LAB1 - 数据表示与操作

- 截止时间：见课程管理系统上的相应作业发布信息。
- 请在实验截止前务必确认提交的文件符合命名、格式等要求，并使用实验数据附带的测试工具确认提交结果的正确性。建议在提交后再下载提交的内容确认无误。如果由于提交了不符合要求的实验结果而导致批改程序扣分或无分，责任自负。

简介

本实验有助于更好地熟悉和掌握计算机中整数和浮点数的二进制编码表示及其操作。实验中，你需要完成一系列编程“难题”——使用有限类型和数量的运算操作实现一组给定功能的函数。

- 实验语言：C
- 实验环境：Linux

实验数据

在本实验中，每位学生可获得一个特定于其学号且包含以下实验数据内容的tar文件（可通过实验发布页面中的链接下载），将该文件下载到实验Linux系统的本地目录中并使用“tar xf xxxxxxxx.tar”命令提取出其中包含的实验文件：

- README —— 有关实验细节的说明文件，请在开始实验前仔细阅读
- bits.c —— 包含一组用于完成指定功能的函数的代码框架，需要你按要求补充完成其函数体代码并“作为实验结果提交”。函数的功能与实现要求详细说明在相应函数和文件首部的注释中（务必认真阅读和遵照说明完成实验）。
- bits.h —— 头文件
- btest.c —— 实验结果测试工具，用于检查作为实验结果的 bits.c中函数实现是否满足实验的功能正确性要求。
- btest.h, decl.c, tests.c —— 生成btest程序的源文件
- dlc —— 实验结果检查工具，用于判断作为实验结果的 bits.c中函数实现是否满足实验的语法规则要求。
- Makefile —— 生成btest、fshow、ishow等工具的Make文件。
- ishow.c —— 整型数据表示查看工具
- fshow.c —— 浮点数据表示查看工具

实验要求

实验前请认真阅读本文档和bits.c中的代码及注释（特别是bits.c前部的编程说明与示例），然后根据要求相应完成bits.c中的各目标函数的实现代码。

你的函数实现代码必须满足如下要求（以及bits.c前部说明中的相关要求）：

1. 基本要求（适用所有函数）：
    - 只能使用有限类型和数量的C语言算术、逻辑操作，具体要求见每个函数前的注释说明中的“Legal ops”（允许使用的操作符）和“Max ops”（允许使用的最多操作符数量）。
    - 不得使用任何形式的强制类型转换。
    - 不得使用除整型外的任何其它数据类型，如数组、结构、联合等。
    - 不得定义和使用宏。
    - 不得定义除已给定的框架函数外的其他函数，不得调用任何函数。
2. 非浮点数函数要求：
    - 只能使用顺序程序结构（即不得使用if、do、while、for、switch等循环或条件分支控制程序结构。

    - 不得使用超过8位表示的常量（即其值必须位于[0,255]中）。
3. 浮点数函数要求：
    - 可以使用循环和条件控制；
    - 可以使用整型和无符号整型常量及变量（取值不受[0,255]限制）；
    - 不得使用任何浮点数据类型、操作及常量。

上述实验要求的主要目的是使得你必须从二进制位的角度考虑数据，进而更清楚地理解数据的二进制表示。

实验结果提交

请将包含函数实现代码的bits.c重命名为“你的学号-bits.c”（注意文件名中所有字符使用小写），并作为实验结果提交。

注意：提交前务必按照本文档后面有关**"**检查你的代码**"**的说明，核实**bits.c**可正确编译且实现了各包含函数功能后再提交。

实验内容

你需要使用C语言编程完成bits.c中的一系列目标函数（已给出函数框架但未给出实现）的功能。 注意：

- 具体的函数功能和实现要求请参看bits.c中各目标函数前的注释说明。
- 关于浮点数的函数均使用unsigned型数据表示浮点数据。

函数示例

```
/*
 * bitAnd - x&y using only ~ and |
 *   Example: bitAnd(6, 5) = 4
 *   Legal ops: ~ |
 *   Max ops: 8
 *   Rating: 1
 */
int bitAnd(int x, int y) {
  return 2;
}
```

函数前的注释部分描述了函数的功能目标和实现要求。其中：

- 函数名后的文字给出了函数需要实现的输出（即功能）
- "Example”指出函数（被评分/测试程序）调用的示例
- "Legal ops”指出你的函数实现允许使用的C语言操作符
- "Max ops”指出你的函数实现中允许使用的操作符（如|和~）的最大数量
- "Rating”指出各函数的难度等级（对应于该函数的实验分值）

你的任务是将函数体中的“return 2;”替换为你用以实现函数功能的代码。 你也可参考tests.c中对应的测试函数来了解所需实现的功能，但是注意这些测试函数并不满足目标函数必须遵循的编码约束条件，只能用做关于目标函数正确行为的参考。

检查提交代码【重要！】

如前所述，实验数据包中包含两个工具程序可帮助检查你的代码的正确性。因此，在提交实验结果前务必使用该工具确认结果正确。

使用**dlc**检查函数实现代码是否符合实验要求中的编码规则

完成bits.c后，调用如下命令进行检查：

```
$ ./dlc bits.c
```

dlc将返回错误信息如果它发现了错误，例如不允许使用的操作符、过多数量的操作符或者非顺序的代码结构。如果程序代码满足规则要求，dlc将不输出任何提示。 使用-e选项调用dlc

```
$ ./dlc -e bits.c
```

可使dlc打印出每个函数使用的操作符数量。 输入“ ./dlc -help”可打印出dlc的可用命令行选项列表。

使用**btest**检查函数实现代码的功能正确性

首先使用如下命令编译生成btest可执行程序：

```
$ make
```

如下调用btest命令检查bits.c中所有函数的功能正确性：

```
$ ./btest
```

注意每次修改bits.c后都必须使用make命令重新编译生成btest程序。 为方便依次检查测试每一函数的正确性，可如下在命令行使用“-f”选项跟上函数名，以要求btest只测试所指定的函数：

```
$ ./btest -f bitXor
```

进一步可如下使用“-1, -2, -3”等选项在函数名后输入特定的函数参数：

```
$ ./btest -f bitXor -1 7 -2 0xf
```

（README文件中有关于btest程序的使用说明）

建议与提示【重要！】

- 在bits.c文件中不要包含stdio.h等头文件，因为这样将使dlc程序无法正常运行并产生一些难以理解的错误信息。注意尽管未包含stdio.h头文件，你仍然可以在bits.c中调用printf函数进行调试（但在最终提交前务必删除这些函数调用），gcc将打印警告信息但你可以忽略它们。
- 注意dlc程序使用比gcc和C++更严格的C变量声明形式。在由“{}”包围的一个代码块中，所有变量声明必须出现在任何非声明语句之前。例如，针对下述代码，dlc将报错：

  ```
  int foo(int x)
  {
      int a = x;
      a *= 3;     /* Statement that is not a declaration */
      int b = a;  /* ERROR: Declaration not allowed here */
  }
  ```

  你必须类似如下代码将变量声明放在最前：

  ```
  int foo(int x)
  {
      int a = x;
      int b;
      a *= 3;
      b = a;
  }
  ```