

UART3 串口重定向工作记录

一、printf 的重定向

1、浅析标准库 printf 函数执行过程

在 main.cpp 文件主函数 main () 中使用 printf () 函数，由于此文件中包含了头文件 #include"stdio.h"所以会跳转到 stdio.h 中。

stdio.h 文件 219 行 int _EXFUN(sprintf, (const char *, ...) _ATTRIBUTE ((__format__ (__printf__, 1, 2)))));声明了 printf 函数在外部定义，以及 printf 的参数类型以及归属。**注：此文件是不能修改的。**

由于我们在 libc/stdio/printf.h 下声明了此函数，所以他会跳转到此处。

在 libc/stdio/printf.c 定义了函数 printf 的实现且包含头文件#include"printf.h",所以将跳转到此文件下执行 printf () 函数。在此文件的 printf () 函数中主要包含如下两重要过程：

①跳转本文件的内部函数_vsnprintf()执行整个判断输出过程（判断输出是否为空，是否逐字符打印，是否特殊数据类型%c等等）②在输出过程中如果用_out_char () 函数，则跳转到本文件下定义的_out_char () 函数来执行，此函数中调用了_putchar()函数来输出字符。

由于_putchar()函数不是在此文件中定义的，且此文件包含了头文件#include"printf.h",在 printf.h 文件下，用 extern void _putchar(char character);声明了_putchar()函数是外部定义的。

所以程序会跳转到 libc/putchar.c 文件下执行此函数，调用控制台输出函数 console_putchar()。而在此文件下又包含了#include"console.h"头文件，所以会跳转过去执行与控制台相关的后续程序。

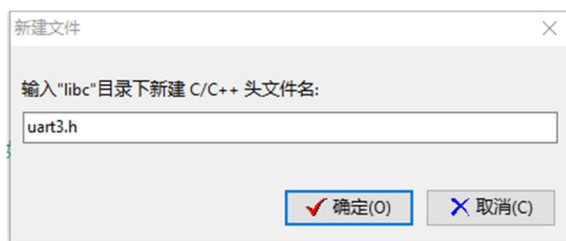
2、简述如何将串口函数 printf 重定向

①创建与串口 UART3 相关的两个文件 uart3.h 与 uart3.c 以实现串口三的初始化、输入与输出等底层操作，具体操作如下：

如下图，在 LoongIDE 集成开发软件左侧的“项目视图”中右键单击“libc”文件夹，选择“新建头文件”选项。



如下图，在弹出的对话框中输入新建头文件的名称“uart3.h”,并点击“确定”。



在新建的 uart3.h 文件中添加如下代码并保存

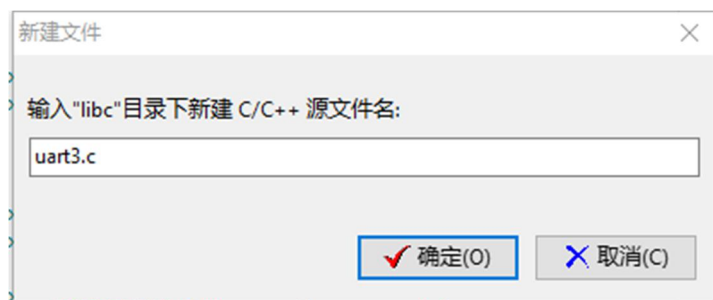
此处省略代码

至此，uart3.h 文件创建成功，接下来创建 uart3.c 文件。

如下图，在 LoongIDE 集成开发软件左侧的“项目视图”中右键单击“libc”文件夹，选择“新建源代码文件”选项。



如下图，在弹出的对话框中输入新建头文件的名称“uart3.c”，并点击“确定”。



在新建的 uart3.c 文件中添加如下代码并保存

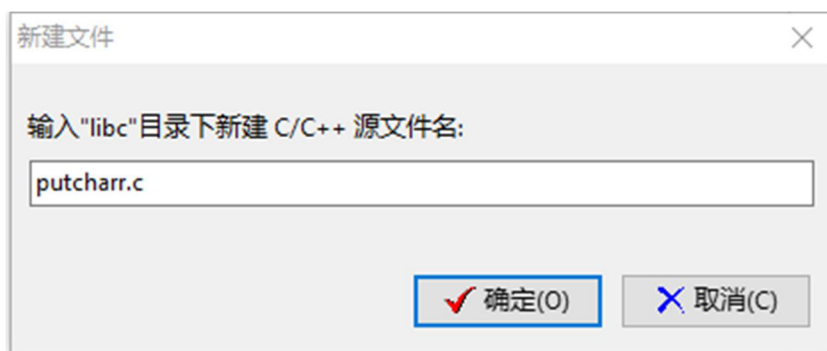
此处省略代码

②分析 putchar.c 文件，单独创建 putcharr.c 文件，在不影响控制台（uart5）的情况下重定向到 uart3。实现调用“uart3.c”文件中的串口底层操作程序通过串口三输出单个字符，具体操作如下：

如下图，在 LoongIDE 集成开发软件左侧的“项目视图”中右键单击“libc”文件夹，选择“新建源代码文件”选项。



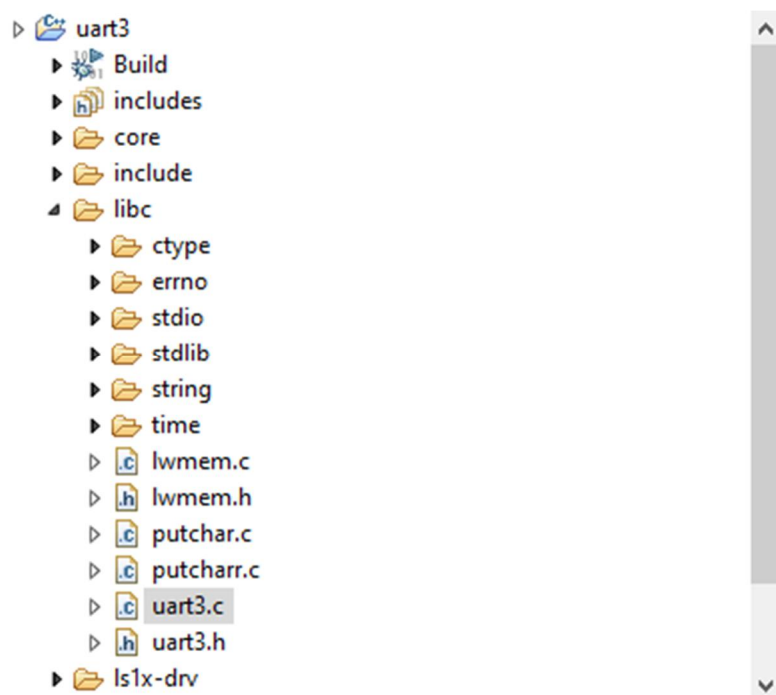
如下图，在弹出的对话框中输入新建头文件的名称“putcharr.c”,并点击“确定”。



在新建的 putcharr.c 文件中添加如下代码并保存

此处省略代码

至此 uart3.h、uart3.c 以及 putcharr.c 文件创建完毕，这些文件均保存在“此工程/libc/”路径下。双击打开 LoongIDE 集成开发软件左侧项目视图窗口下的 libc 文件夹即可看到。



③修改 printf.h 与 printf.c 文件，在源文件的基础上添加一些函数，以达到在不影响原控制台功能的前提下，依旧可以使用串口三。具体操作步骤如下：

针对 printf.h 文件的修改，在其文件下添加如下代码：

此处省略代码

上方前两行声明了_putchar1() 与 uart3_initialize()是来自外部的函数(分别在 putcharr.c

与 uart3.c 文件中实现)，以便完成 printf.c 文件中对二者的函数调用。而第三行代码声明了我们要新建的串口三输出的函数 iprintf ()【注：此函数名不可以任意取名】。

针对 printf.c 文件的修改，在其文件下添加如下代码：

此处省略代码

至此就完成了 printf () 的重定向。

3、针对重定向后的 printf 做测试

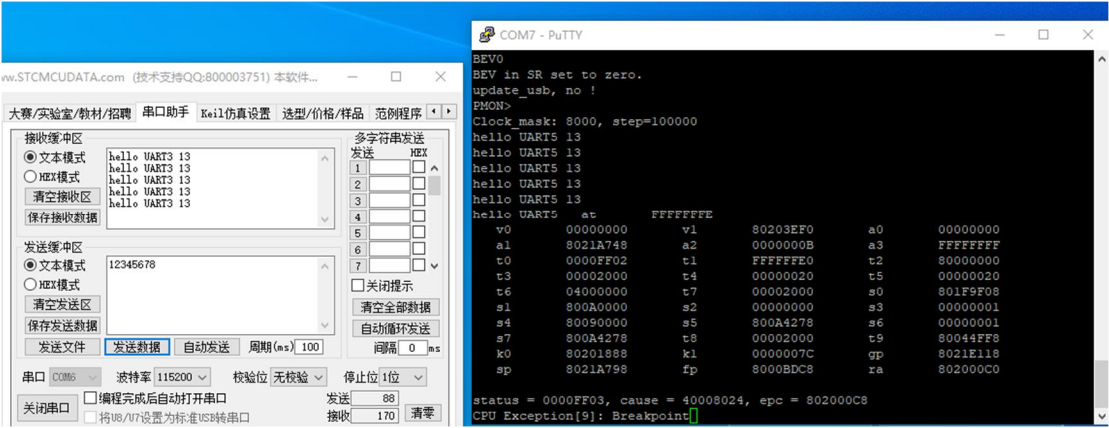
在 main.cpp 文件中添加#include “stdio.h”即可调用 printi()函数向 uart3 输出。因为上述修改没对控制台（uart5）做任何修改，所以依旧可以调用 printf () 函数向 uart5 输出。

此外，添加#include "libc/uart3.c"后，主函数可直接调用 uart3 的底层函数。

在此附加如下代码用于测试串口重定向是否成功，在 main.cpp 文件添加：

此处省略代码

每输入一次八位数据，均在串口三和串口五输出不同内容，直至 for 循环结束，在串口五输出 hello UART5 后结束。原控制台输出的信息依旧在串口五，不能更改。测试结果如下：



可以看到，上图测试结果完全符合预期。

4、printf 重定向中的那些坑：

①关于串口三初始化函数的使用：

主函数 main () 中不对串口三初始化则串口三不能输出任何信息，必须初始化。

重定向 printi()输出函数中不对串口三初始化则会输出乱码，必须初始化。

函数_putchar1()对串口三初始化会出现乱码，不能初始化。

②关于 include 路径的问题：

Include 只会在同级文件下查找，否则要带文件路径，用“../”表示返回上级路径。故只要路径正确可以添加任意头文件或者源文件。

③重定向为什么不修改原有的 printf () 或者 printk () 函数，而是新建 printi () 函数？

因为 printf () 与 printk () 是原有的控制台输出函数。如果修改 printf () 函数则控制台提示信息“Clock_mask”、“step”与程序运行结束输出的寄存器值、指针值等都会显示在串口三。如果修改 printk () 函数，虽然控制台提示信息“Clock_mask”、“step”会显示在串口五，但与程序运行结束输出的寄存器值和指针值等依旧会显示在串口三。这是我們不想看到的。

④为什么新建 printi () 函数，而不是其他什么函数名？

因为主函数在调用输出函数的时候，是通过其包含头文件#include“stdio.h”来查找的，而此头文件是禁止我们修改的，所以只能新建此头文件下已经有声明过的函数。并且此文件对函数参数类型等特征做了规定，并不是每个都能拿来用。我们需要找到和原始 printf () 相同类型的函数，并且保证该函数没有被用过才能在外部重新声明并使用。显然 printi () 是我们找到的符合要求的函数。

⑤为什么新建 putcharr.c 文件，而不是对原有的 putchar.c 文件做修改？

同第三条一样，还是为了控制台功能的正常使用。

⑥串口三与串口五在函数执行过程中，只有 print.c 文件中_vsnprintf () 函数是共用的，其他都是新加的。综上所述，与其叫“串口重定位”不如叫“如何制作串口三的驱动”。因为没改变原有控制台（串口五）任何功能，还加上了串口三的功能。

⑦关于换行：

控制台串口五用 PUTTY 软件打开，串口三用串口调试助手打开。串口五换行用\n，而串口三用\r\n。其实转换到 HEX 模式即可看到，串口三发送的数据\r 为 0D，\n 为 0A。

二、scanf 重定向

1、浅析标准库函数 scanf 的执行过程

如果在主程序中调用 scanf() 函数时，主函数所在文件会根据其包含的头文件#include“stdio.h”，跳转到该头文件寻找 scanf() 的定义。

在 stdio.h 文件中针对 scanf() 函数有如下定义，int _EXFUN(scanf, (const char *, ...) _ATTRIBUTE ((__format__ (__scanf__, 1, 2))))；该文件声明 scanf() 是一个外部定义的函数，将从其他文件寻找。

在标准输入输出库中，scanf.h 下有该函数的声明，scanf.c 文件下有该函数的实现，所以会跳转到 scanf.c 执行。scanf 输入主要有以下两部分：①引号里面的部分为控制输入类型的字符串，以下称之为控制流；②引号外则为个数不固定的，需要被赋值的各变量的地址（地址即存储器地址，各变量值即对应地址下存储的数据）。

因为 scanf 函数需要被赋值的参量个数不固定，所以要用到可变参数列表 va_list ap 用于定义可变参数列表 ap，va_start 用于获取可变参数列表的第一个参数的地址，va_end 用于清楚可变参数列表。可变参数列表按顺序依次存储需要被赋值的变量的地址。

在 va_start 与 va_end 之间调用了—个非常重要的函数_doscan()，该函数输入一个指向控制流首地址的指针、可变参数列表以及 stdin (stdin 本质是一个指向文件的指针) 该文件存储的是从串口输入的数据（这些数据以字符类型存储在该文件中）——标准输入数据流（以下简称为数据流）。_doscan() 则利用这些输入，将数据流的数据按照控制流的指示转化为对应数据类型后，向可变参数列表中存放的地址的对应存储空间写入数据，这样就完成了

诸多变量的赋值。

_doscan()函数执行流程如下：①跳过控制流与数据流开头没用的空格；②检测控制流，如果读取到的非空字符不是“%”则对数据流进行检测，如果数据流与控制流不匹配，则发生错误退出函数执行，以此做到控制流与数据流的匹配；③读取到“%”后跳过去；④检测后边是不是“*”符号，如果是此符号代表只读取数据，不进行存储；⑤接下来判断控制流是不是数字，如果是则代表对数据宽度（即位数）进行控制；⑥然后判断控制流是不是特殊说明符“h”代表 short 类型、“l”代表 long 类型、“L”代表 double 类型；⑦接下来，控制流字符肯定就是输入数据的种类（八进制整数、十进制整数、十六进制整数、指针、浮点数、字符、字符串、扫描输入模式等）；⑧根据不同的标志位以及输入数据类型，按照对应的处理逻辑，向可变参数列表中存放的地址的对应存储空间写入数据。

附：这里以*va_arg(ap, unsigned long *) = (unsigned long) val;为例讲解可变参数列表的使用。va_arg 宏，获取可变参数的当前参数，返回指定类型并将指针指向下一参数。此处 va_arg 宏获取可变参数列表 ap（列表中存储需要被赋值变量的地址）当前参数，并规定其类型为“指向无符号长整型数据的指针”，“*”+“该指针”就是需要被赋值的变量。该代码实现了将 val 进行强制数据类型转换后，赋值给对应变量的操作。而且可变参数列表指针自动指向下一个参数，下一次直接调用 va_arg 即可，无需对其指针做任何修改。

2、简述如何将串口函数是 scanf 重定向

①首先修改 uart3.c 文件下的串口读取函数，以实现如下功能：

输入过程中逐字符回显，因为控制台输入与输出在同一个窗口不需要回显。串口调试助手输入与输出不在同一窗口，需要回显才能判断开发板是否受到数据以及收到的数据是否正确。

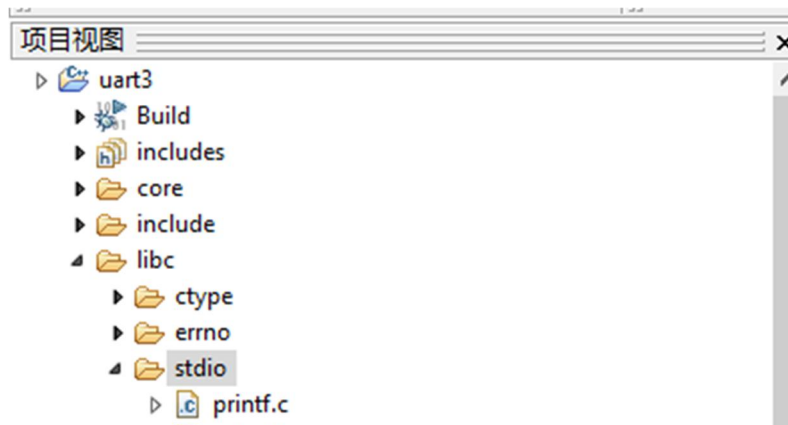
之前的 uart3_read 只能实现将指定长度字符串读入到缓存区中，使用极其不灵活。经修改后，只定义输入数据的最大长度。达到最大长度或者遇到回车换行结束，灵活度大大提高。

修改后的 uart3_read () 函数如下：

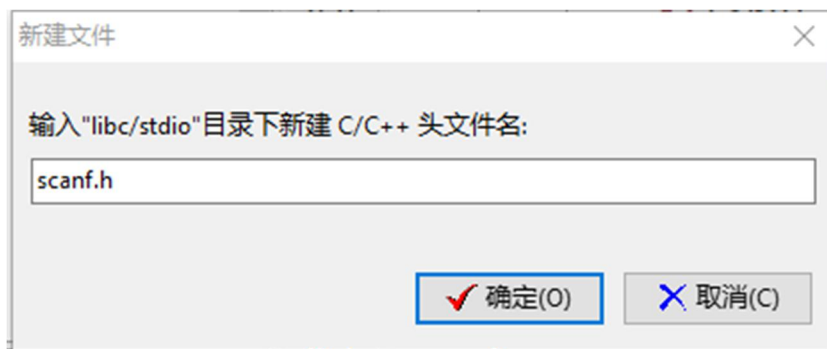
此处省略代码

②创建 scanf.h 与 scanf.c 两个文件，以实现串口输入重定向，具体操作如下：

如下图，在 LoongIDE 集成开发软件左侧的“项目视图”中双击打开“libc”文件夹，右键单击“stdio”文件夹，选择“新建头文件”选项。



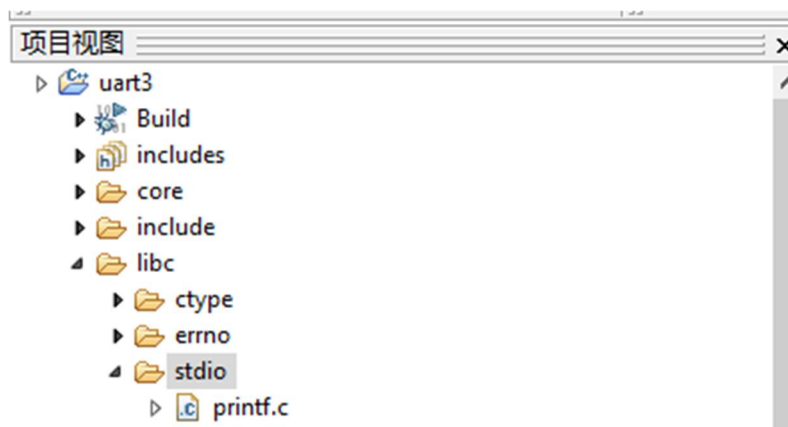
如下图，在弹出的对话框中输入新建头文件的名称“scanf.h”，并点击“确定”。



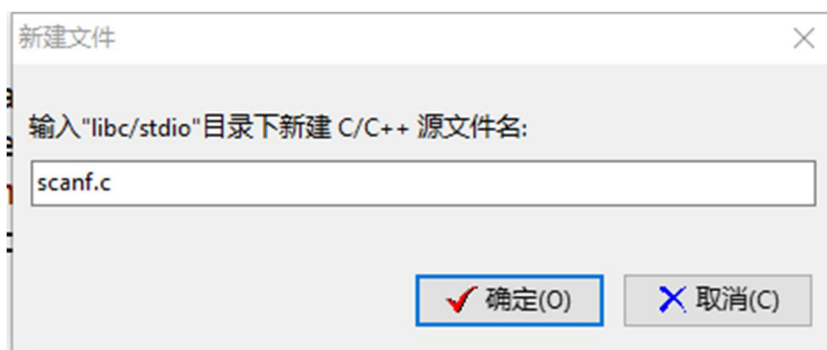
在新建的 scanf.h 文件中添加如下代码并保存

此处省略代码

如下图，在 LoongIDE 集成开发软件左侧的“项目视图”中，双击打开“libc”文件夹，右键单击“stdio”文件夹，选择“新建源代码文件”选项。



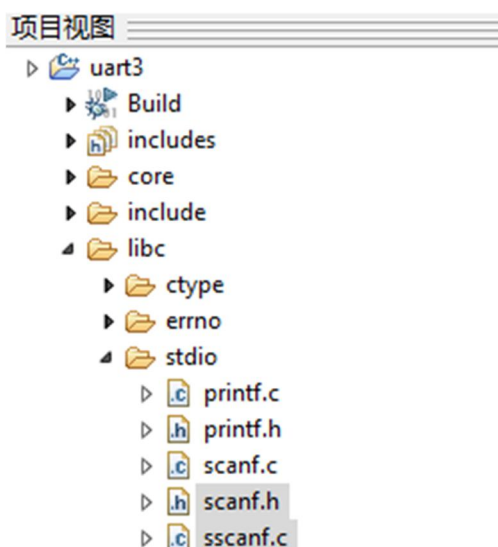
如下图，在弹出的对话框中输入新建头文件的名称“scanf.c”，并点击“确定”。



在新建的 scanf.c 文件中添加如下代码并保存

此处省略代码

结束后将在“libc/stdio/”文件夹下看到“scanf.h”与“scanf.c”两个文件，如下图所示：



③在主程序下添加如下代码，用于测试串口重定向：

此处省略代码

至此，串口重定向工作全部完成。

3、测试、对比与标准库函数的 scanf 的异同

在 scanf.c 与测试用的主程序文件 main.cpp 中用了大量的宏定义：scanf 中的宏定义决定是否支持某一功能，注释掉对应宏定义将其禁用；main.cpp 中的宏定义决定测试 scanf 函数的哪一功能，无需修改代码只需修改宏定义即可转变测试功能。

① 测试 scanf 函数的只读模式（%*）

在标准输入输出库中 scanf 语句控制流字符串带有符号“*”，表示对该数据只读取但不会

存储（即不会保存在变量中）。如下图所示，当输入数据为 123 和 456 时，数据 123 只读取不存储，所以不会存放于变量 a 中。注意：只读模式下没用调用 va_arg 宏，所以可变参数列表的指针并没有自动加一，所以第二个输入的数据 456 将会保存到变量 a 中，而不是变量 b。因为数据流读取完毕，没有值赋给变量 b，所以变量 b 保持初始值 0。在此，如果输入为 123、456 和 789，则 b 的值将会是 789。如下图所示，输入字符同样支持只读模式。

```

1 #include<stdio.h>
2 int main()
3 {
4     int a=0,b=0;
5     scanf("%d %d",&a,&b);
6     printf("%d %d \n",a,b);
7 }
    
```

```

1 #include<stdio.h>
2 int main()
3 {
4     char a='0',b='0';
5     scanf("%c %c",&a,&b);
6     printf("%c %c \n",a,b);
7 }
    
```

```

123 456
456 0
    
```

```

q w
w 0
    
```

重定向后的只读模式测试如下：

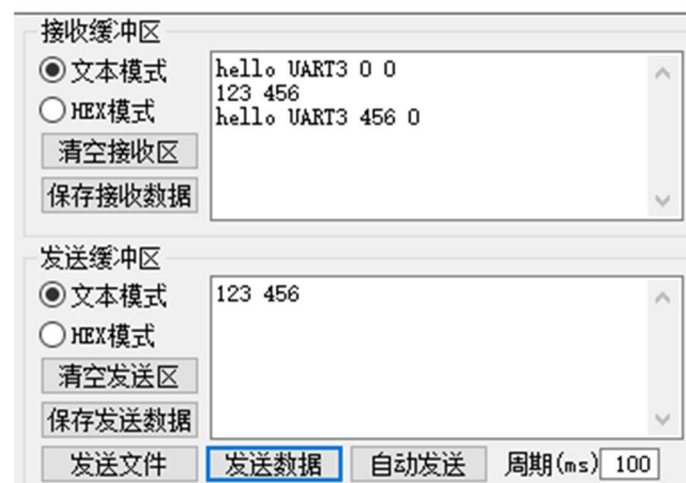
将 main.cpp 文件下除“#define TEST_NOASSIGN”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译。

编译成功后，用 EJTAG 线将开发板与电脑连接，以进行裸机程序下载。用串口线将开发板串口三与电脑 USB 端口进行连接，以进行串口重定向的测试。最后将开发板上电。

打开设备管理器，查看串口连接的端口号，打开串口调试助手根据上述端口号选择端口，并将波特率设置为 115200，校验位设置为无校验，停止位设置为 1 位，点击“打开串口”。并将串口调试助手的发送缓冲区与接收缓冲区都设置为文本模式。

回到 loongIDE 软件下，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“123 456”后按下回车，再点击发送数据。



由上图可以看出，与标准库函数相同，123 被识别为只读数据，456 被赋值给第一个变量。同样字符类型数据也支持只读模式，在此不做测试。

② 数据流与控制流匹配

在标准输入输出库中 scanf 语句控制流与输入数据流的内容要匹配。例如在下图中，控制流在第一个“%”前使用了“input is”则在输入数据流的时候也必须使用“input is”，控制流在第一个“%”与第二个“%”之间使用逗号作为分隔符，则在输入数据流的时候也必须使用逗号将两

个数据隔开。

```
#include<stdio.h>
int main()
{
    int a1 = 0,a2 = 0;
    scanf("input is %d,%d",&a1,&a2);
    printf("%d %d \n",a1,a2);
}
```

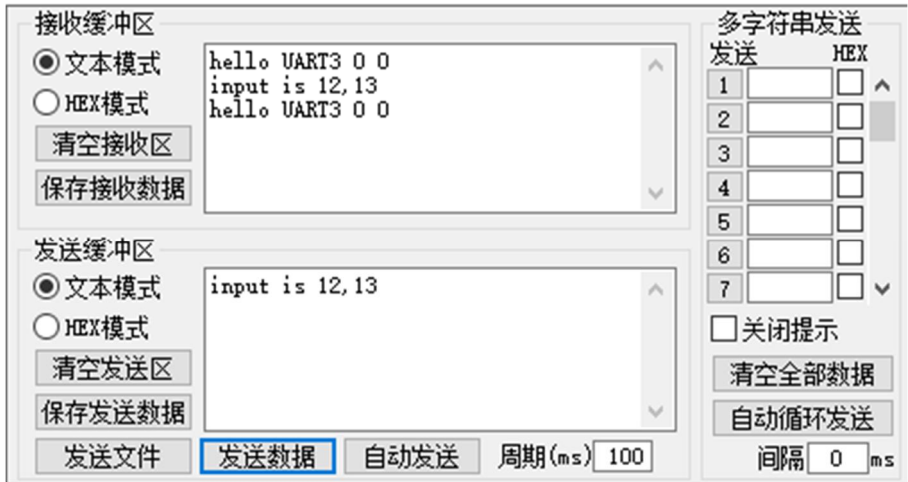
C:\Users\Administrator\Desktop\未命名1.exe

input is 12,13
12 13

重定向后的测试如下：

将 main.cpp 文件下除“#define TEST_MATCHING”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“input is 12, 13”后按下回车，再点击发送数据。



如上图所示，重定向后的 scanf 并不支持控制流与数据流匹配。在重定向过程中对标准 scanf 函数的语法结构上做了以下修改：控制流“%”符号前不能有除空格之外的字符，控制流每两个“%”之间用空格隔开，不能加逗号之类的字符，否则将会被判别为非法字符，直接推出 scanf 函数。

③ 返回已读取字符数 (%n)

如下图，scanf 先将数据流中的“123456”进行数据类型转换后赋值给变量 a。然后将已经从数据流中读取过的字符数赋值给变量 b，已经读取过的字符为“1-6”六个字符外加一个空格，所以变量 b 被赋值为 7。

```
1 #include<stdio.h>
2 int main()
3 {
4     int a,b;
5     scanf("%d %n",&a,&b);
6     printf("%d %d",a,b);
7 }
```

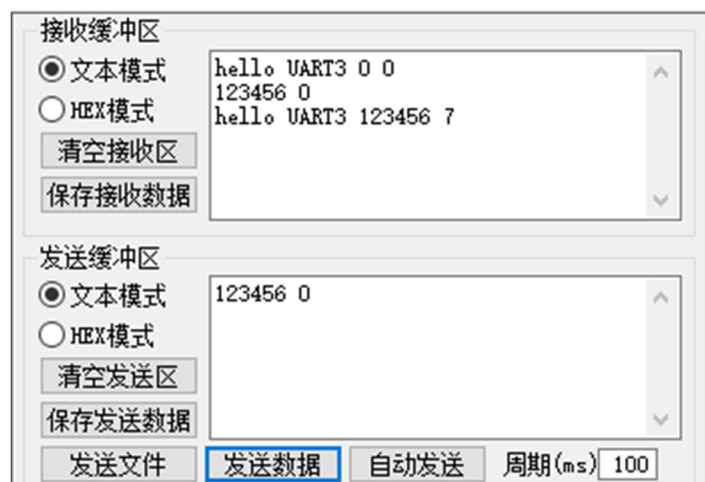
C:\Users\Administrator\Desktop\未命名1.exe

```
123456 0
123456 7
```

重定向后的返回已读取字符数测试如下：

将 main.cpp 文件下除“#define TEST_RETURN_NUMBER”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“123456 0”后按下回车，再点击发送数据。



由上图可知，123456 已经被赋值给了第一个变量，返回的已读取字符数 7 被赋值给了第二个变量。其功能和标准库下的 scanf 函数完全一致。

④ 输入十进制整数 (%d)

首先是标准库函数 scanf 对整数位数的控制。同对浮点数的位数控制一样，控制输入整数的位数。如下图中输入第一个十进制整数型变量“%2d”读取了数据流中的“12”赋值给 a，同样继续读取后续的数据进行数据类型转换时，虽然没到“%3d”位数上限 3，但是读到小数点字符，程序将其识别为不符合数据类型转换的非法字符将退出数据类型转换，所以此时变量 b 将被赋值为 3。

```
1 #include<stdio.h>
2 int main()
3 {
4     int a = 0, b = 0;
5     scanf("%2d %3d", &a, &b);
6     printf("%d %d \n", a, b);
7 }
```

C:\Users\Administrator\Desktop\未命名1.exe

123.456 123.456
12 3

其次是数据的截断。如下图，在第一个数据读入为“123”后读取到小数点字符，中断数据类型转换，将“123”赋值给第一个变量 a。接着读取后续数据流，进行第二个变量的数据类型转换，此时数据流还保存着之前的小数点字符，所以直接退出数据类型转换，没有数值赋值给变量 b，所以 b 保持初始值 0。

```
1 #include<stdio.h>
2 int main()
3 {
4     int a=0,b=0;
5     scanf("%d %d", &a, &b);
6     printf("%d %d \n", a, b);
7 }
```

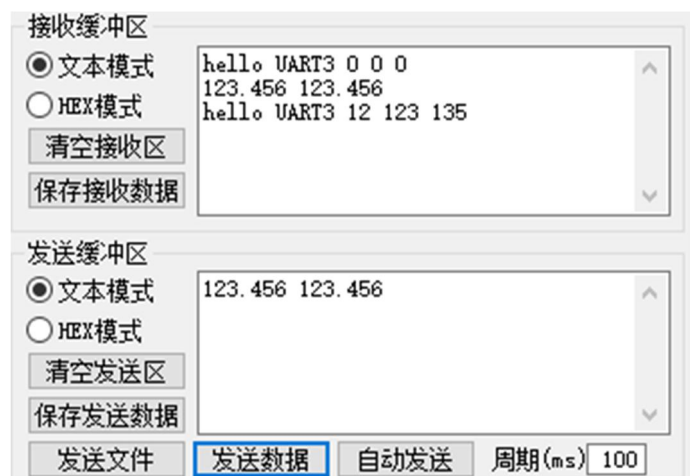
C:\Users\Administrator\Desktop\未命名1.exe

123.456 123.456
123 0

重定向后的十进制整数输入测试如下：

将 main.cpp 文件下除“#define TEST_INTEGER_D”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“123.456 123.456”后按下回车，再点击发送数据。




如上图所示，第一个数据接收两位即“123.456”中的 12，第二个数据接收第二组“123.456”中的前三位 123，计算他们求和的结果并返回 12+123=135 正确。可见对输入数据位数的控制与标准库函数相同，但是数据截断的位置不同。

串口重定向过程中，对标准库函数 scanf 做了以下优化：接收完一个整型数据后，将其后边的非法字符删除（一直到空格结束）。比如在此处，代码中定义的是整型数据，使用者误将小数输入也不会造成太大影响。

⑤ 输入八进制整数 (%o)

标准库函数 scanf 支持八进制数输入，输入八进制的“11”与“17”转换为十进制输出即 9 和 15。

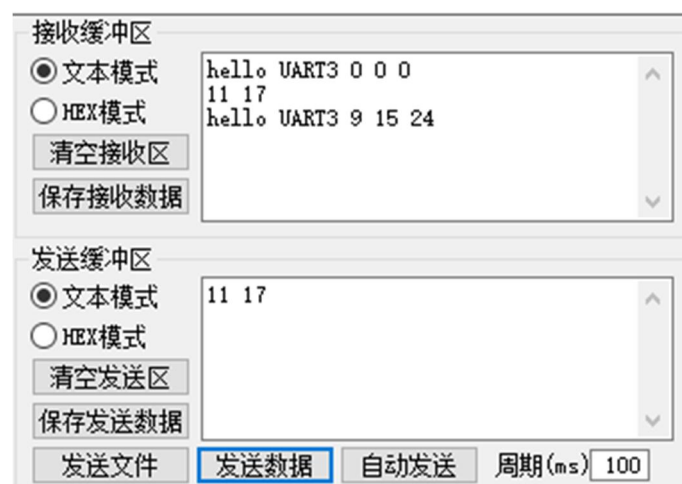
```
1  #include<stdio.h>
2  int main()
3  {
4      int a = 0, b = 0;
5      scanf("%o %o", &a, &b);
6      printf("%d %d \n", a, b);
7  }
```



重定向后的八进制整数输入测试如下：

将 main.cpp 文件下除“#define TEST_INTEGER_O”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“11 17”后按下回车，再点击发送数据。



由上图可见输入八进制数“11”与“17”成功，转换为十进制数“9”和“15”输出成功，并将其求和结果成功转化为十进制“24”计算正确。故重定向后支持八进制整数输入。

⑥ 输入十六进制整数 (%x 或者%X)

标准库函数 scanf 支持十六进制数输入，输入十六进制的“1A”与“1E”转换为十进制输出即 26 和 30。

```
1  #include<stdio.h>
2  int main()
3  {
4      int a = 0, b = 0;
5      scanf("%x %x", &a, &b);
6      printf("%d %d \n", a, b);
7  }
```

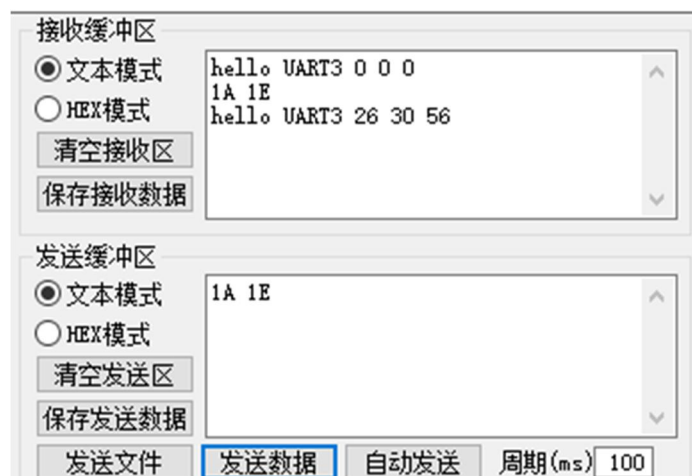
C:\Users\Administrator\Desktop\未命名1.exe

```
1A 1E
26 30
```

重定向后的十六进制整数输入测试如下：

将 main.cpp 文件下除“#define TEST_INTEGER_X”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“1A 1E”后按下回车，再点击发送数据。



由上图可见输入十六进制数“1A”与“1E”成功，转换为十进制数“26”和“30”输出成功，并将其求和结果成功转化为十进制“56”计算正确。故重定向后支持十六进制整数输入。

⑦ 输入浮点数（%e %E %f %F %g %G）

首先是标准库函数 scanf 对浮点数位数的控制。标准的 scanf 函数不能控制浮点数的小数位数，只能控制浮点数宽度，如在下图中针对第一个输入的浮点数“%4f”指定其数据宽度为 4，则在输入 12.1234 后只会读取其前四位“12.1”，将其赋值给第一个浮点型变量 a。注：读取浮点型数据时小数点也算一位。

其次是数据的截断。在第一个数据读入为“12.1”后，数据流中还剩“234 12.234”，所以程序会继续依次读入之后的数据。当读完“234”后此时才读入三个字符，没到“%5f”要求的长度上限五位，但是遇到了空格字符，此时程序会终止数据类型转换，将“234”赋值给第二个浮点型变量 b。


```
1 #include<stdio.h>
2 int main()
3 {
4     float a=0,b=0;
5     scanf("%4f %5f",&a,&b);
6     printf("%f %f \n",a,b);
7 }
```

C:\Users\Administrator\Desktop\未命名1.exe

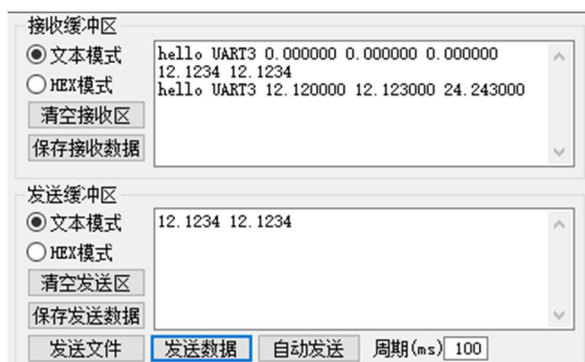
12.1234 12.1234
12.100000 234.000000

重定向后的浮点数输入测试如下：

将 main.cpp 文件下除“#define TEST_FLOAT”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“12.1234 12.1234”后按下回车，再点击发送数据。

由图中可知，重定向后的 scanf 在输入浮点数时，其使用与标准库函数并不相同。主要做了如下优化：其一不再指定输入数据的位数，而是指定精度即小数点之后的位数，使用更加方便，如“%2f”为输入精度为 0.01 的小数。其二，对数据断点做了修改，例如输入第一个数据“12.1234”虽然只读取到“12.12”赋值给第一个变量，但是他会把后边的多余数据“32”舍弃，防止对下一数据造成影响，所以输入数据为“12.12”与“12.123”，其求和结果亦正确。



⑧ 输入字符 (%c)

标准库函数 scanf 支持输入空格，如在下图中，将字符型变量 a 赋值为空格，将字符型变量 b 赋值为 s。

```
1 #include<stdio.h>
2 int main()
3 {
4     char a=' ',b='s';
5     scanf("%c %c",&a,&b);
6     printf("%c %c \n",a,b);
7 }
```

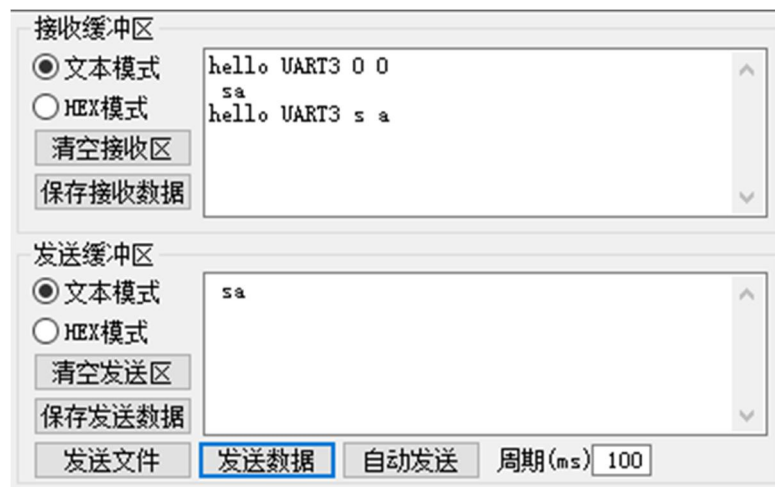
C:\Users\Administrator\Desktop\未命名1.exe

s
s

重定向后的字符输入测试如下：

将 main.cpp 文件下除“#define TEST_CHAR”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“sa”后按下回车，再点击发送数据。



由此可见，当输入字符类型数据时，重定向后不需要用空格将两个字符隔开（这点与标准库相同），但是重定向后的 scanf 会自动略过空格，只接收非空数据，所以用不用空格隔开效果都一样。

⑨ 输入字符串 (%s)

如下图所示，scanf 函数读入字符串时会在检测到非空字符时开始，检测到空格后结束。

```
1  #include<stdio.h>
2  int main()
3  {
4      char a[10],b[10];
5      scanf("%s %s",&a,&b);
6      printf("%s %s \n",a,b);
7  }
```

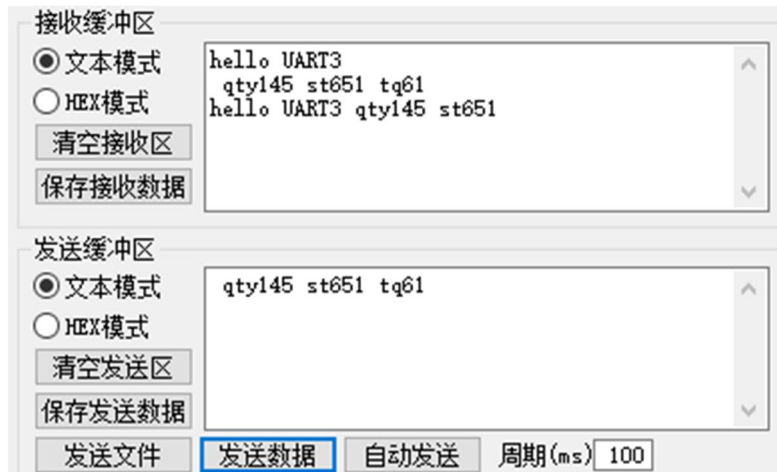
C:\Users\Administrator\Desktop\未命名1.exe

```
qty145 st651 tq61
qty145 st651
```

重定向后的字符串输入测试如下：

将 main.cpp 文件下除“#define TEST_STRING”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

在串口调试助手的发送缓冲区下输入数据“qty145 st651 tq61”后按下回车，再点击发送数据。



如上图所示，scanf 函数读入字符串时会在检测到非空字符时开始，检测到空格后结束。与标准库函数相同。

⑩ 输入指针 (%p)

输入 16 进制地址，即可将指针指向该地址下存放的对应变量。本质很简单，创建一个指针*p，利用 scanf 函数将 a 的地址赋值给 p，该作用相当于 p=&a，所以指针就指向了 a。输出的*p 即该指针指向的 a 的值。

```

1  #include<stdio.h>
2  int main()
3  {
4      int a = 5;
5      int *p = NULL;
6      printf("%p \n", &a);    //a的地址, 16进制
7      printf("%d \n", &a);    //a的地址, 10进制
8      scanf("%p", &p);        //输入16进制p的值 (a的地址), 让指针p指向a
9      printf("%p \n", p);      //输出p的值 (16进制)
10     printf("%d \n", *p);      //输出p所指向的内容
11 }

```

C:\Users\Administrator\Desktop\未命名1.exe

```

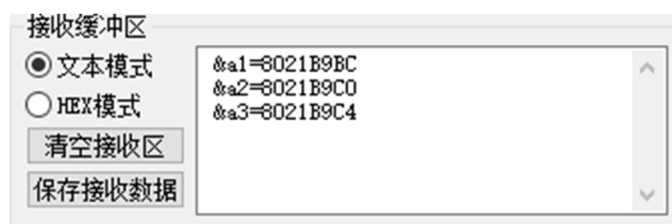
000000000062FE1C
6487580
62FE1C
000000000062FE1C
5

```

重定向后的指针输入测试如下：

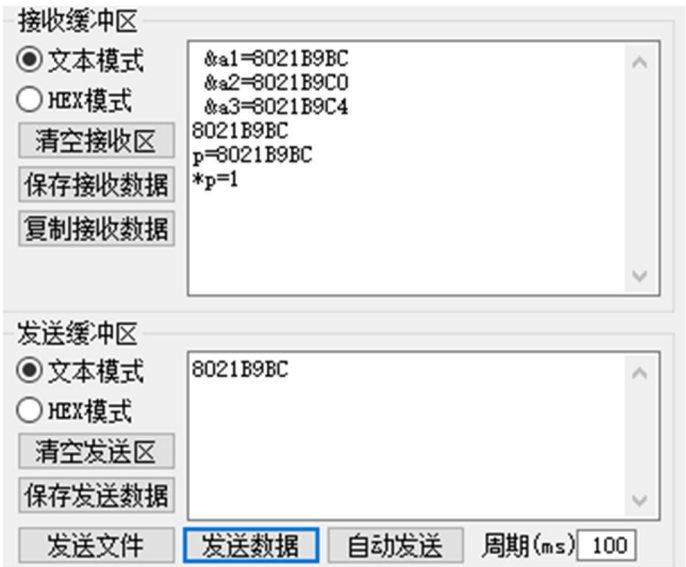
将 main.cpp 文件下除“#define TEST_STRING”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

如下图所示，首先在串口调试助手的接收缓冲区下收到三个变量 a1、a2、a3 的地址。



在串口调试助手的发送缓冲区下输入数据“（上图中 a1 的地址）”后按下回车，再点击

发送数据。



由上图中可以看到，指针 p 已经被赋值为变量 a1 的地址，指针 *p 所指向的就是变量 a1 的值。综上所述，重定向之后支持指针输入，与标准库函数用法相同。

⑪ 扫描模式输入字符串 (%[^X])

重定向后扫描模式输入字符串测试如下：

将 main.cpp 文件下除“#define TEST_STRING”之外的宏定义全部注释掉，在 loongIDE 软件上方操作菜单中点击“项目”之后再点击“编译”（快捷键为 Ctrl+F9），对源代码进行编译，在其上方操作菜单中点击“调试”之后再点击“运行”（快捷键为 F9），对源代码进行调试。

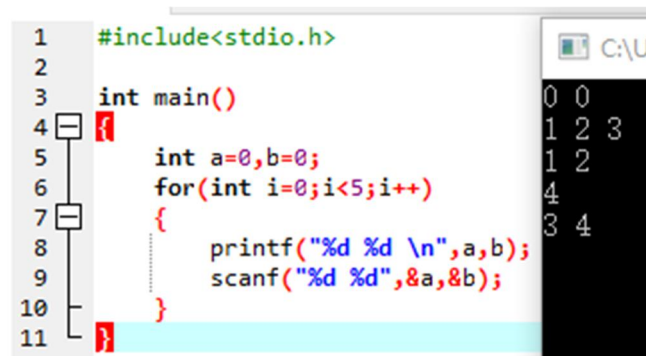
在串口调试助手的发送缓冲区下输入数据“qwerasd yui 123”后按下回车，再点击发送数据。



如上图所示，第一个字符串为“qwer”，第二个字符串为“sd yui 123”。再 main.cpp 文件下，调用 scanf 函数 scanf("%[^a] %[^]",&string1,&string2);表示第一个字符串以字符 a 结束，第二个字符串以回车换行结束。在输入第一个字符串时，当扫描到字符“a”将不再运行，第一个字符串输入结束。从字符“a”之后的字符“s”一直到回车换行均填入第二个字符串中。所以，第一个字符串为“qwer”，第二个字符串为“sd yui 123”。

⑫ 数据流的缓存

在标准库函数中，如果 scanf 控制流定义接收两个数据，实际上向数据流输入了三个数据，那么第三个数据将被放在数据缓存区，当下一次输入的时候再用。



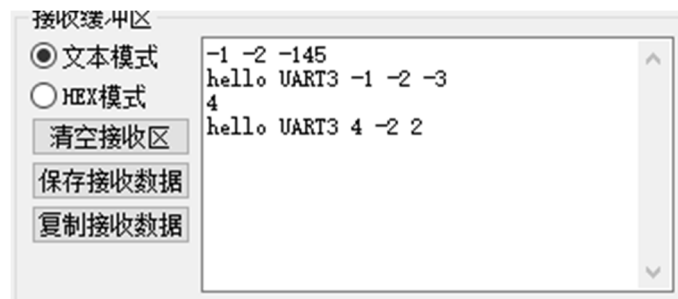
```
1 #include<stdio.h>
2
3 int main()
4 {
5     int a=0,b=0;
6     for(int i=0;i<5;i++)
7     {
8         printf("%d %d \n",a,b);
9         scanf("%d %d",&a,&b);
10    }
11 }
```

Terminal output:

```
0 0
1 2 3
1 2
4
3 4
```

如上图所示：第一次输入了三个数据 1 2 3，1 被赋值给了 a，2 被赋值给了 b，多出来的 3 被放在数据缓存区。当下一次输入数据 4 时，先读取缓存的数据 3 赋值给 a，再将新读取出来的值 4 赋值给 b。

但是串口重定向之后的 scanf 不支持缓存，会将多出来的数据舍弃掉。



如上图所示，第一次输入整数“-1 -2 -145”，只有-1 被赋值给第一个变量，-2 被赋值给第二个变量，计算 $-1 + (-2) = -3$ 并输出。由此可见重定向之后支持负数输入。然而多出来的数据-145 被舍弃，下次输入 4 时，4 被赋值给了第一个变量，第二个变量没被赋值所以保持原状为-2，计算 $4 + (-2) = 2$ 显示输出。

4、反思总结

scanf()函数的重定向的代码结构可以进一步优化，在此提出了几个可以优化的方向：

- ① 将输入整数与浮点数的符号标志位并入 flag 中。
- ② 将_doscan 中定义的变量设为全局变量，之后就可将重复率很高的功能块封装为函数，使代码变得更加简洁。
- ③ 在_doscan 中定义一个指向数据缓存区的指针，用该指针代替 nrchars 与 ic。

Start here x 123.c x

```
1 #include<stdio.h>
2
3 int change(const char *c)
4 {
5     c = c+3 ;
6     return c;
7 }
8
9 int main()
10 {
11     char a[4]={'1','2','3','4'};
12     const char *p;
13     p = &a;
14     printf("*p = %c \n",*p);
15     p = change(p); //p = p + 3;
16     printf("*p = %c \n",*p);
17 }
18
```

C:\Users\Administrator\Desktop\123.exe

```
*p = 1
*p = 4

Process returned 8 (0x8)   execution time : 0.054 s
Press any key to continue.
```