实验一

姓名：黄丹禹

学号：2012030

日期：2022/10/26

一、 实验平台

a) 基于 Visual Studio 2022，使用+语言编写

b) 调用函数库：

```
#include <iostream>
#include <vector>
#include <string>
```

二、 实验目的

单序列基本操作（并判断是否受序列长度影响，对于无限长序列应完车即时允许情况下应完成即时操作）

a) 满足前、后补零操作

b) 满足序列延迟、提前操作

c) 满足序列反转操作

d) 满足序列拉伸、压缩操作（上采样、下采样）

e) 满足序列差分、累加操作

三、 实验核心公式及问题

a) 前后补零实现：

实验中，后补零直接向 vector 中利用 push_back()函数插入 k 个 0；前补零通过 vector 的 insert 函数，向位置 0 插入 k 个 0。

b) 若用数组完成补零操作:

方法一：为新建固定长度的数组，长度固定比如为 20，后补零就是将已初始化的部分后面继续初始化 k 个 0 即可；前补零需要将数组中已经初始化的每个元素从后向前，每个元素向后移动 k 个位置，之后把移动出来的 k 个位置赋成 0。当数组长度不够是，将数组长度扩大为原来的二倍后再插入新的。

方法二：通过 begin 和 end 值得到已有（end-begin+1）个数据，每次补零都新建一个

数组将新插入的 0 和原数据复制到新数组中，之后 delete 原数组，原理和方法一相同，不同处在于方法一有空间冗余但更新数组次数（即 double 数组长度的次数）较少，时间复杂度较低，方法二每次都需要 new 一个新数组并进行大量数据移动、delete 旧数组，时间复杂度很大。

c) 上采样后的序列长度：

设拉伸 num 倍，上采样后的序列的长度不单单是原序列长度乘 num 这么简单，因为 0 时刻不移动，相当于以 0 时刻为基准，向正负无穷方向拉伸，所以相当于把序列的初始、结束时间扩大 num 倍。原序列的长度为(end-begin+1)，结果序列的长度为(end*num-begin*num+1)。

d) 下采样中 0 时刻的保留：

传入的 num 为采样率，通过循环中每次将循环的 i=i+num 来控制采样频率。

通过将第一个采样点设置为abs(begin)%num来确保下采样中一定能采到0时刻的数据，并保证 0 时刻的数据不变（相当于以 0 时刻数据为中心向两侧以每 num 个数据采一次样）。

四、 实验设计

a) 前后补零：

前补零（flag 为 1 时在左侧补零）通过 insert 函数在 0 位置处插入 num 个 0；

后补零(flag 为 0 时在右侧补零)通过 vector 封装的 push_back 函数向后添加 num 个 0。

数组的实现方式、思路见第三部分核心问题 1。

```
//补零
seq seq::supplement(int flag, int num) {
    //flag==1 左侧补零
    if (flag == 1) {
        for (int i = 0; i < num; i++) {
            this->a.insert(a.begin(), 0);
        }
        this->begin = this->begin - num;
    }
    //flag==0, 右侧补零
    else {
        for (int i = 0; i < num; i++) {
            this->a.push_back(0);
        }
        this->end = this->end + num;
    }
    return *this;
}
```

测试数据：A={1,2,3} ，n=0:2，在左侧补 3 个 0：

```
补零前：begin:0
end:2
1 2 3
补零后：begin:-3
end:2
0 0 0 1 2 3
```

测试数据：A={1,2,3} ，n=0:2，在右侧补 2 个 0：

```
补零前：begin:0
end:2
1 2 3
补零后：begin:0
end:4
1 2 3 0 0
```

无限长序列可以进行前向补零：

初始化为 A={1,2,3} n=0:2。在无限长序列中输入 5，并在前向补 6 个零：

```
初始化序列：
begin:0
end:2
1 2 3
5
6
begin:-6
end:3
0 0 0 0 0 0 1 2 3 5
```

b) 序列延迟、提前操作：

修改序列的 begin 和 end 后将修改后的序列返回即可：

```
//序列延迟、提前操作
seq seq::move(int flag, int num) {
    //flag==1，减法，相当于延迟
    if (flag == 1) {
        this->begin -= num;
        this->end -= num;
    }
    //加法，提前
    else {
        this->begin += num;
        this->end += num;
    }
    return *this;
}
```

测试数据：A={1,2,3} ，n=0:2，延迟 3：

```
延迟前：begin:0
end:2
1 2 3
延迟后：begin:3
end:5
1 2 3
```

测试数据：A={1,2,3} ，n=0:2，提前 2：

```
提前前: begin:0
end:2
1 2 3
提前后: begin:-2
end:0
1 2 3
```

无法操作无限长序列。

c) 序列反转操作：

以中间数为轴，两侧的数据进行交换，以得到反转后的 vector，然后修改序列的 begin 和 end 值即可：

```
//翻转
seq seq::reverse() {
    int len = this->a.size();
    double temp = 0;
    for (int i = 0; i < len / 2; i++) {
        temp = a[i];
        a[i] = a[len - 1 - i];
        a[len - 1 - i] = temp;
    }
    temp = begin;
    begin = end * (-1);
    end = temp * (-1);
    return *this;
}
```

测试数据：A={1,2,3} ，n=0:2 ：

```
翻转前: begin:0
end:2
1 2 3
翻转后: begin:2
end:0
3 2 1
```

无法操作无限长序列。

d) 序列拉伸操作（上采样）

原序列的长度为(end-begin+1)，结果序列的长度为(end*num-begin*num+1)，详细解释见第三部分核心问题 3。

注意：由于上采样的结果序列中包含所有原序列中的值，所以 0 时刻不需要特殊考虑（与下采样不同），所以通过 i 循环拉伸后的长度次，通过 i 与采样率 n 除法的余数控制插入的值为原数据还是 0，最后得到上采样后的序列。

```
//拉伸
seq seq::wider(int num) {
    vector<double> temp;
    for (int i = 0; i < (end*num-begin*num+1); i++) {
        if (i % num == 0)
            temp.push_back(this->a[i / num]);
        else
            temp.push_back(0);
    }
    seq t(this->begin * num, end * num, temp);
    return t;
}
```

测试数据：A={1,2,3} ，n=-1:1，上采样率为 3：

```
原序列:
begin:-1
end:1
1 2 3
上采样结果:
begin:-3
end:3
1 0 0 2 0 0 3
```

无法操作无限长序列。

e) 序列压缩操作（下采样）

注意如何从头遍历 vector 过程中保证 0 时刻可以被采样，达到以 0 时刻为基准，从左右两侧向 0 时刻压缩 num 倍的效果，详细解释见第三部分核心问题 4。

```
//压缩
seq seq::shorter(int num) {
    vector<double> temp;
    for (int i = abs(begin)%num; i < (end  - begin  + 1); i=i+num) {
        temp.push_back(a[i]);
    }
    seq t(begin / num, end / num, temp);
    return t;
}
```

测试数据：A={1,2,3,4,5,6,7,8,9,0} ，n=--3:6，上采样率为 2：

```
原序列:
begin:-3
end:6
1 2 3 4 5 6 7 8 9 0
下采样结果:
begin:-1
end:3
2 4 6 8 0
```

无法操作无限长序列。

f) 序列差分操作

N 重差分即为在 N-1 差分结果的基础上再进行一次差分操作，所以先构造单次差分：

```
vector<double> seq::difference0(vector<double> temp) {
    vector<double> t2;
    for (int i = 0; i < temp.size()-1; i++) {
        t2.push_back(temp[i + 1] - temp[i]);
    }
    return t2;
}
```

在多重差分中循环调用：

```
//差分
seq seq::difference(int num) {
    vector<double> temp=this->a;
    for (int i = 0; i < num; i++) {
        temp = this->difference0(temp);
    }
    seq t(begin, end - num, temp);
    return t;
}
```

处理无限长序列：

测试数据：A={5,9,4,1,6} n=0:4　输入 8 后，序列变为 5,9,4,1,6,8，二重积分结果为-9,2,8,-3，符合预期：

```
初始化序列:
begin:0
end:4
5 9 4 1 6
8
n重差分结果：begin:0
end:3
-9 2 8 -3
```

g) 序列累加操作

将序列中所有值进行累加，将结果返回：

```
//累加
double seq::accumulate() {
    /* ... */
    double sum = 0;
    for (int i = 0; i < a.size(); i++) {
        sum = sum + a.at(i);
    }
    return sum;
}
```

测试数据：

A={1,2,3} n=0:2,输入 8、2 得到 14、16，符合预期：

```
初始化序列:
begin:0
end:2
1 2 3
输入前累加结果：6
8
输入后累加结果：14
2
输入后累加结果：16
```