



中山大學

SUN YAT-SEN UNIVERSITY

程序控制结构进阶与练习

學大山中立國

中山大学计算机学院



讲课人: 陈武辉

目录

CONTENTS

01

实数比较

02

逻辑运算短路

03

位掩码 (**bitMASK**)

04

调试-分支语句

05

调试-循环语句

实数比较

计算机中连续的实数用 float 或 double 表示。由于离散表示，使得变量中存储的计算结果很多时候是实际实数的近似值。

因此，两个实数变量不能使用 == 或 != 直接比较，或判断实数变量为零。特别是循环语句，循环条件表示式中使用类似 ($x==y$) 这样的比较，可能导致不确定的死循环（有时能循环中止，有时死循环）
实战中，实数比较运算一般仅使用 > 和 < 两个比较运算符。

两个实数相等使用 $\text{fabs}(x-y) < \text{eps}$ ， eps 是合适的小数如 $1e-7$

实数等于零使用 $\text{fabs}(x) < \text{eps}$ ， eps 又称精度控制量（值）

实数比较 – 案例研究

计算 PI，精确到小数点后5位。PI的计算公式如下：

$$\pi = 4 * \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots + \frac{1}{4n-3} - \frac{1}{4n-1} \right)$$

```
/*pi*/
pi = 0;
int i = 1;
const double eps = 1e-6; // 精度保证5位
while ( 1.0/i > eps) { // 不合适用 for 语句
    pi += 1.0/i;
    i++;i++;
    pi -= 1.0/i;
    i++;i++;
}
pi *= 4;
printf("%.5f\n",pi);
```

请打开pi.c

逻辑运算短路

表达式 $a \&& b$ 如果子表达式 a 为 `false` 则子表达式 b 结果无论是什么，结果必然为 `false`。为了优化程序执行速度， b 不求值，称为 **and** 短路。

表达式 $a || b$ 如果子表达式 a 为 `true` 则子表达式 b 不求值，结果必然为 `true`。称为 **or** 短路。

例如：

```
int a,b,c,d;  
a = b = c = 0;  
d = a++ || ++b || c++;  
printf("%d%d%d%d",a,b,c,d);
```

因此，在 **and** 表达式中把大概率为 `false` 的子表达式写在左边有利于提升程序性能。同样 **or** 表达式把大概率为 `true` 的子表达式写在左边。

逻辑运算短路

有 int 变量 a 和 b，请问下面的语句是什么意思

`b && (c=(double)a/b);`

等价于：

```
if (!b) result=false;
else result=(c=(double)a/b);
```

因此，逻辑运算短路等价于 if ... else if ... 多路分支语句。例如

`e1||e2||e3||...||en` 等价于

```
if (e1) result=true;
else if (e2) result=true;
else if (e3) result=true;
...
else result=(en);
```

有副作用 (Side effects) 的表达式

副作用: 如果一个表达式不仅算出一个值，还修改了其他变量，就说这个表达式有副作用。例如：**自增减运算、赋值运算、传引用函数**

有副作用的表达式并不一定有害，但以下情况会导致表达式难以被理解、测试，甚至被不同编译器解释为不同语义的代码。

- `&&`、`||`、`?:` 表达式包含有副作用的子表达式。例如 `a || i++`
- 子表达式之间因副作用必须用特定顺序编译，会导致**未定义行为**。例如：
 - `i++ +i++; b*2 + 5*(1+b++)`
 - `a[i++]=i; f(&i) + i`
- 函数调用参数中包含有副作用表达式。例如 `f(c=d,e++)`

可能出现在理论考试中

理论考试，实际编程，
都不该出现

bitMASK位操作与应用案例

复习：

- & 可以清零指定的位
- | 可以设置指定位
- ^ 可以使指定的位反相

案例用 char 表示文件的权限，对应位为1表示有权限

- 第一位表示可读
- 第二位表示可写
- 第三位表示可执行

请阅读 file-permission.c 解释每一个语句的含义

调试 Debug) 与错误 (Error)

调试 **Debug**) 又称除错，是发现和减少计算机程序错误的过程。

程序错误的种类：

- 语法（编译）错误，如语句结束缺少“；”号。这类错误编译器会给出提示
- 运行时错误，程序运行时异常退出，给出或不给错误信息。如段错误
- 逻辑错误，程序正常运行，但未得到期望的结果。

调试，是发现后两类错误的重要手段

调试 (debug) 方法

调试方法一：

Where
?

What?

Which
?

- 使用 printf 函数在指定位置打印路径相关信息或变量值

调试方法二：

- 工具，如 dev-c++ (gdb)
- 语句跟踪，使用断点（break point）或执行下一句等指令跟踪程序执行路径
- 变量查看，在跟踪的过程中观察变量的变化

因此，需要掌握调试工具的使用以及调试的技巧

调试 (debug) 过程

调试前：

- 调试工具准备
- 设计测试案例
- 修改测试程序

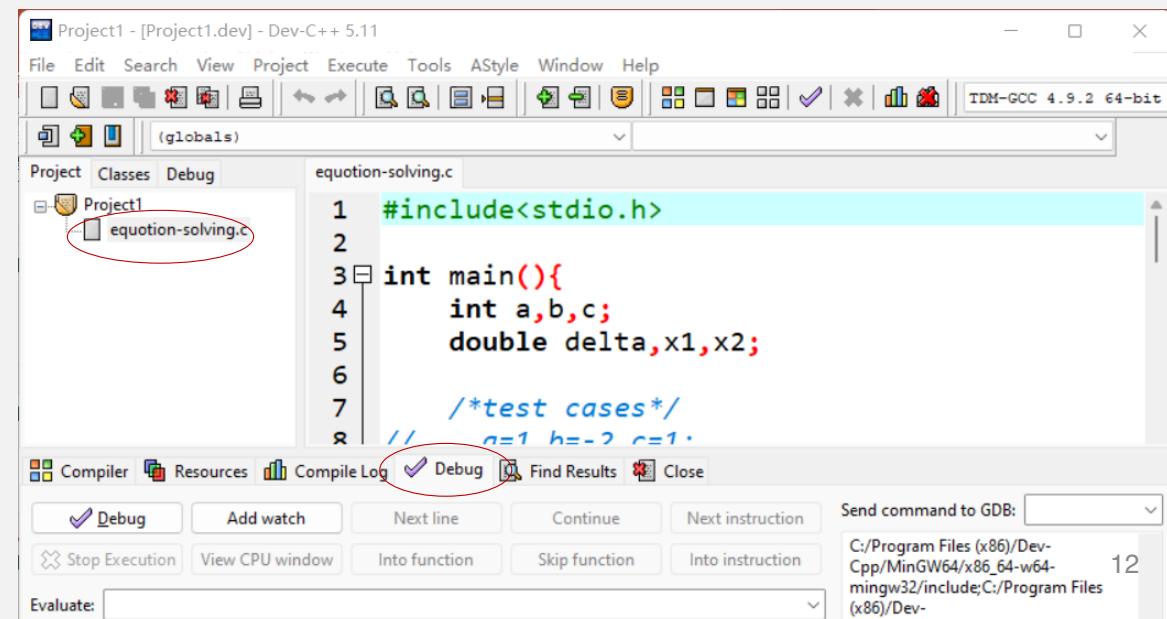
调试中：

- 断点选择
- 变量查看
- 常用调试命令

调试 debug)- 工具 dev C++

调试须知：必须先创建项目再调试

- 创建一个项目 (project) ，选择控制台应用 (console application)
- 将要调试 c 代码添加到该项目
- 点击代码，打开代码
- 修改代码
- 按 F5 键开启调试



调试(debug)- 代码修改

调试程序一般没有输入语句，如 `scanf`

调试程序四段结构

- 变量申明或定义
- 测试案例
- 结果计算
- 输出结果

程序四段结构

- 变量申明或定义
- 数据输入
- 结果计算
- 输出结果

常用快捷键：`ctrl+ /` 触发注释代码，在测试案例或数据输入间切换

调试 debug)- 分支语句测试案例设计

测试案例是一组输入变量，它能覆盖所有可能的分支。例如

求方程 $ax^2+bx+c=0$ 的实数解。

输入：三个整数 a, b, c ， $|a|, |b|, |c| \in [0, 1000]$ 。

考 a, Δ 是否为零情况

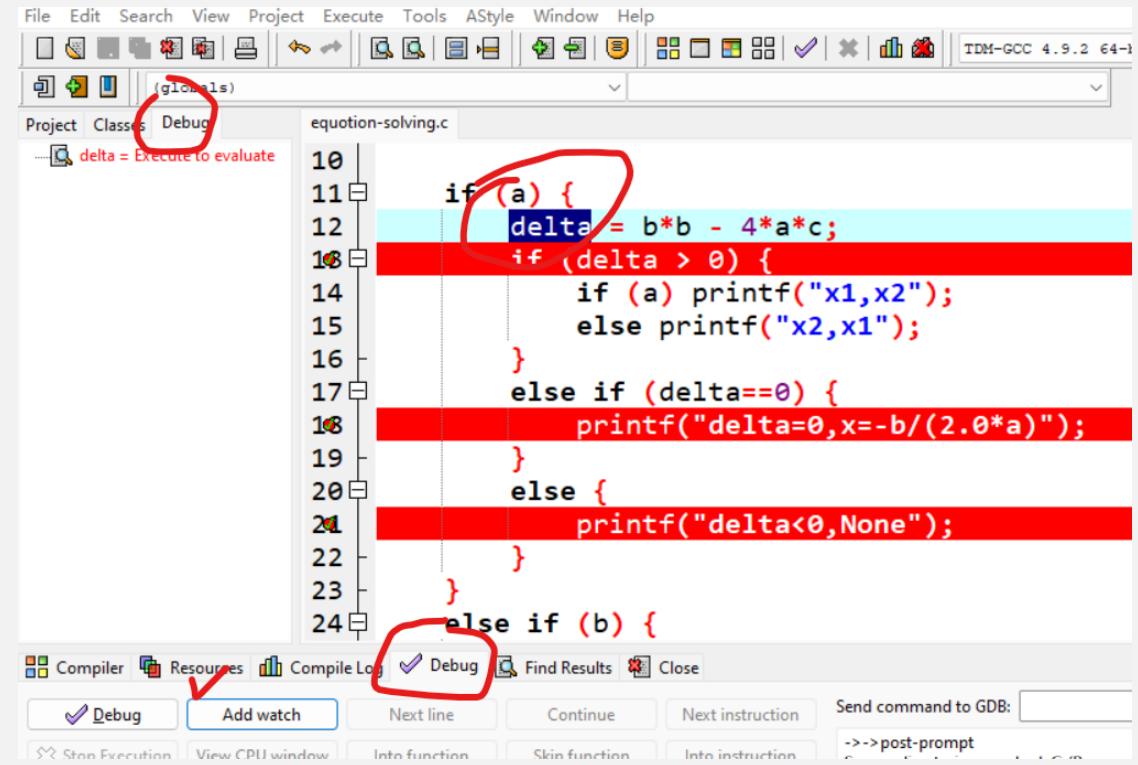
- 左边是分析
- 红色每行是一个案例
- 通常是最典型的输入
- 易于观察各变量值

a	delta	b	a	b	c	期望输出
$\neq 0$	>0		1	0	-4	-2, 2
	$=0$		1	-2	1	1
	<0		1	2	2	None
$=0$	---	$\neq 0$	0	1	1	1
	---	0	0	0	1	None

调试 debug)- 分支语句案例

调试步骤：

- 创建一个新项目
- 加入案例代码，equation-solving.c
- 设置断点：点击 13，18，21行行号，这些行变红
- 查看变量：选中变量 delta，按按钮 Add watch
- 按 F5



The screenshot shows a debugger interface with a C code editor. The code is as follows:

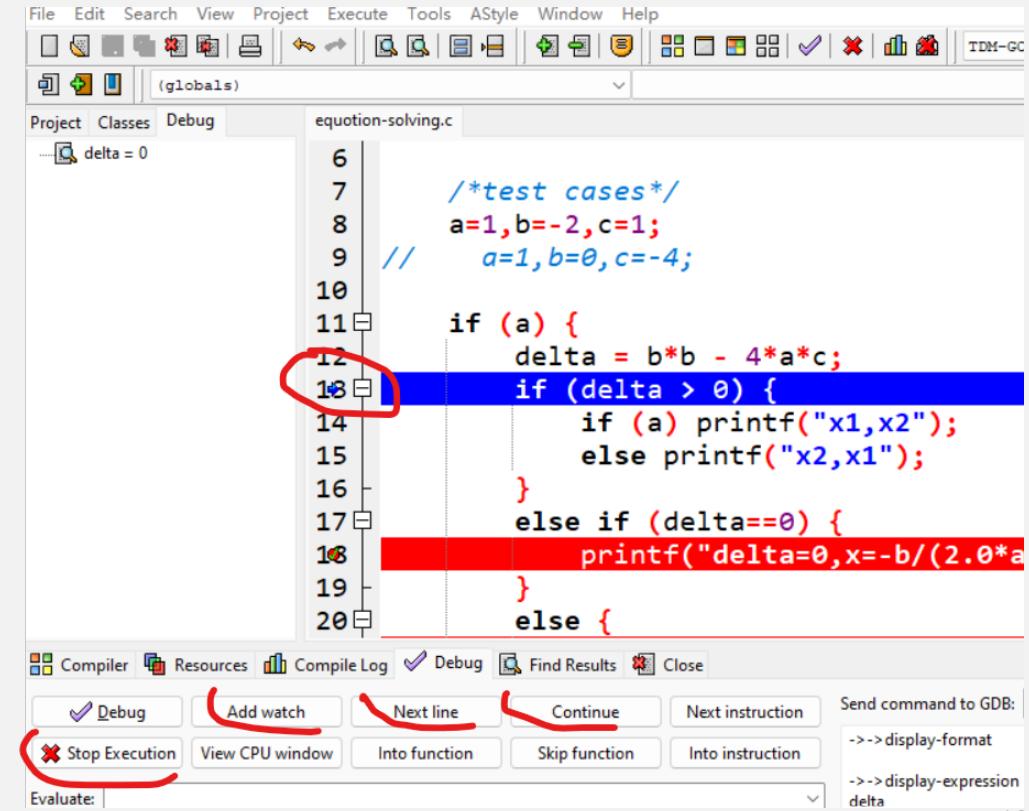
```
10 if (a) {  
11     delta = b*b - 4*a*c;  
12     if (delta > 0) {  
13         if (a) printf("x1,x2");  
14         else printf("x2,x1");  
15     }  
16     else if (delta==0) {  
17         printf("delta=0,x=-b/(2.0*a)");  
18     }  
19     else {  
20         printf("delta<0,None");  
21     }  
22 }  
23 else if (b) {  
24 }
```

Several lines of code are highlighted in red: lines 12, 13, 18, 21, and 23. A red circle highlights the "Debug" tab in the toolbar at the bottom. A red arrow points to the "Add watch" button in the same toolbar.

调试 debug)- 分支语句案例

调试步骤：

- 这时程序会停在第一个断点（蓝色行）
- 选中任何变量，点 Add watch 观察变量值
- 点击行号，添加/去除断点
- Next line 执行下一条语句
- Continue 执行到下一个断点或程序结束
- Stop Execution 调试结束



The screenshot shows the TDM-GDB debugger interface. The main window displays a C program named 'equation-solving.c'. A breakpoint is set on line 13, which contains the code: `if (delta > 0) {`. This line is highlighted in blue. The line number 13 is circled in red. The 'Stop Execution' button in the toolbar at the bottom is also circled in red.

```
File Edit Search View Project Execute Tools AStyle Window Help
File Project Classes Debug equation-solving.c
(globals)
Project delta = 0
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
/*test cases*/
a=1,b=-2,c=1;
// a=1,b=0,c=-4;
if (a) {
    delta = b*b - 4*a*c;
    if (delta > 0) {
        if (a) printf("x1,x2");
        else printf("x2,x1");
    }
    else if (delta==0) {
        printf("delta=0,x=-b/(2.0*a");
    }
}
else {
```

Compiler Resources Compile Log Debug Find Results Close
Debug Add watch Next line Continue Next instruction
Stop Execution View CPU window Into function Skip function Into instruction
Send command to GDB:
->-> display-format
->-> display-expression
Evaluate: delta

调试 debug)- 分支语句练习

任务一：记录执行路径

- 断点设在11行
- 用 next line 执行到函数结束
- 记录执行完后经过的行号

任务二：拦截所有可能

- 断点设置在 if 或 else 后一行，思考为什么不要把语句与if，else写在一行上
- 修改测试案例，如 1 2 2，观察执行情况

任务三：检测分支

- 断点设置在 13 行
- 按 next line ，请分析这个案例下 if 条件的结果

调试 debug)- 分支语句练习

任务四：设计测试案例

- 请给出 $a \&& delta > 0$ 时的两个测试案例
- 它们确保打印 $x1, x2$ 的情况

任务五：完善该程序代码

- 分别用设计的 5 个案例检测程序
- 调试并得到期望结果
- 最后用 `ctrl+/` 屏蔽调试代码，给出数据输入代码

调试 debug)- 循环语句案例

用牛顿法计算平方根

- 代码 newton.c 来源于网上，经过适当修改。

测试案例设计

- 循环案例设计较复 ，不要求掌握，作业题中都会给出。这里是 2 的平方根

断点选择和 watch 变量

- 循环调试断点一般选择3处：while，for 关键字所在行，
- 循环体的第一行，循环语句的下一行
- Watch 是 条件表达式、关键变量
- 使用 Next line 跟踪一次迭代
- 使用 Continue 跟踪每个迭代关键变量的变化，观察迭代是否趋向结束（收敛）

调试 debug)- 循环语句调试练习

任务一：使用 next line 观察第一次迭代

任务二： 使用 Continue 记录每次迭代后 x_0 的变化，3-5个值

任务三： 找到导致死循环的 bug

任务四：完善程序，使它能计算你输入的数

课堂练习

剪刀石头布，对战计算机。采用 $2n+1$ 轮决定胜负：

- 首先输入 n 决定比赛轮数，如 $n=1$ 则表示3轮2胜制
- 每轮规则如下：
 - 计算机随机产生0-2数
 - 玩家输入 0, 1, 2 分别表示剪刀、石头和布
 - 石头砸剪刀、布包石头、剪刀裁布则玩家胜本轮，否则计算机胜
 - 输出本轮胜者，累计获胜次数
 - 最先达到胜 $(n+1)$ 次者获得最终胜利

请用 C 语言实现该游戏



中山大學

SUN YAT-SEN UNIVERSITY

谢谢

國立中山大學

中山大学计算机学院



编制人: 课题组