数值分析第 1 次上机作业

学号：221840189，姓名：王晨光

§1 问题一

1.1 问题

编程以二进制的形式显示单精度或双精度浮点数.

1.2 算法思路

二进制形式的单双精度浮点数表示分为三部分：符号位，指数，尾数. 通过它们的定义，我们可以逐个确定下来. 其中单精度浮点数通过一共 32 位二进制数表示，计算指数部分时偏置为 127；双精度浮点数通过一共 64 位浮点数表示，计算指数部分时偏置为 1023.

Algorithm 1 浮点数的二进制表示

Require: 一个十进制浮点数 a.
Ensure: 浮点数的单精度与双精度的二进制表示字符串 s_1,s_2，或失败（超界）信息.

function TRANSFORM(a)
  if a 超出单精度范围 **then**
    输出“超出单精度范围”
  else
    $float$ $a_1 \leftarrow a$
    if $a_1 < 0$ **then**
      符号位为 1，$a_1 = |a_1|$
    else
      符号位为 0
    end if
    分离 a_1 的整数部分 $\bar{a_1}$ 与小数部分 $a_1 - \bar{a_1}$
    转换 $\bar{a_1}$,$a_1 - \bar{a_1}$ 为二进制
    指数部分为 $length(\bar{a_1}) - 1 + 127$ 的二进制表示
    尾数部分为 $\bar{a_1}$ 去掉首位后与 $a_1 - \bar{a_1}$ 合并
    按顺序输出符号位，指数，尾数，得到二进制表示的单精度浮点数
  end if
  if a 超出双精度范围 **then**
    输出“超出双精度范围”
  else
    $double$ $a_2 \leftarrow a$
    if $a_1 < 0$ **then**
      符号位为 1，$a_2 = |a_2|$
    else
      符号位为 0
    end if
    分离 a_2 的整数部分 $\bar{a_2}$ 与小数部分 $a_2 - \bar{a_2}$
    转换 $\bar{a_2}$,$a_2 - \bar{a_2}$ 为二进制
    指数部分为 $length(\bar{a_2}) - 1 + 1023$ 的二进制表示
    尾数部分为 $\bar{a_2}$ 去掉首位后与 $a_2 - \bar{a_2}$ 合并
    按顺序输出符号位，指数，尾数，得到二进制表示的双精度浮点数
  end if
end function

1.3 结果分析

测试程序，得到 123.456 的单精度浮点数二进制表示为：
01000010 11110110 11101001 01111001
双精度为：
01000000 01011110 11011101 00101111 00011010 10011111 10111110 01110111
经检验，结果正确无误.

§2 问题二

2.1 问题

编程找出单精度或双精度浮点数的机器精度、下溢值和上溢值并与理论值比较。(非规格化浮点数)

2.2 算法思路

以单精度浮点数为例，双精度浮点数同理.

1. 机器精度

在计算机浮点系统中，一个浮点数的表达形式是 $fl(x) = x(1+\varepsilon), |\varepsilon| \leqslant \varepsilon_{mach}$，其中 ε_{mach} 是机器精度，它说明了用浮点系统表示一个非零实数 x 的最大可能相对误差，即 $|(fl(x)-x)/x \leqslant \varepsilon_{mach}|$. 因此，我们可以令 $a=1$，并不断除以 2，直至 a 的大小在浮点系统中忽略不计，即与 x 相加后在计算机中判定为 x，从而得到所需的机器精度.

Algorithm 2 计算机器精度

```
float    a = 1
while 1 + a/2 ≠ 1 do
    a ← a/2
end while
得到机器精度 a
```

2. 上溢值

上溢值是计算机中对应类型所能储存的最大数，考虑数字在计算机中的二进制储存形式，我们假设一个 float 类型的数 a，令 $a=1$，从而我们可以将 a 乘以 2，并逐次判断 a 是否仍等于 $2\cdot a - a$，直至不满足循环条件时，即得上溢值.

Algorithm 3 计算上溢值

```
float    a = 1
while a = 2 · a − a do
    a ← 2 · a
end while
得到上溢值 a
```

3. 下溢值

下溢值是计算机所能储存的最小非零正数，考虑到数据在计算机中的二进制保存形式，我们可以将数字 1 不断除以 2，并逐次判断 a 是否等于 0，从而至循环结束后可得下溢值.

Algorithm 4 计算下溢值

1: $float \quad a = 1$
2: **while** $a/2 \neq 0$ **do**
3: $\quad a \leftarrow a/2$
4: **end while**
5: 得到下溢值 a

2.3 结果分析

图 1: 问题二的算法最终结果与理论值对比

1. 我们调用 limits 库中的 numeric_limits<float>::epsilon() 从而得到单精度浮点数所储存的理论机器精度：1.19209e-07, 这与我们所写程序得到的结果一致，从而证明该方法一定程度上是正确的，我们求出了单精度浮点数的机器精度.

2. 我们调用 limits 库中的 numeric_limits<float>::max() 从而得到单精度浮点数所储存的理论上溢值：3.40282e+38, 而程序所得结果为 1.70141e+38，基本可以认定为理论值的 1/2，可能是循环过程中累计的机器误差导致在最后一次循环后结果偏大，无法通过判断语句，从而影响最终结果. 根据单精度浮点数的结构，若让单精度浮点数可取得最大正值，可以令指数位为 127（根据程序所得结果，该值与现实中一致），再令尾数位均取得 1，即 $(2-2^{-23}) \times 2^{127} = 3.40282 \times 10^{38}$. 这与 limits 库中存储的理论上溢值一致.

3. 我们调用 limits 库中的 numeric_limits<float>::denorm_min() 从而得到单精度浮点数所储存的理论下溢值：1.4013e-45, 这与我们所写程序得到的结果一致，从而证明该方法一定程度上是正确的，我们求出了单精度浮点数的下溢值.

§3 结论

根据单双精度浮点数的二进制表示的定义，可以得到将十进制数转换为二进制浮点数的算法.
单精度浮点数的机器精度为：1.19209e-07，上溢值为：3.40282e+38，下溢值为：1.4013e-45.
双精度浮点数的机器精度为：2.22045e-16，上溢值为：1.79769e+308，下溢值为：4.94066e-324.

§4 附录: 题目一程序代码

```cpp
#include <iostream>
#include <iomanip>
#include <bitset>
#include <limits>

using namespace std;

int main() {
    double doubleNum;
    float floatNum;

    cout << "请输入一个数字：";
    cin >> doubleNum;

    if (doubleNum > numeric_limits<float>::max() ||
    doubleNum < -numeric_limits<float>::max()) {
        cout << "警告：输入的数字超出了单精度浮点数的范围！" << endl;
    }
    else {
        floatNum = static_cast<float>(doubleNum);
        cout << "单精度浮点数" << floatNum << "的二进制表示为：" << endl;
        for (int i = sizeof(floatNum) * 8 - 1; i >= 0; --i) {
            cout << ((reinterpret_cast<unsigned&>(floatNum) >> i) & 1);
            if (i % 8 == 0) cout << ' ';
        }
        cout << endl;
    }

    if (doubleNum > numeric_limits<double>::max() ||
    doubleNum < -numeric_limits<double>::max()) {
        cout << "警告：输入的数字超出了双精度浮点数的范围！" << endl;
    }
    else {
        cout << "双精度浮点数" << doubleNum << "的二进制表示为：" << endl;
        for (int i = sizeof(doubleNum) * 8 - 1; i >= 0; --i) {
            cout << ((reinterpret_cast<unsigned long long&>(doubleNum) >> i) & 1);
            if (i % 8 == 0) cout << ' ';
        }
        cout << endl;
    }
    return 0;
}
```

§5 附录: 题目二程序代码

```cpp
#include <iostream>
#include <limits>
#include <cmath>

using namespace std;

int main() {
    // 计算单精度浮点数的机器精度
    float single_epsilon = 1.0f;
    while (1.0f + single_epsilon / 2 != 1.0f) {
        single_epsilon /= 2.0f;
    }

    // 计算双精度浮点数的机器精度
    double double_epsilon = 1.0;
    while (1.0 + double_epsilon / 2 != 1.0) {
        double_epsilon /= 2.0;
    }

    // 计算单精度浮点数的最小值和最大值
    float single_min = 1.0f;
    float single_max = 1.0f;

    while (single_min / 2.0f != 0.0f) {
        single_min /= 2.0f;
    }

    while (single_max == single_max * 2.0f - single_max) {
        single_max *= 2.0f;
    }

    // 计算双精度浮点数的最小值和最大值
    double double_min = 1.0;
    double double_max = 1.0;

    while (double_min / 2.0 != 0.0) {
        double_min /= 2.0;
    }

    while (double_max == double_max * 2.0 - double_max) {
        double_max *= 2.0;
    }

```

```
44      cout << "单精度浮点数的机器精度:"
45      << single_epsilon << endl;
46      cout << "单精度浮点数的下溢值:"
47      << single_min << endl;
48      cout << "单精度浮点数的上溢值:"
49      << single_max << endl;
50      cout << "单精度浮点数的理论机器精度:"
51      << numeric_limits<float>::epsilon() << endl;
52      cout << "单精度浮点数的理论下溢值:"
53      << numeric_limits<float>::denorm_min() << endl;
54      cout << "单精度浮点数的理论上溢值:"
55      << numeric_limits<float>::max() << endl;
56      cout << endl;
57      cout << "双精度浮点数的机器精度:" << double_epsilon << endl;
58      cout << "双精度浮点数的下溢值:" << double_min << endl;
59      cout << "双精度浮点数的上溢值:" << double_max << endl;
60      cout << "双精度浮点数的理论机器精度:"
61      << numeric_limits<double>::epsilon() << endl;
62      cout << "双精度浮点数的理论下溢值:"
63      << numeric_limits<double>::denorm_min() << endl;
64      cout << "双精度浮点数的理论上溢值:"
65      << numeric_limits<double>::max() << endl;
66
67      return 0;
68  }
```