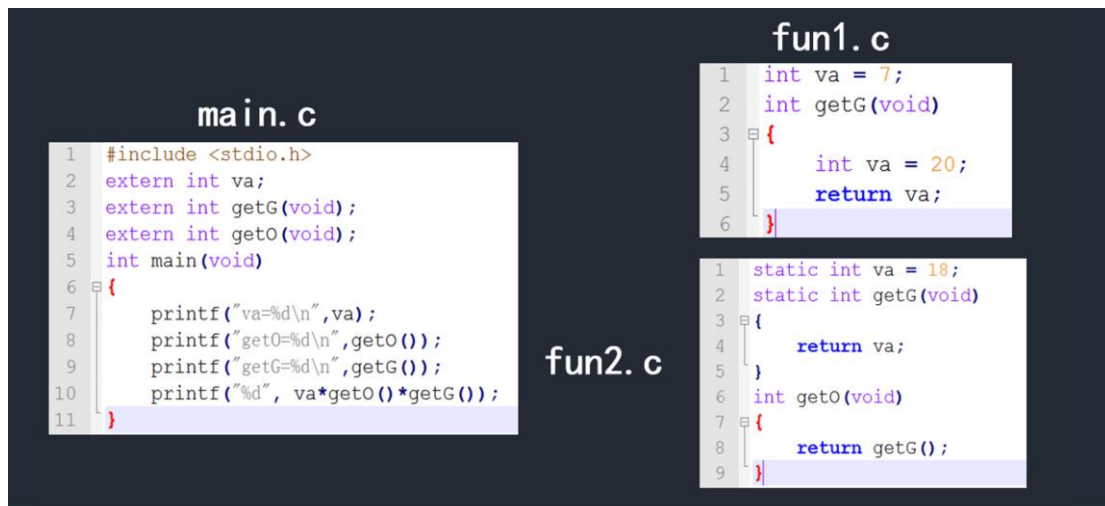
6、静态函数

函数返回值类型前 加 static 修饰 为静态函数

```
#include <stdio.h>
static void func()
{
    printf("00_fun.c中的函数\n");
}
```

静态函数不能被其他源文件使用。（重要）

7、案例



知识点 3 【gcc 的编译过程】背

预处理、编译、汇编、链接

1、**预处理**：头文件包含、宏替换、条件编译、删除注释（不作语法检查）

gcc -E 01_code.c -o 01_code.i

2、**编译**：将预处理好的.i 编译成 汇编文件.s (作语法检查)

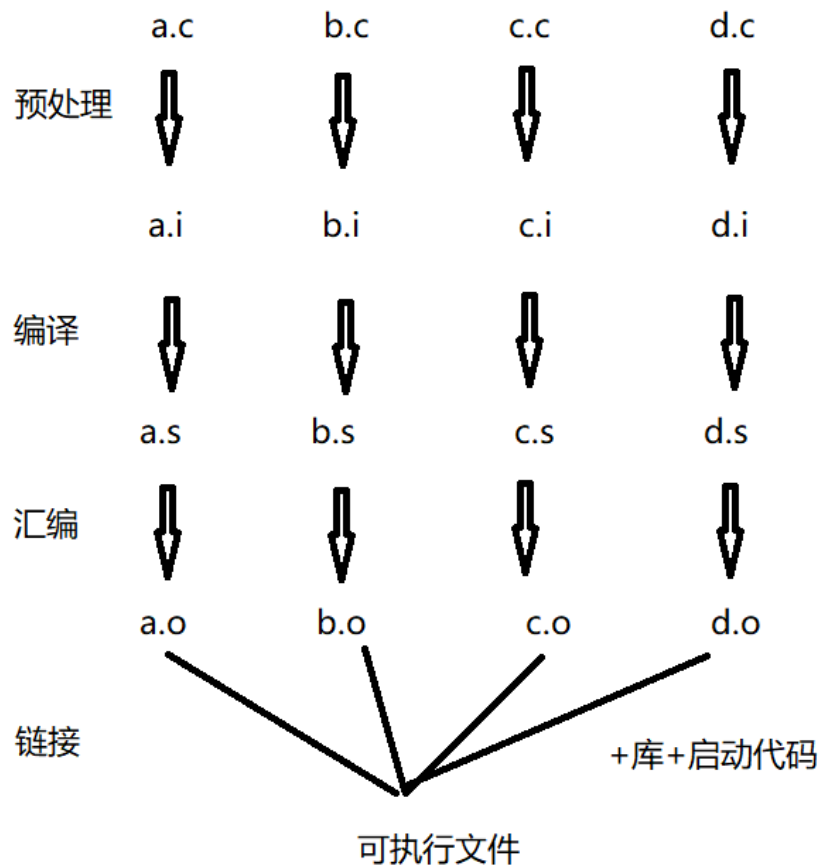
gcc -S 01_code.i -o 01_code.s

3、**汇编**：将汇编文件.s 生成 二进制文件.o

gcc -c 01_code.s -o 01_code.o

4、**链接**：将各个独立的二进制文件+库函数+启动代码 生成**可执行文件**

gcc 01_code.o -o main

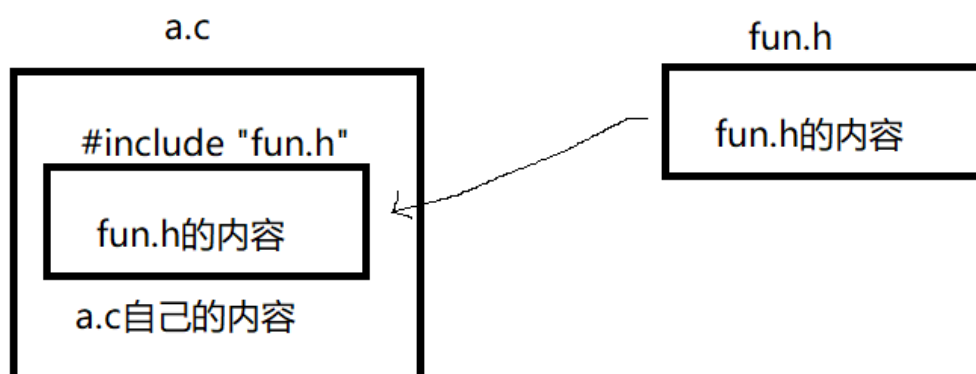


一步到位:

`gcc 源文件 -o 可执行文件名`

`gcc 源文件` 默认生成的可执行文件名为 `a.out`

知识点 4 【include 头文件包含】



#include "":从**当前目录**查找头文件, 如果找不到 才从**系统指定目录**找头文件。(包用户定义的头文件)

#include<>: 只从**系统指定目录**去找头文件。(包含系统头文件)

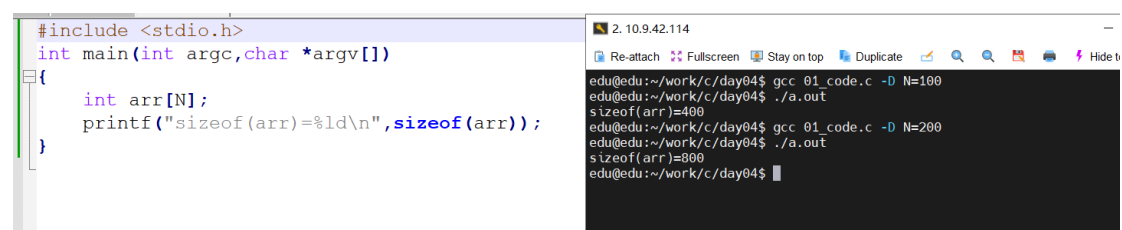
知识点 5 【define 定义宏】

1、无参的宏

```
#define N 100
```

在预处理阶段 将所有出现 N 的位置替换成 100

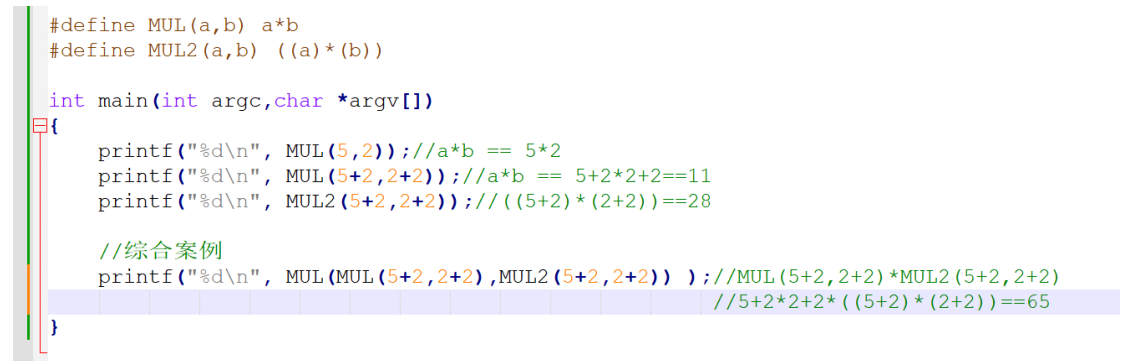
```
int arr[N];
```



```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int arr[N];
    printf("sizeof(arr)=%ld\n", sizeof(arr));
}
```

```
edu@edu:~/work/c/day04$ gcc 01_code.c -D N=100
edu@edu:~/work/c/day04$ ./a.out
sizeof(arr)=400
edu@edu:~/work/c/day04$ gcc 01_code.c -D N=200
edu@edu:~/work/c/day04$ ./a.out
sizeof(arr)=800
edu@edu:~/work/c/day04$
```

2、有参的宏 (宏函数)



```
#define MUL(a,b) a*b
#define MUL2(a,b) ((a)*(b))

int main(int argc, char *argv[])
{
    printf("%d\n", MUL(5,2)); //a*b == 5*2
    printf("%d\n", MUL(5+2,2+2)); //a*b == 5+2*2+2==11
    printf("%d\n", MUL2(5+2,2+2)); //((5+2)*(2+2))==28

    //综合案例
    printf("%d\n", MUL(MUL(5+2,2+2),MUL2(5+2,2+2)) ); //MUL(5+2,2+2)*MUL2(5+2,2+2)
    //5+2*2+2*(5+2)*(2+2)==65
}
```

3、宏函数和函数的区别 (背)

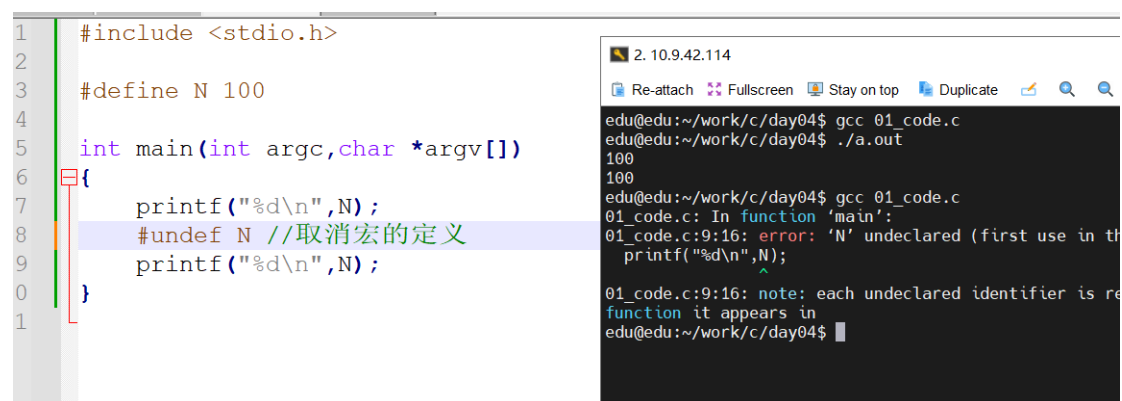
宏函数:

1、带参的宏 在**预处理**展开 有大量重复代码(占空间) 没有函数调用带来的**出入栈**的开销(**时间**)。用空间换时间。

- 2、宏的参数没有类型 不能保证参数的完整性
- 3、宏没有作用域的限制 不能作为结构体或类的成员 (**重要**)

函数:

- 1、函数调用 需要出入栈的开销 (耗时间), 代码只有一份 (节约空间)。(用时间换空间)
- 2、函数的参数 有类型 可以保证参数的**完整性**。
- 3、有作用域的限制 能作为结构体或类的成员。



The image shows a code editor on the left and a terminal window on the right. The code editor contains a C program that defines a macro `N` as 100, then defines a function `main` that prints `N` twice. The second use of `N` is preceded by `#undef N`, which is highlighted in blue. The terminal window shows the compilation process: `gcc 01_code.c` and `./a.out`. The first run prints 100. The second run, after re-compiling, shows an error: `01_code.c:9:16: error: 'N' undeclared (first use in this function)`, with a caret pointing to the `N` in the `printf` statement. Below the error, a note states: `note: each undeclared identifier is reported before the translation unit`.

```
1 #include <stdio.h>
2
3 #define N 100
4
5 int main(int argc, char *argv[])
6 {
7     printf("%d\n", N);
8     #undef N //取消宏的定义
9     printf("%d\n", N);
10 }
11
```

```
2. 10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day04$ gcc 01_code.c
edu@edu:~/work/c/day04$ ./a.out
100
100
edu@edu:~/work/c/day04$ gcc 01_code.c
01_code.c: In function 'main':
01_code.c:9:16: error: 'N' undeclared (first use in this function)
    printf("%d\n", N);
                   ^
01_code.c:9:16: note: each undeclared identifier is reported before the translation unit
edu@edu:~/work/c/day04$
```

知识点 6 【条件编译】

<code>#ifndef XXX</code>	<code>#ifdef XXX</code>	<code>#if XXX</code>
语句1;	语句1;	语句1;
<code>#else</code>	<code>#else</code>	<code>#else</code>
语句2;	语句2;	语句2;
<code>#endif</code>	<code>#endif</code>	<code>#endif</code>


```

1 #include <stdio.h>
2 #include <string.h>
3 int main(int argc, char *argv[])
4 {
5     char buf[128] = "";
6     printf("请输入一个字符串:");
7     fgets(buf, sizeof(buf), stdin);
8     buf[strlen(buf)-1] = '\0';
9
10    //将大小 转换成小写
11    int i=0;
12    while(buf[i] != '\0')
13    {
14        #ifdef BIG_TO_SMALL
15        if(buf[i] >= 'A' && buf[i] <= 'Z')
16        {
17            buf[i] += 32;
18        }
19        #else
20        if(buf[i] >= 'a' && buf[i] <= 'z')
21        {
22            buf[i] -= 32;
23        }
24        #endif
25        i++;
26    }
27
28    printf("%s\n", buf);
29 }

```

```

2. 10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day04$ gcc 02_code.c -D BIG_TO_SMALL
edu@edu:~/work/c/day04$ ./a.out
请输入一个字符串:hello WORLD
hello world
edu@edu:~/work/c/day04$ gcc 02_code.c
edu@edu:~/work/c/day04$ ./a.out
请输入一个字符串:hello WORLD
HELLO WORLD
edu@edu:~/work/c/day04$

```

```

1 #include <stdio.h>
2 #include <string.h>
3 int main(int argc, char *argv[])
4 {
5     char buf[128] = "";
6     printf("请输入一个字符串:");
7     fgets(buf, sizeof(buf), stdin);
8     buf[strlen(buf)-1] = '\0';
9
10    //将大小 转换成小写
11    int i=0;
12    while(buf[i] != '\0')
13    {
14        #if BIG_TO_SMALL
15        if(buf[i] >= 'A' && buf[i] <= 'Z')
16        {
17            buf[i] += 32;
18        }
19        #else
20        if(buf[i] >= 'a' && buf[i] <= 'z')
21        {
22            buf[i] -= 32;
23        }
24        #endif
25        i++;
26    }
27
28    printf("%s\n", buf);
29 }
30

```

```

2. 10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day04$ gcc 02_code.c -D BIG_TO_SMALL=1
edu@edu:~/work/c/day04$ ./a.out
请输入一个字符串:hello WORLD
hello world
edu@edu:~/work/c/day04$ gcc 02_code.c -D BIG_TO_SMALL=0
edu@edu:~/work/c/day04$ ./a.out
请输入一个字符串:hello WORLD
HELLO WORLD
edu@edu:~/work/c/day04$

```

知识点 7 【防止头文件重复包含】

1、方式一：Linux 的方式 #ifndef 防止头文件重复包含

```
03_code.c
1 #include <stdio.h>
2 #include "a.h"
3 #include "b.h"
4
5 int main(int argc, char *argv[])
6 {
7     printf("data=%d\n", data);
8 }
9
```

```
a.h
1 #ifndef _A_H_
2 #define _A_H_
3
4 #include "b.h"
5
6 #endif
```

```
b.h
1 #ifndef _B_H_
2 #define _B_H_
3
4 int data = 100;
5
6
7 #endif
```

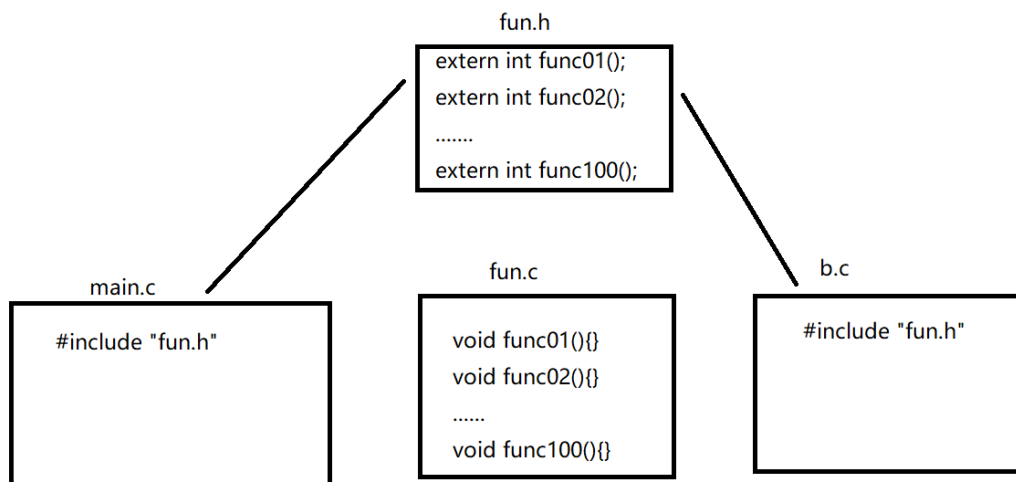
2、方式二：windows 的方式 防止头文件重复包含

```
03_code.c
1 #include <stdio.h>
2 #include "a.h"
3 #include "b.h"
4
5 int main(int argc, char *argv[])
6 {
7     printf("data=%d\n", data);
8 }
9
```

```
a.h
1 #pragma once
2
3 #include "b.h"
4
5
```

```
b.h
1 #pragma once
2
3 int data = 100;
4
5
```

知识点 8 【多文件编程】



04_code.c

```
#include <stdio.h>  
  
#include "04_fun.h"  
  
int main(int argc, char *argv[])  
{  
  
printf("%d\n", my_add(100,20));  
  
printf("%d\n", my_sub(100,20));  
  
printf("%d\n", my_mul(100,20));  
  
printf("%d\n", my_div(100,20));  
  
  
return 0;  
}
```

04_fun.c

```
int my_add(int a, int b)  
{
```

```
return a+b;

}

int my_sub(int a, int b)

{

return a-b;

}

int my_mul(int a, int b)

{

return a*b;

}

int my_div(int a, int b)

{

return a/b;

}
```

04_fun.h

```
#ifndef __04_FUN_H__

#define __04_FUN_H__

extern int my_add(int a, int b);

extern int my_sub(int a, int b);

extern int my_mul(int a, int b);

extern int my_div(int a, int b);

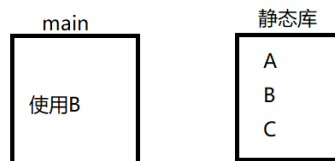
#endif
```

```
gcc 04_code.c 04_fun.c
```

```
./a.out
```

知识点 9 【静态库和动态库的制作】

1、静态库和动态库的区别

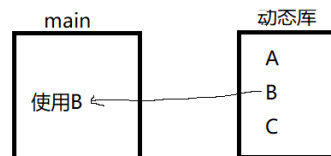


静态链接：将静态库的所有函数都链接到可执行文件中

优点：对库的依赖不大

缺点：

- 1、可执行文件大
- 2、如果库反生变化 需要 重新链接



1、在**链接**阶段 仅仅建立和所需库函数的连接关系

2、在**运行**阶段 才将所需的库函数 包含在可行文件中

优点：生成的可执行文件小

缺点：

- 1、对库的环境依赖特别大

```
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day04$ ls
00_code.c 01_code.c 01_code.o 02_code.c 03_code.c 05_code.c a.out fun
00_fun.c 01_code.i 01_code.s 02_code.i 04_code.c a.h b.h main
edu@edu:~/work/c/day04$ gcc 05_code.c -o main
edu@edu:~/work/c/day04$ gcc 05_code.c -o main1 -static
edu@edu:~/work/c/day04$ ls main* -lh
-rwxrwxr-x 1 edu edu 8.4K 7月 25 14:20 main
-rwxrwxr-x 1 edu edu 888K 7月 25 14:20 main1
edu@edu:~/work/c/day04$
```

2、制作静态库

2.1 制作静态库

1、将需要制作库的源文件生成二进制文件.o

```
gcc -c 04_fun.c -o 04_fun.o
```

2、使用 ar rc 将 04_fun.o 二进制文件生成库

```
ar rc libmylib.a 04_fun.o
```

注意：以 lib 开头，.a 结尾 库名称是 mylib

2.2 使用静态库

2.2.1 直接将库放入项目目录。

```
edu@edu:~/work/c/day04$ gcc 04_code.c libmylib.a
edu@edu:~/work/c/day04$ ./a.out
120
80
2000
5
edu@edu:~/work/c/day04$
```

2.2.2 将库放入指定的目录

```
edu@edu:~/work/c/day04$ ls
00_code.c  01_code.i  02_code.c  04_code.c  05_code.c  b.h  main1
00_fun.c   01_code.o  02_code.i  04_fun.c   a.h         fun
01_code.c  01_code.s  03_code.c  04_fun.o   a.out       main
edu@edu:~/work/c/day04$ gcc 04_code.c -I./fun -L./fun -lmylib
edu@edu:~/work/c/day04$ ./a.out
120
80
2000
5
edu@edu:~/work/c/day04$
```

-I 指的是头文件的路径 -L 库的路径 -l 指的库的名称

2.2.3 将库放入系统指定目录

系统默认的头文件路径: /usr/include

系统默认的库的路径:/usr/lib

```
1 #include <stdio.h>
2 #include <04_fun.h>
3 int main(int argc, char *argv[])
4 {
5     printf("%d\n", my_add(100,20));
6     printf("%d\n", my_sub(100,20));
7     printf("%d\n", my_mul(100,20));
8     printf("%d\n", my_div(100,20));
9
10    return 0;
11 }
```

```
2.10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day04$ ls
00_code.c  01_code.i  02_code.c  04_code.c  05_code.c  b.h  main1
00_fun.c   01_code.o  02_code.i  04_fun.c   a.h         fun
01_code.c  01_code.s  03_code.c  04_fun.o   a.out       main
edu@edu:~/work/c/day04$ gcc 04_code.c -lmylib
edu@edu:~/work/c/day04$ ./a.out
120
80
2000
5
edu@edu:~/work/c/day04$
```

3、制作动态库

3.1 制作动态库

gcc -shared 04_fun.c -o libmylib.so

3.2 使用动态库

3.2.1 动态库在项目 目录

```
edu@edu:~/work/c/day04$ ls
00_code.c  01_code.i  02_code.c  04_code.c  05_code.c  b.h        main
00_fun.c   01_code.o  02_code.i  04_fun.c   a.h        fun        main1
01_code.c  01_code.s  03_code.c  04_fun.o   a.out      libmylib.so
edu@edu:~/work/c/day04$ gcc 04_code.c libmylib.so
edu@edu:~/work/c/day04$ ./a.out
./a.out: error while loading shared libraries: libmylib.so: cannot open shared o
bject file: No such file or directory
edu@edu:~/work/c/day04$ export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
edu@edu:~/work/c/day04$ ./a.out
120
80
2000
5
edu@edu:~/work/c/day04$
```

3.2.2 将动态库放入指定路径

```
edu@edu:~/work/c/day04$ ls
00_code.c  01_code.i  02_code.c  04_code.c  05_code.c  b.h        main1
00_fun.c   01_code.o  02_code.i  04_fun.c   a.h        fun        main1
01_code.c  01_code.s  03_code.c  04_fun.o   a.out      main
edu@edu:~/work/c/day04$ gcc 04_code.c -I./fun -L./fun -lmylib
edu@edu:~/work/c/day04$ ./a.out
./a.out: error while loading shared libraries: libmylib.so: cannot open shared o
bject file: No such file or directory
edu@edu:~/work/c/day04$ export LD_LIBRARY_PATH=./fun:$LD_LIBRARY_PATH
edu@edu:~/work/c/day04$ ./a.out
120
80
2000
5
edu@edu:~/work/c/day04$
```

3.2.3 将动态库放入 系统目录

