

# noj实验2报告

## 0.0 实验题目//需求分析

File Name: T002.cpp

### 实验1.2: 高精度计算PI值

Time Limit: 3000ms , Memory Limit: 10000KB , Accepted: 0 , Total Submissions: 0

VIDEO1

#### Description

限制使用双向链表作存储结构, 请根据用户输入的一个整数(该整数表示精确位数), 高精度计算PI值。可以利用反三角函数幂级展开式来进行计算。

#### Input

输入的一个正整数n

#### Output

输出PI的值, 精确到小数点后n位, 最后输出一个回车。

Sample Input	5
Sample Output	3.14159

需求分析:

用三角函数的展开式计算PI的高精度值。

问题在于存储展开式数据。采用类似于多项式, 但加上双向链表特点即可解决。

## 1.0 概要设计

### 1.1 建立链表

```
typedef struct node {  
  
    int data;  
    struct node*next;  
    struct node*pre;  
}node,*list;
```

## 1.2 设计函数

### 1.2.1 申请空间函数

```
void init(list &l)
```

主要实现malloc操作。

### 1.2.2 释放空间

```
void destorylist(list &l)
```

### 1.2.3 插入元素操作

```
void insert(list &l, int data)
```

### 1.2.4 主操作函数

```
main
```

## 2.0 代码//详细设计

```
#include<stdio.h>  
#include<stdlib.h>  
  
typedef struct node {  
    int data;  
    struct node*next;  
    struct node*pre;  
}node,*list;  
  
int n;  
  
void init(list &l)//初始化链表  
{  
    list tail = l, p;  
    int i = 0;  
    p=(list)malloc(sizeof(node));  
    tail->next = p;  
    p->pre = tail;  
    tail = p;  
    p->data = 2;  
    tail->next = NULL;  
}  
  
void destorylist(list &l)//释放节点
```

```

{
    list p=l->next;
    list q;
    while (p->next)
    {
        q = p->next;
        free(p);
        p = q;
    }
    free(l);
    l = NULL;
}

void insert(list &l, int data)//插入函数
{
    list p = l;
    if(p == NULL)
    {
        return;
    }
    else
    {
        while (p->next)
            p = p->next;
        list temp = (list)malloc(sizeof(node));
        p->next = temp;
        temp->data = data;
        temp->pre = p;
        temp->next = NULL;
    }
}

void tran(list l)//空时
{
    printf("%d.",l->next->data);
    l=l->next;
    list p;
    p=l->next;
    while(p!=NULL&&n)
    {
        printf("%d",p->data);
        p=p->next;
        n--;
    }
    printf("\n");
}

int main()//主要操作函数
{
    list p,sum1;
    int i;

```

```

scanf("%d",&n);
p=(list)malloc(sizeof(node));
sum1=(list)malloc(sizeof(node));
p->next = NULL;

```

```

sum1->next=NULL;
p->pre = NULL;
sum1->pre=NULL;

init(p);
init(sum1);

for(i=1;i<=1000;i++)
{
    insert(p,0);
    insert(sum1,0);
}
list q,sum;
int ret,tmp,cnt=3,num=1;
int flag=1;
while(flag)
{
    q=p->next;
    sum=sum1->next;
    ret=0;
    while(q->next)
    {
        q=q->next;
    }
    while (q)
    {
        tmp=q->data*num+ret;
        q->data=tmp%10;
        ret=tmp/10;
        if(q->pre==NULL)break;
        else
        {
            q=q->pre;
        }
    }

    ret=0;
    q=p->next;
    while (q)
    {
        tmp=q->data+ret*10;
        q->data=tmp/cnt;
        ret=tmp%cnt;
        if(q->next==NULL)break;
        else
        {
            q=q->next;
        }
    }

    q=p->next;
    sum=sum1->next;
    while(q->next&&sum->next)
    {
        q=q->next;
        sum=sum->next;
    }
    flag=0;
}

```

```
while (q&&sum)
{

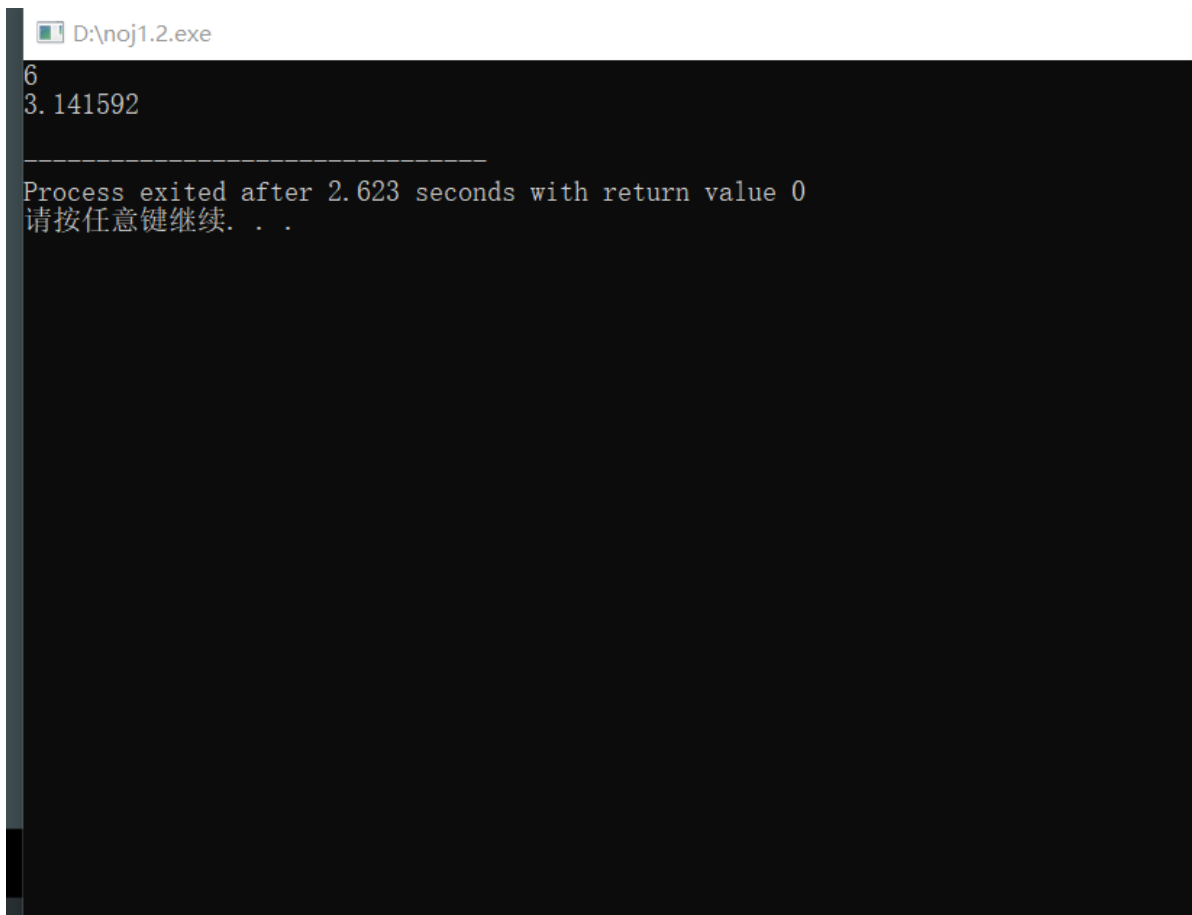
    tmp=sum->data+q->data ;
    sum->data=tmp%10;
    if(sum->pre==NULL || q->pre==NULL)
    {
        break;
    }
    else
    {
        sum->pre->data+=tmp/10;
        flag |= q->data;
        sum=sum->pre;
        q=q->pre;
    }

}

num++;
cnt+=2;
}
tran(sum1);
destorylist(p);
destorylist(sum1);
return 0;
```

```
}
```

### 3.0测试结果



```
D:\noj1.2.exe
6
3.141592
-----
Process exited after 2.623 seconds with return value 0
请按任意键继续. . .
```

```
D:\noj1.2.exe
10000
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480865132823066
470938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756659334461284756482337867831
652712019091456485669234603486104543266482133936072602491412737245870066063155881748815209209628292540917153643678925903
600113305305488204665213841469519415116094330572703657595919530921861173819326117931051185480744623799627495673518857527
248912279381830119491298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051
320005681271452635608277857713427577896091736371787214684409012249534301465495853710507922796892589235420199561121290219
608640344181598136297747713099605187072113499999983729780499510597317328160963185950244594553469083026425223082533446850
352619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805
321712268066130019278766111959092164198681

-----
Process exited after 2.808 seconds with return value 0
请按任意键继续. . .
```

## 4.0 实验心得

实现链表真的是麻烦而不太难的事情，反复在调指针的bug。

在纸上的分析确实很重要，刚开始毫无思路，到后来纸上分析后，就简化为双向链表的存取及运算。

注：参考了网上资料思路。