Computational Physics Assignment I

学号：515072910017

姓名：李进辉

Note：All codes are in C language, using -std=c99 standard.

---

1. Detemine the value of A

e = 14 ;f = 0.625
so mantissa = 1.f = 1.625000 ;true exponent =14 - 127 = -113
so the full value: $A = 1.625000 * 2^{-113}$

2. Find an equivalent formula

(a) $ln(1+1/x)$

(b) $\frac{1}{\sqrt{x^2+1}+x}$

(c) $cos(2x)$

(d) $cos(\frac{x}{2})$

3. Write a program to determine limits

- For single precision:

```c
#include<stdio.h>
int main(){
    int N = 200;
    float under = 1, over = 1;
    for (int i = 0; i<N; i++){
        under /= 2.0;
        over *= 2.0;
        printf("N = %d, under = %e, over = %e\n", i+1, under, over);
    }
    return 0;
}
```

Result:

```
N = 145, under = 2.242078e-044, over = 1.#INF00e+000
N = 146, under = 1.121039e-044, over = 1.#INF00e+000
N = 147, under = 5.605194e-045, over = 1.#INF00e+000
N = 148, under = 2.802597e-045, over = 1.#INF00e+000
N = 149, under = 1.401298e-045, over = 1.#INF00e+000
N = 150, under = 0.000000e+000, over = 1.#INF00e+000
N = 151, under = 0.000000e+000, over = 1.#INF00e+000
N = 152, under = 0.000000e+000, over = 1.#INF00e+000
```

```
N = 121, under = 3.761582e-037, over = 2.658456e+036
N = 122, under = 1.880791e-037, over = 5.316912e+036
N = 123, under = 9.403955e-038, over = 1.063382e+037
N = 124, under = 4.701977e-038, over = 2.126765e+037
N = 125, under = 2.350989e-038, over = 4.253530e+037
N = 126, under = 1.175494e-038, over = 8.507059e+037
N = 127, under = 5.877472e-039, over = 1.701412e+038
N = 128, under = 2.938736e-039, over = 1.#INF00e+000
N = 129, under = 1.469368e-039, over = 1.#INF00e+000
N = 130, under = 7.346840e-040, over = 1.#INF00e+000
N = 131, under = 3.673420e-040, over = 1.#INF00e+000
N = 132, under = 1.836710e-040, over = 1.#INF00e+000
```

As we can see int the result, the underflow of float is **1.401298e-045**, the overflow is **1.701412e+0.38**.

- For double precision:

```c
#include<stdio.h>
int main(){
    int N = 1500;
    double under = 1, over = 1;
    for (int i = 0; i<N; i++){
        under /= 2.0;
        over *= 2.0;
        printf("N = %d, under = %e, over = %e\n", i+1, under, over);
    }
    return 0;
}
```

Result:

```
N = 1071, under = 3.952525e-323, over = 1.#INF00e+000
N = 1072, under = 1.976263e-323, over = 1.#INF00e+000
N = 1073, under = 9.881313e-324, over = 1.#INF00e+000
N = 1074, under = 4.940656e-324, over = 1.#INF00e+000
N = 1075, under = 0.000000e+000, over = 1.#INF00e+000
N = 1076, under = 0.000000e+000, over = 1.#INF00e+000
N = 1077, under = 0.000000e+000, over = 1.#INF00e+000
N = 1078, under = 0.000000e+000, over = 1.#INF00e+000
```

```
N = 1016, under = 1.424047e-306, over = 7.022239e+305
N = 1017, under = 7.120236e-307, over = 1.404448e+306
N = 1018, under = 3.560118e-307, over = 2.808896e+306
N = 1019, under = 1.780059e-307, over = 5.617791e+306
N = 1020, under = 8.900295e-308, over = 1.123558e+307
N = 1021, under = 4.450148e-308, over = 2.247116e+307
N = 1022, under = 2.225074e-308, over = 4.494233e+307
N = 1023, under = 1.112537e-308, over = 8.988466e+307
N = 1024, under = 5.562685e-309, over = 1.#INF00e+000
N = 1025, under = 2.781342e-309, over = 1.#INF00e+000
N = 1026, under = 1.390671e-309, over = 1.#INF00e+000
N = 1027, under = 6.953356e-310, over = 1.#INF00e+000
```

As we can see int the result, the underflow of float is **4.940656e-324**, the overflow is **8.988466e+307**.

4. Write a program to determine your machine precision

- For single precision:

```c
#include<stdio.h>
int main(){
    float eps = 1.0;
    while((1 + eps/2) != 1){
        eps /= 2;
    }
    printf("Eps = %e\n", eps);
    return 0;
}
```

We can get the single precision: **1.192093e-007**

- For double precision:

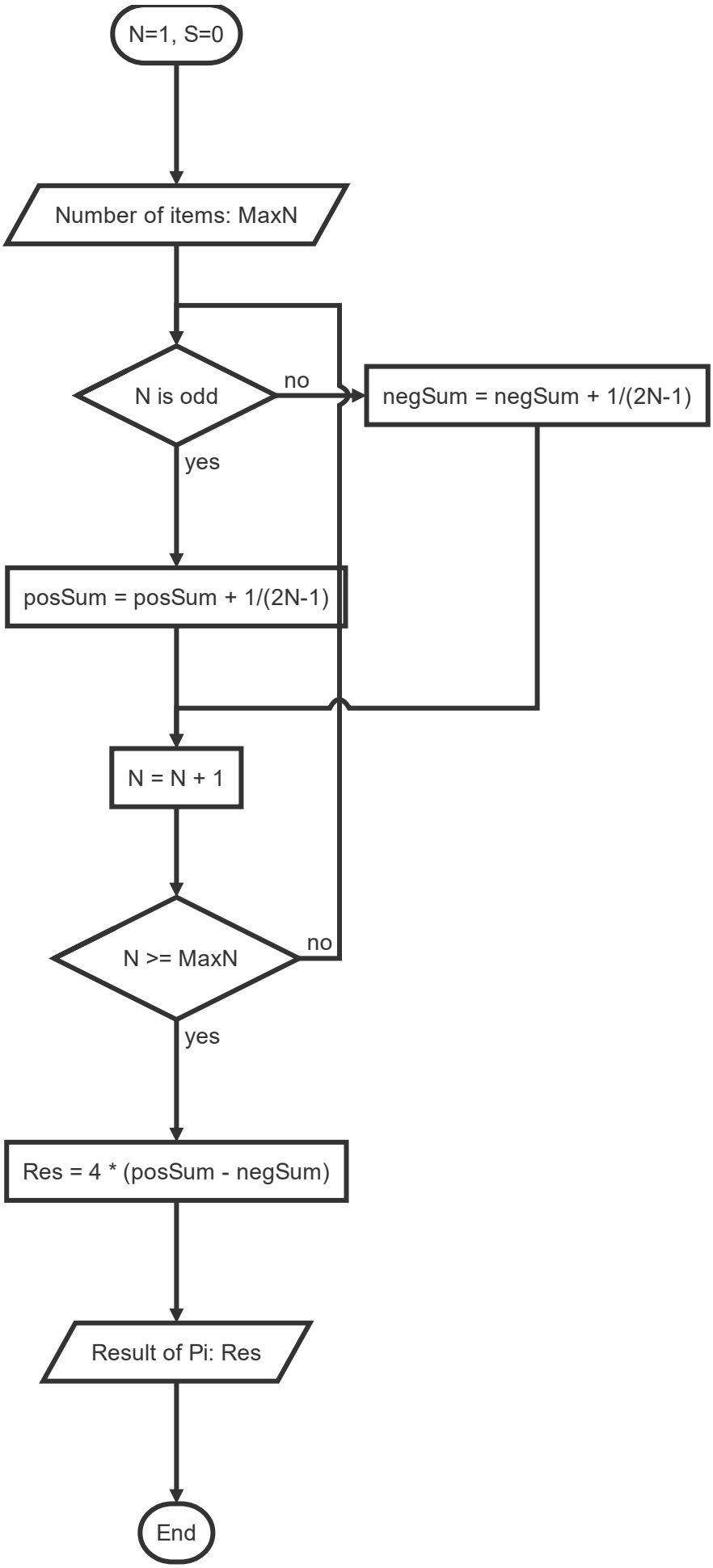
```c
#include<stdio.h>
int main(){
    double eps = 1.0;
    while((1 + eps/2) != 1){
        eps /= 2;
    }
    printf("Eps = %e\n", eps);
    return 0;
}
```

We can get the double precision: **2.220446e-016**

5. The value of π

1. Describe algorithm

- **Step:**


N=1, S=0
Number of items: MaxN
N is odd
no
yes
negSum = negSum + 1/(2N-1)
posSum = posSum + 1/(2N-1)
N = N + 1
N >= MaxN
no
yes
Res = 4 * (posSum - negSum)
Result of Pi: Res
End

2. Write a program

```
#include<stdio.h>
int main(){
    int MaxN;
    float posSum = 0;
    float negSum = 0;
    printf("Please input the number of the items:");
    scanf("%d", &MaxN);
    for (int i=1; i<MaxN; i++){
        if (i%2 != 0){
            posSum += 1.0/(2*i-1);
        }else {
            negSum += 1.0/(2*i-1);
        }
    }
    printf("The value of Pi is %f", 4*(posSum-negSum));
    return 0;
}
```

The relative error is: $\frac{Res-\pi}{\pi}$

3. calculate π and relative error

- n=10, π = 3.252367, relative error = 3.526%
- n=20, π = 3.194189, relative error = 1.674%
- n=20, π = 3.167231, relative error = 0.816%
- n=1000, π = 3.142588, relative error = 0.032%

4. Comment

As we can see, the relative error decreases as the n increases, in order to get a more accurate solution, we have to increase n as much as we can.