



第三章 程序设计基础

模块3.2：处理数据

主讲教师：同济大学计算机科学与技术学院 陈宇飞
同济大学计算机科学与技术学院 龚晓亮



目录

- 如何debug
- 浮点数



1. 如何debug

- 参考文件 《VS_debug入门指南》
- 结合学生程序案例bmi. cpp演示



2. 浮点数

作业题**计算BMI**：测试用例160 50的时候，19.53125这个结果能够被精确表示，但是出现错误结果的原因主要有两个：

- ① 运算过程中有一些**中间结果产生了误差**，导致最终结果出现误差，这种应该是占比偏多的
- ② 全程无误差地计算结果，最终精确地表示出了19.53125。但是输出的时候由于系统使用的是**银行家舍入方法**，和普通的四舍五入不同（样例按照四舍五入来的），导致输出结果与样例结果不同



zz

You

讲讲银行家舍入规则



ChatGPT

银行家舍入规则（Banker's Rounding），也称为“四舍六入五成双”规则，是一种在数学、会计和计算机科学中常用的四舍五入方法。与传统的四舍五入规则（即，小于5时舍去，大于等于5时进位）相比，银行家舍入提供了一种更加公平和统计偏差较小的舍入方式。其具体规则如下：

1. **四舍六入**：当要舍入的数字小于5时，向下舍入；大于5时，向上舍入。
2. **五成双**：当要舍入的数字正好是5，并且5后面没有其他数字（即精确到5），或者5后面的所有数字都是0时，舍入结果会使得前一位数字变成偶数。也就是说，如果前一位是奇数，则向上舍入使其成为偶数；如果前一位已经是偶数，则保持不变（向下舍入）。

这种舍入方式的优势在于，当大量的数被舍入时，向上和向下舍入的情况会更加平衡，从而减少了系统性的舍入误差。这在金融和会计领域尤其重要，因为它有助于在大量计算中保持数值的准确性和公平性。

例如，根据银行家舍入规则：

- `2.5` 会被舍入为 `2`（因为2是更近的偶数）。
- `3.5` 会被舍入为 `4`（使3成为偶数）。
- `2.6` 会被舍入为 `3`（标准的四舍五入）。
- `2.4` 会被舍入为 `2`（标准的四舍五入）。

这种舍入方法在一些编程语言和库中被默认采用，尤其是在处理金融数据时。



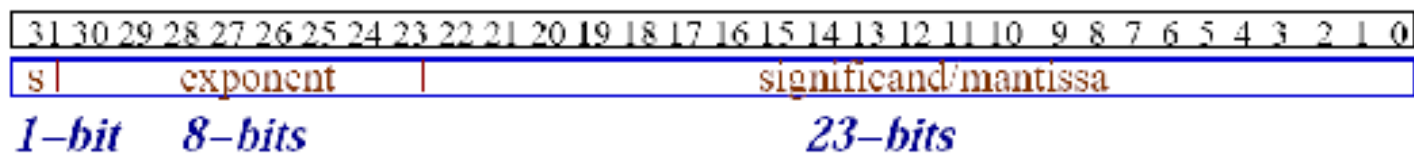


2. 浮点数

PDS: CS 11002

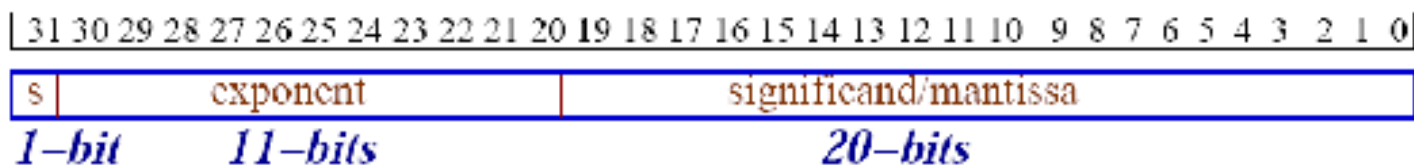
Computer Sc. & Engg: IIT Kharagpur

IEEE 754标准



Single Precession (32-bit)

<http://blog.csdn.net/xishadon>



Double Precession (64 bit)



浮点数参与关系运算时要考虑误差

例1: float f=100.25;

32bit机内表示 0100 0010 1100 1000 1000 0000 0000 0000

尾数是 100 1000 1000 0000 0000 0000

尾数转换为十进制小数是 0.56640625

尾数表示的十进制小数是 1.56640625 (加整数部分的1后)

$1.56640625 \times 2^6 = 100.25$ (未体现出误差)

例2: float f = 1234567.7654321;

32bit机内表示 0100 1001 1001 0110 1011 0100 0011 1110

尾数是 001 0110 1011 0100 0011 1110

尾数转换为十进制小数形式是 0.1773755503845214844

尾数表示的十进制小数形式是 1.1773755503845214844 (加1后)

$1.1773755503845214844 * 2^{20} = 1234567.75$ (体现出误差)



```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      float f1 = 100.25;
6      cout << (f1 - 100.25) << endl;
7      cout << (f1 == 100.25) << endl;
8
9      float f2 = 1234567.7654321;
10     cout << (f2 - 1234567.7654321) << endl;
11     cout << (f2 == 1234567.7654321) << endl;
12     return 0;
13 }
```

```
Microsoft Visual Studio 调试控制台
0
1
-0.0154321
0
```

demo.cpp(9,28): warning C4305: “初始化”: 从“double”到“float”截断

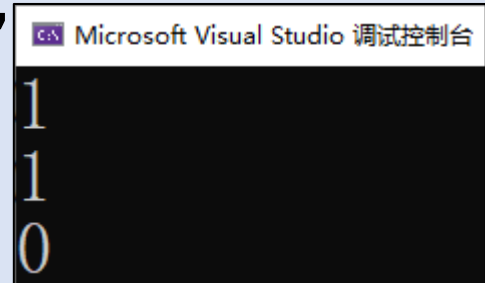


```
#include <iostream>
#include <cmath> //fabs需要, VS可不加
using namespace std;
int main()
{
    float f1=100.25;
    cout << (fabs(f1-100.25) < 1e-6) << endl;

    float f2=1234567.7654321;
    cout << (fabs(f2-1234567.7654321) < 1e-1) << endl;
    cout << (fabs(f2-1234567.7654321) < 1e-2) << endl;
    return 0;
}
```

fabs是求绝对值的系统函数

通用方法：当两数相减的绝对值小于某个值则认为相等





```
int main()
{
```

```
    float b = 1.1; //有warning
```

```
    cout << (b - 1.1) << endl;
```

2.38419e-08

```
    cout << (b == 1.1) << endl;
```

0

```
    cout << (fabs(b - 1.1) < 1e-6) << endl;
```

1

```
    float c = 1.0; //无warning
```

```
    cout << (c - 1.0) << endl;
```

0

```
    cout << (c == 1.0) << endl;
```

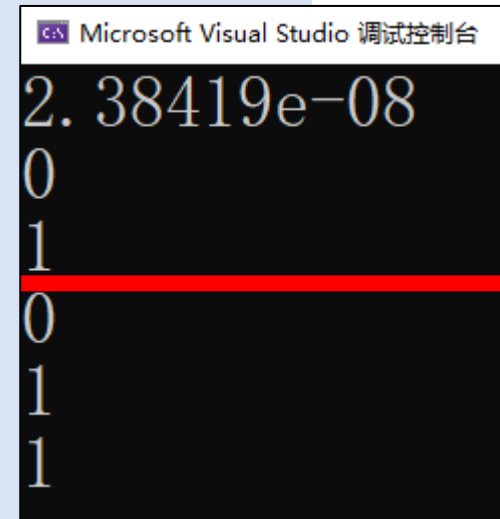
1

```
    cout << (fabs(c - 1.0) < 1e-6) << endl;
```

1

```
    return 0;
```

```
}
```





```
#include <iostream>
#include <cmath> //VS可不加
using namespace std;
int main()
{ double d1=123.456789012345678;
  double d2=123.456789123456789;
  cout << (d1==d2) << endl;
  cout << (fabs(d1-d2)<1e-6) << endl;
  cout << (fabs(d1-d2)<1e-7) << endl;

  float f1=123.456789012345678;
  float f2=123.456789123456789;
  cout << (f1==f2) << endl;
  cout << (fabs(f1-f2)<1e-6) << endl;
  cout << (fabs(f1-f2)<1e-7) << endl;

  return 0;
} //VS有两个warning
```

warning C4305: “初始化”: 从“double”到“float”截断
warning C4305: “初始化”: 从“double”到“float”截断

0
1
0

1
1
1

Microsoft
0
1
0
1
1
1



```
#include <iostream>
#include <iomanip>
#include <cmath> //VS可不加
using namespace std;
int main()
{
```

```
    cout << setiosflags(ios::fixed); //指定fixed输出
    double d1 = 123.456789012345678;
    cout << setprecision(15) << d1 << endl;
    cout << setprecision(20) << d1 << endl << endl;
    double d2 = 123.456789123456789;
    cout << setprecision(15) << d2 << endl;
    cout << setprecision(20) << d2 << endl << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
123.456789012345681
123.45678901234568058953

123.456789123456787
123.45678912345678668316
```



```
#include <iostream>
#include <iomanip>
#include <cmath> //VS可不加
using namespace std;
int main()
{
    cout << setiosflags(ios::fixed); //指定fixed输出
    float f1 = 123.456789012345678;
    cout << setprecision(15) << f1 << endl;
    cout << setprecision(20) << f1 << endl << endl;
    float f2 = 123.456789123456789;
    cout << setprecision(15) << f2 << endl;
    cout << setprecision(20) << f2 << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
123.456787109375000
123.45678710937500000000
```

```
123.456787109375000
123.45678710937500000000
```



2. 浮点数

- ✓浮点数的运算伴随误差，要善于利用debug工具排查问题；
- ✓float和double的存储精度有差异，根据实际情况选用；
- ✓浮点数参与关系运算时要考虑误差：用==判断浮点数是否相等，某些情况下可能与预期结果不符合，因此禁用；
- ✓fabs()函数是通用的避免浮点数误差的方法，使用时不要超过有效位数限定；
- ✓浮点数输出时可以任意指定位数，但超过有效位数限定的值不可信