**WEEK9 Operator Overwrite, MSD, LSD**

郁思敏 10153901225

1. **Main.cpp**

#include <iostream>

#include <vector>

#include "Integer.h"

#include "Sort.h"

using namespace std;

int main()

{

int i;

int n;

int digit = 0;

cout << "please enter the size of your vector:" << endl;

cin >> n;

cin.clear();

cin.sync();

vector<Integer> a;

for(i=0; i<n; i++)

{

Integer temp(1);

a.push\_back(temp);

}

for(i=0; i<n; i++)

{

if(digit < a[i].GetSize())

digit = a[i].GetSize();

}

//MSD(a, 0, n-1, digit);

LSD(a, 0, n-1, 1, digit);

for(i=0; i<n; i++)

{

cout << a[i];

}

//test operator overwrite

//a:123456789123456789123456789123456789123456789123456789123456789123456789

//b:24353

// Integer a(1);

// Integer b(1);

// Integer c;

// c = a+b;

// cout << "a:" << a << endl;

// cout << "b:" << b << endl;

// cout << "c:" << c << endl;

// if(a>b)

// cout << "a>b" << endl;

// else

// cout << "a<=b" << endl;

return 0;

}

1. **Integer.h**

#ifndef INTEGER\_H

#define INTEGER\_H

#include <iostream>

#include <vector>

using namespace std;

class Integer

{

public:

Integer();

Integer(int temp); //initialize vector.

int GetSize(); //number.size()

int GetOne(int position); //return number[position]

//comparison operators

bool operator ==(const Integer &b)const;

bool operator >(const Integer &b)const;

bool operator <(const Integer &b)const;

//assignment operator

Integer &operator =(const Integer &b);

//arithmetic operators

Integer operator +(Integer &b);

//out stream operator

friend ostream &operator <<(ostream &out, Integer &a);

//private:

vector<int> number;

};

#endif // INTEGER\_H

1. **Integer.cpp**

#include "Integer.h"

Integer::Integer()

{

//do nothing

}

Integer::Integer(int temp) //initialize vector.

{

char ch;

bool flag = true;

cout << "please enter your number:" << endl;

do

{

ch = cin.get();

if(ch != '\n')

{

number.push\_back((ch-'0'));

}

else

flag = false;

}

while(flag);

}

int Integer::GetSize()

{

return number.size();

}

int Integer::GetOne(int position)

{

return number[position];

}

ostream &operator <<(ostream &out, Integer &a)

{

for(unsigned i=0; i<a.number.size(); i++)

out << a.number[i];

out << " ";

return out;

}

bool Integer::operator ==(const Integer &b)const

{

bool flag = true;

if(number.size() == b.number.size())

{

for(unsigned i=0; i<number.size(); i++)

{

if(number[i] != b.number[i])

{

flag = false;

break;

}

}

}

else

flag = false;

return flag;

}

bool Integer::operator >(const Integer &b)const

{

bool flag = true;

if(number.size() == b.number.size())

{

for(unsigned i=0; i<number.size(); i++)

{

if(number[i] <= b.number[i])

{

flag = false;

break;

}

}

}

else if(number.size() < b.number.size())

flag = false;

return flag;

}

bool Integer::operator <(const Integer &b)const

{

bool flag = true;

if(number.size() == b.number.size())

{

for(unsigned i=0; i<number.size(); i++)

{

if(number[i] >= b.number[i])

{

flag = false;

break;

}

}

}

else if(number.size() > b.number.size())

flag = false;

return flag;

}

Integer& Integer::operator =(const Integer &b)

{

number.clear();

for(unsigned i=0; i<b.number.size(); i++)

number.push\_back(b.number[i]);

return \*this;

}

Integer Integer::operator +(Integer &b)

{

Integer temp;

int i, j, digit, carry=0;

for(i=(int)(number.size()-1), j=b.GetSize()-1; i>=0 && j>=0; i--, j--)

{

digit = number[i]+b.number[j]+carry;

if(digit>=10)

{

digit -= 10;

carry = 1;

}

else

carry = 0;

temp.number.insert(temp.number.begin(), digit);

}

while(i>=0)

{

digit = number[i]+carry;

if(digit>=10)

{

digit -= 10;

carry = 1;

}

else

carry = 0;

temp.number.insert(temp.number.begin(), digit);

i--;

}

while(j>=0)

{

digit = b.number[j]+carry;

if(digit>=10)

{

digit -= 10;

carry = 1;

}

else

carry = 0;

temp.number.insert(temp.number.begin(), digit);

j--;

}

return temp;

}

1. **Sort.h**

#ifndef SORT\_H\_INCLUDED

#define SORT\_H\_INCLUDED

#include <iostream>

#include <vector>

#include "Integer.h"

using namespace std;

int GetDigit(Integer &x, int digit) //return the number of x in digit-th location

{

int size = x.GetSize();

if(digit > size)

return 0;

else

return x.GetOne(size-digit);

}

void MSD(vector<Integer> &arr, int start, int end, int digit) //current digit-th

{

const int radix = 10; //0~10 bucket

int cnt[radix], i, j;

// 10 buckets

for(i=0; i<radix; i++)

{

cnt[i] = 0;

}

Integer \*bucket = new Integer[end-start+1];

//calculate each bucket's number

for(i=start; i<=end; i++)

{

cnt[GetDigit(arr[i], digit)]++;

}

//calculate bucket's edge index

for(i=1; i<radix; i++)

{

cnt[i] = cnt[i]+cnt[i-1];

}

for(i=end; i>=start; i--)

{

j = GetDigit(arr[i], digit);

bucket[cnt[j]-1] = arr[i];

cnt[j]--;

}

//copy bucket to array.

j = 0;

for(i=start; i<=end; i++)

{

arr[i] = bucket[j++];

}

delete(bucket);

//iteration in each bucket.

for(i=0; i<radix; i++)

{

int p1, p2;

p1 = start+cnt[i];

if(i==radix-1)

p2 = end-start;

else

p2 = start+cnt[i+1]-1;

if(p1<p2 && digit>1)

{

MSD(arr, p1, p2, digit-1);

}

}

}

void LSD\_(vector<Integer> &arr, int start, int end, int digit) //current digit-th

{

const int radix = 10; //0~10 bucket

int cnt[radix], i, j;

// 10 buckets

for(i=0; i<radix; i++)

{

cnt[i] = 0;

}

Integer \*bucket = new Integer[end-start+1];

//calculate each bucket's number

for(i=start; i<=end; i++)

{

cnt[GetDigit(arr[i], digit)]++;

}

//calculate bucket's edge index

for(i=1; i<radix; i++)

{

cnt[i] = cnt[i]+cnt[i-1];

}

for(i=end; i>=start; i--)

{

j = GetDigit(arr[i], digit);

bucket[cnt[j]-1] = arr[i];

cnt[j]--;

}

//copy bucket to array.

j = 0;

for(i=start; i<=end; i++)

{

arr[i] = bucket[j++];

}

delete(bucket);

}

void LSD(vector<Integer> &arr, int start, int end, int digit, int max\_digit)

{

while(digit <= max\_digit)

{

LSD\_(arr, start, end, digit);

digit++;

}

}

#endif // SORT\_H\_INCLUDED