误差函数（也称之为高斯误差函数，error function or Gauss error function）是一个非基本函数（即不是初等函数），其在概率论、统计学以及偏微分方程中都有广泛的应用。它的表达式为，

$$erf\left(x\right)=\frac{2}{\sqrt{\pi}}\int_{0}^{x}e^{-\eta^{2}}d\eta$$

请用 C++编程，输出[-2, 2]区间误差函数的值，并绘制曲线。

1. 问题分析

以上误差函数的定义式包含一个积分，不利于编程。我们将它转换成级数表达式，

$$erf\left(x\right)=\frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}\left(-1\right)^{n}\frac{x^{2n+1}}{n!\left(2n+1\right)}$$

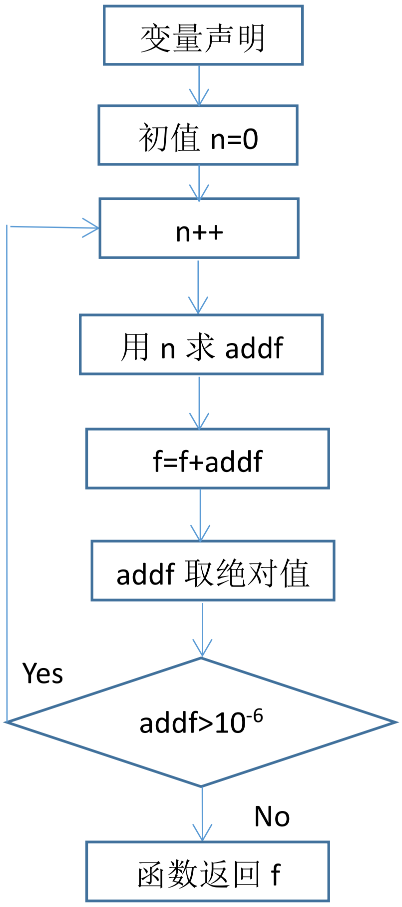
我们将问题分为两部分：误差函数迭代求解与误差函数的输出。其中，误差函数级数中有阶乘与幂次方表达式，又可被编程成两个独立的函数。课本已经有阶乘与幂次方的函数，如下。

```
#include <iostream>
using namespace std;
const double sqrt_PI=1.77245385; //sqrt_PI 是 PI 的开方
double factorial(long int n)
{
    double f=1.0;
    for(int i=2;i<=n; i++)
        f*= (double)i;
    return f;
}
double power(double x, int n)
{
    if (x == 0.) return 0.;
    double product=1.;
    if (n >= 0)
        while (n > 0)
            { product *= x;
              n--;
            }
    else
        while (n < 0)
            { product /= x;
              n++;
            }
    return product;
}
void main()
{


}
```

2. 误差函数迭代求解

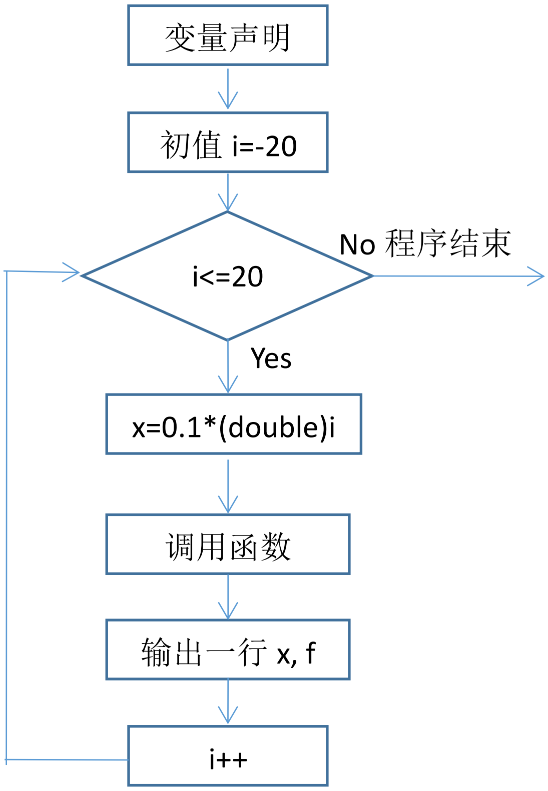
仿照以上两个函数的结构，添加一个函数计算 erf(x)。可参考以下流程图，写成 do-while 循环。其中步骤“用 n 求 addf”需要调用阶乘与幂次方的函数。




```
//误差迭代函数
double erf2(double x)
{
    int n = 0;
    double f = 0;
    double addf;
    do
    {
        addf = power(-1, n) * power(x, 2 * n + 1) / (factorial(n) * (2 * n + 1));
        f += addf;
        addf = fabs(addf);
        n++;
    } while (addf > 10e-6);
    return 2 / sqrt_PI * f;
}
```

3. 误差函数的输出

参考以下流程图，写一个循环 for 循环输出 x 与 erf(x)。




```
int main()
{
    for(int i = -20; i <= 20; i++)
    {
        double x = 0.1 * double (i);
        double f = erf2(x);
        double f_std = erf(x); //标准库中的erf()函数，用于检验结果是否正确
        cout << fixed << setprecision(1) << setw(4) << x << "    ";
        cout << fixed << setprecision(6) << setw(9) << f << "    ";
        cout << fixed << setprecision(6) << setw(9) << f_std << endl; //打印正确结果
        //为了输出结果美观，此处使用流操纵算子进行格式化输出，需要<iomanip>头文件

        return 0;
}
```

输出结果为：

其中第二列为自定义函数的输出，第三列为标准库中的 erf()函数的输出

-2.0	-0.995322	-0.995322
-1.9	-0.992791	-0.992790
-1.8	-0.989090	-0.989091
-1.7	-0.983789	-0.983790
-1.6	-0.976348	-0.976348
-1.5	-0.966105	-0.966105
-1.4	-0.952285	-0.952285
-1.3	-0.934008	-0.934008
-1.2	-0.910313	-0.910314
-1.1	-0.880206	-0.880205
-1.0	-0.842701	-0.842701
-0.9	-0.796908	-0.796908
-0.8	-0.742101	-0.742101
-0.7	-0.677801	-0.677801
-0.6	-0.603856	-0.603856
-0.5	-0.520500	-0.520500
-0.4	-0.428392	-0.428392
-0.3	-0.328627	-0.328627
-0.2	-0.222703	-0.222703
-0.1	-0.112463	-0.112463
0.0	0.000000	0.000000
0.1	0.112463	0.112463
0.2	0.222703	0.222703
0.3	0.328627	0.328627
0.4	0.428392	0.428392
0.5	0.520500	0.520500
0.6	0.603856	0.603856
0.7	0.677801	0.677801
0.8	0.742101	0.742101
0.9	0.796908	0.796908
1.0	0.842701	0.842701
1.1	0.880206	0.880205
1.2	0.910313	0.910314
1.3	0.934008	0.934008
1.4	0.952285	0.952285
1.5	0.966105	0.966105
1.6	0.976348	0.976348
1.7	0.983789	0.983790
1.8	0.989090	0.989091
1.9	0.992791	0.992790
2.0	0.995322	0.995322