**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

一、调试成功程序及说明

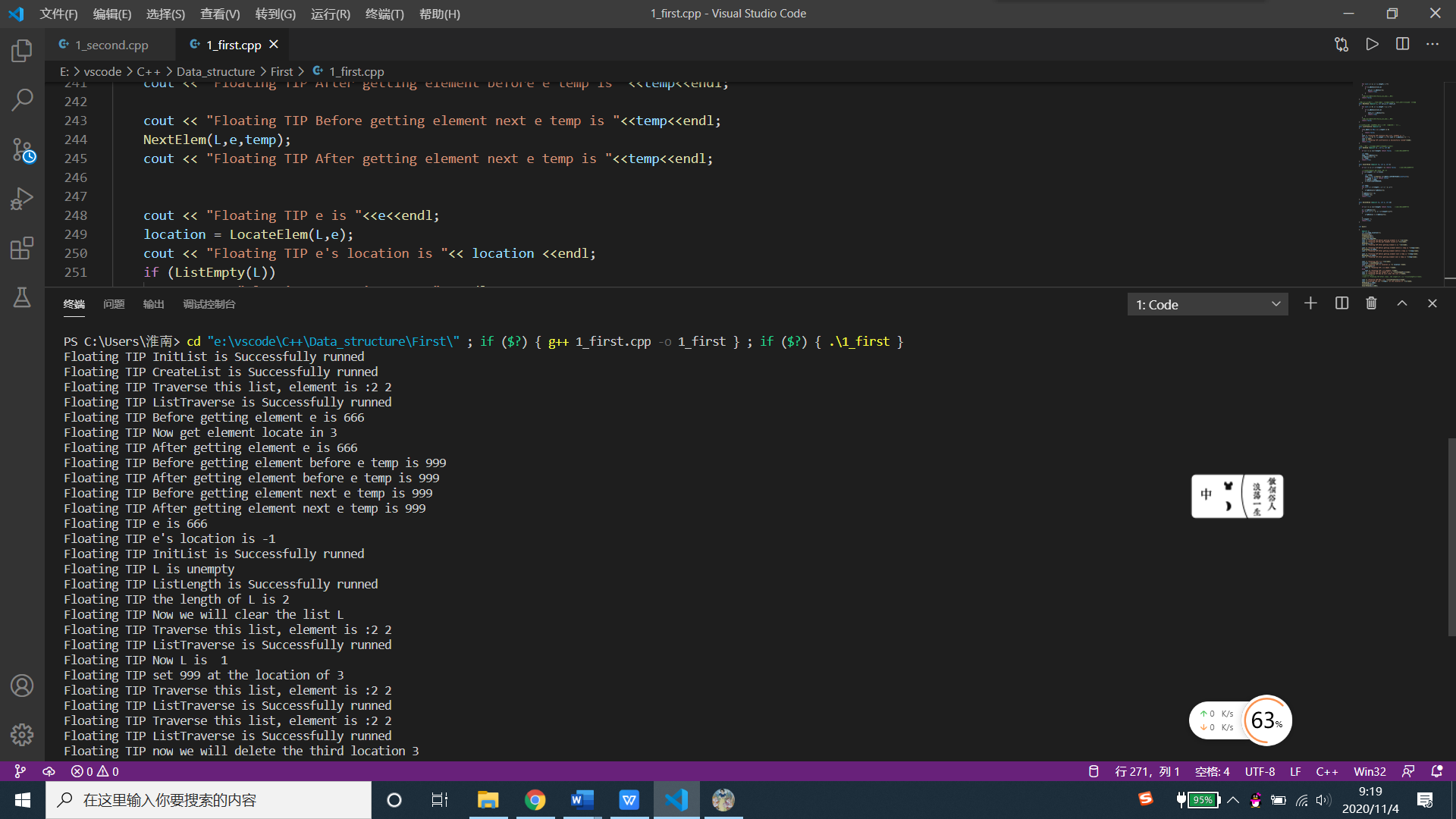
1、

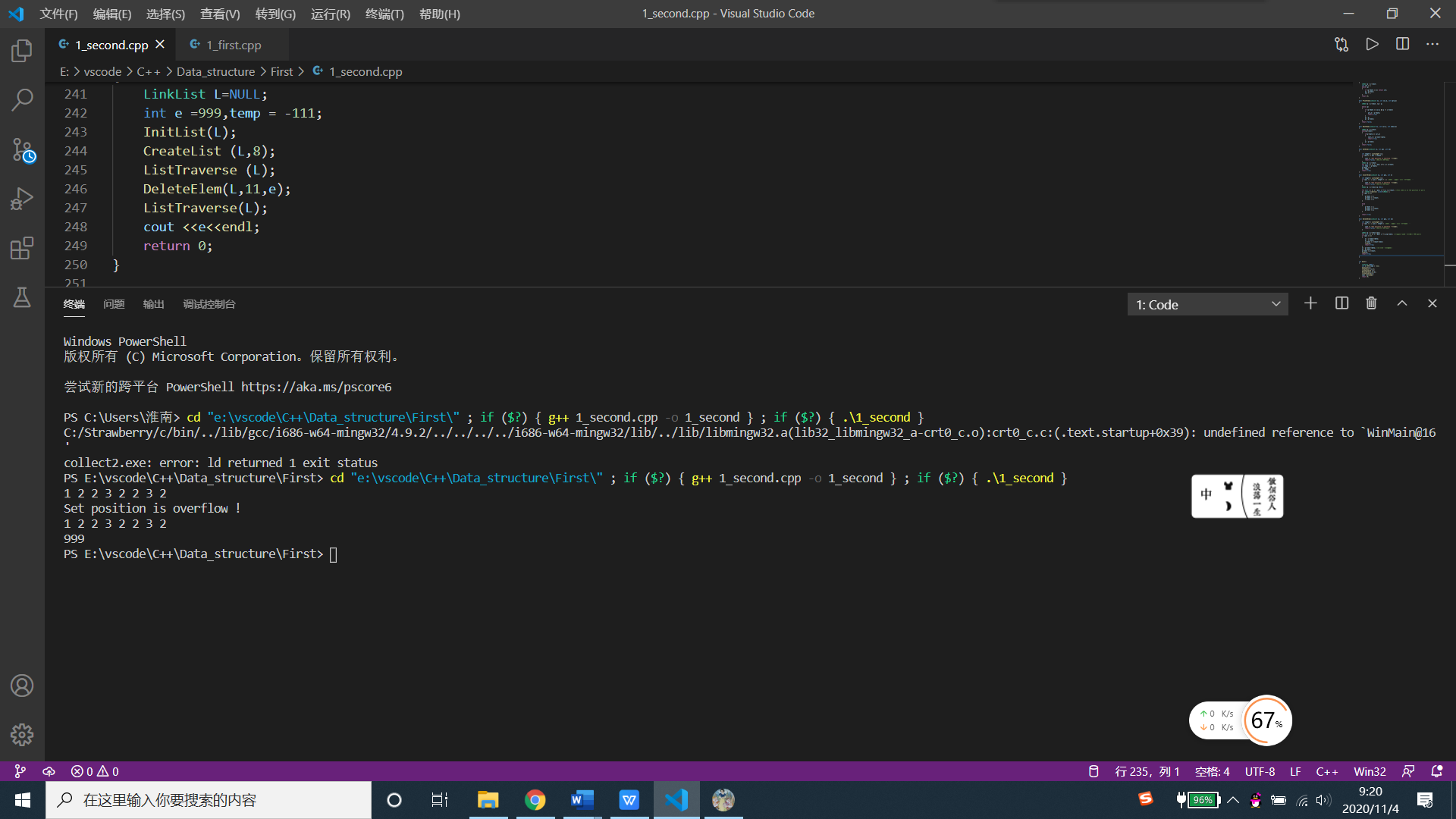
题目：编程实现课本P12 ADT List 基本操作14个：

* 1. 用顺序存储结构实现； （2）用链式存储结构实现；

算法思想：根据上课讲解，实现两种储存方式下的基本操作，为了方便调试，可以先实现遍历输出的函数，后续操作调试起来更直观。

运行结果：





结果分析：结果均正确，与预期结果一致

附源程序。

顺序储存

#include <iostream>

#include <cstdlib>

using namespace std;

/\*

1．编程实现书P12 ADT List 基本操作14个：

（1）用顺序存储结构实现； （2）用链式存储结构实现；

\*/

const int LISTINITSIZE = 16;

const int LISTICRESIZE = 8;

typedef struct SqeList

{

int \*pData = NULL; //动态储存空间的基地址

int length; //储存元素个数

int size; //当前已分配空间大小

}SqeList;

bool InitList(SqeList \*L) //传入的是引用型，用指针来传

{

L->pData = (int\*)malloc(LISTINITSIZE\*sizeof(int));

if(L->pData == NULL)

{

cout << "Floating TIP Memory overflow"<<endl;

exit(0);

}

L->size = LISTINITSIZE;

L->length = 0;

cout << "Floating TIP InitList is Successfully runned"<<endl;

return true;

}

bool CreateList(SqeList \*L)

{

int n;

//cout << "Please enter the length of this SqeList you want to creat: "<<endl;

//cin >>n;

n=2;

while(n > L->size)

{

int \*temp;

temp = (int \*)realloc (L->pData,LISTINITSIZE\*sizeof(int));

if (temp == NULL) return false;

L->pData = temp;

L->size+=LISTICRESIZE;

cout <<"Floating TIP L.size has increased and now is "<<L->size<<endl;

}

//cout <<"Please enter "<<n<<" elements separated by Spaces :"<<endl;

int test;

int ttt[50]={2,2,2,2,3,2,4,5,6,2};

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

{

//cin >> test;

//L->pData[i] = test;

L->pData[i]=ttt[i];

L->length+=1;

}

cout << "Floating TIP CreateList is Successfully runned"<<endl;

return true;

}

bool DestroyList(SqeList \*L)

{

if(L->pData != NULL )

{

free(L->pData);

L->pData = NULL;

}

L->length = 0 ;

L->size = 0;

cout << "Floating TIP DestroyList is Successfully runned"<<endl;

return true;

}

bool ClearList (SqeList \*L)

{

L->length = 0;

cout << "Floating TIP ClearList is Successfully runned"<<endl;

return true;

}

//判断是否为一个空表

bool ListEmpty (SqeList L)

{

if (L.length == 0)

{

cout << "Floating TIP ListEmpty is Successfully runned"<<endl;

return true; //表中元素的长度是0,那么就是一个空表

}

cout << "Floating TIP InitList is Successfully runned"<<endl;

return false;

}

int ListLength(SqeList L)

{

cout << "Floating TIP ListLength is Successfully runned"<<endl;

return L.length; //直接返回表的长度

}

bool GetElem (SqeList L,int i,int &e)

{

if (i<1||i>L.length) return 0;

e = L.pData[i-1];

cout << "Floating TIP CreateList is Successfully runned"<<endl;

return true;

}

//从表中找到e这个元素的位置，并返回他的位置信息

int LocateElem (SqeList L, int e)

{

for (int i = 0; i < L.length; i ++ )

{

if(L.pData[i] == e)

{

cout << "Floating TIP LocateElem is Successfully runned"<<endl;

return i+1;

}

}

return -1;

//没找到,返回-1

}

//如果cur\_e在表L中,而且不是第一个元素，那么通过pre\_e返回他前边的一个元素

bool PriorElem (SqeList L, int cur\_e,int &pre\_e)

{

for (int i = 1; i < L.length; i ++)

{

if (L.pData[i]==cur\_e)

{

pre\_e = L.pData[i-1];

return true;

}

}

//如果查找失败，返回False,pre\_e不进行操作

return false;

}

//如果cur\_e在表L中,而且不是最后一个元素，那么通过next\_e返回他后边的一个元素

bool NextElem (SqeList L, int cur\_e,int &next\_e)

{

for (int i = 0; i < L.length - 1; i ++)

{

if (L.pData[i]==cur\_e)

{

next\_e = L.pData[i+1];

return true;

}

}

//如果查找失败，返回False,pre\_e不进行操作

return false;

}

//在屏幕上打印所有的数据,并用空格隔开，最后输出一个空行

bool ListTraverse (SqeList L)

{

if(L.pData == NULL || L.length == 0)

{

return false;

}

cout << "Floating TIP Traverse this list, element is :";

for (int i = 0; i < L.length ; i ++) cout << L.pData[i] << " ";

cout << endl;

cout << "Floating TIP ListTraverse is Successfully runned"<<endl;

return true;

}

//将线性表中第i个元素用e代替，并将旧值用e返回

bool SetElem (SqeList \*L , int i, int &e)

{

if (i < 1 || i>L->length) return false; //i不在可选的范围内

int temp;

temp = L->pData[i-1];

L->pData[i-1] = e;

e = temp;

return true;

}

bool InsertElem (SqeList \*L, int i, int e)

{

if (i < 1 || i > L->length + 1) return false; //i不在可选的范围内

//插入时长度不够,就再申请一点空间

if (L->length + 1 > L->size)

{

int \*temp;

temp = (int \*)realloc (L->pData,LISTINITSIZE\*sizeof(int));

if (temp == NULL) return false;

L->pData = temp;

L->size+=LISTICRESIZE;

}

int temp;

for (int j = L->length ; j > i - 1; j--)

{

L->pData[j]=L->pData[j-1];

}

L->pData[i-1] = e;

L->length +=1;

return true;

}

bool DeleteElem (SqeList \*L, int i, int &e)

{

if (i < 1 || i>L->length) return false; //i不在可选的范围内

e = L->pData[i-1];

for (int j = i - 1; j < L->length-1;j++)

{

L->pData[j] = L->pData[j+1];

}

L->length --;

return true;

}

int main()

{

SqeList L;

int i,e,temp,location=-1;

InitList(&L);

CreateList(&L);

ListTraverse(L);

e=666,i=3,temp=999;

cout << "Floating TIP Before getting element e is "<<e<<endl;

cout << "Floating TIP Now get element locate in "<<i<<endl;

GetElem(L,i,e);

cout << "Floating TIP After getting element e is "<<e<<endl;

cout << "Floating TIP Before getting element before e temp is "<<temp<<endl;

PriorElem(L,e,temp);

cout << "Floating TIP After getting element before e temp is "<<temp<<endl;

cout << "Floating TIP Before getting element next e temp is "<<temp<<endl;

NextElem(L,e,temp);

cout << "Floating TIP After getting element next e temp is "<<temp<<endl;

cout << "Floating TIP e is "<<e<<endl;

location = LocateElem(L,e);

cout << "Floating TIP e's location is "<< location <<endl;

if (ListEmpty(L))

cout << "Floating TIP L is empty "<<endl;

else

cout << "Floating TIP L is unempty "<<endl;

cout << "Floating TIP the length of L is "<<ListLength(L)<<endl;

cout << "Floating TIP Now we will clear the list L"<<endl;

//ClearList(&L);

//cout << "Floating TIP After clear, the length of L is "<<ListLength(L)<<endl;

cout << "Floating TIP Now L is "<<ListTraverse(L)<<endl;

cout <<"Floating TIP set "<<temp<<" at the location of "<<i<<endl;

SetElem(&L,i,temp);

ListTraverse(L);

InsertElem(&L,5,888);

ListTraverse(L);

cout << "Floating TIP now we will delete the third location 3"<<endl;

DeleteElem(&L,3,e);

ListTraverse(L);

return 0;

}

链式储存

#include <iostream>

#include <stdlib.h>

using namespace std;

typedef struct LNode

{

int data; //数据域

struct LNode \*next; //指针域

}LNode,\*LinkList;

bool InitList(LinkList &L)

{

L = (LNode\*)malloc(sizeof(LNode)); //这里申请了一个头结点,链表指针指向了头结点

if(L == NULL) exit (0);

L->data = 0;

L->next=NULL;

return true;

}

bool DestroyList (LinkList &L)

{

LNode \*p=L->next,\*q=p;

L->next=NULL;

while (p)

{

p=p->next;

free(q);

q=p;

}

free (L);

L = NULL;

return true;

}

//创建一个有n个元素的链式结构的线性表

bool CreateList(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={1,2,2,3,2,2,3,2,4,2,4,3,5,6,7,8,9,2,5,4};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool ListTraverse (LinkList L)

{

if (L)

{

LNode \*p=L->next;

while (p)

{

cout << p->data << " ";

p = p->next;

}

cout << endl;

return true;

}

else

{

return false;

}

}

//清空链表

bool ClearList (LinkList &L)

{

LNode \*p=L->next,\*q=p;

while (p)

{

p = p->next;

free(q);

q=p;

}

L->next = NULL;

return true;

}

//judge list whether empty

bool ListEmpty (LinkList &L)

{

if ( L->next == NULL) return true; //if list has one node is head

else return false;

}

int ListLength(LinkList &L)

{

if(!L) return -1;

LNode \*p = L->next;

int cnt = 0;

while(p)

{

p = p->next;

cnt ++;

}

return cnt;

}

bool GetElem (LinkList &L, int pos, int &e)

{

int length = ListLength(L);

if (pos<1 || pos > length) return false; //如果数据位置非法，那么就返回操作失败的标志

LNode \*p = L->next;

for (int i=1;i<pos;i++)

{

p = p->next;

}

e = p->data;

return true;

}

int LocateElem (LinkList &L, int e)

{

LNode \*p = L->next;

int cnt = 1;

while (p)

{

if (p->data == e) return cnt;

p = p->next;

cnt ++ ;

}

return 0;

}

bool PriorElem(LinkList &L, int cur\_e, int &pre\_e)

{

LNode \*p = L->next, \*q = p;

while (p)

{

if (p->data == cur\_e && p != L->next)

{

pre\_e = q->data;

return true;

}

q = p;

p = p->next;

}

return false;

}

bool NextElem(LinkList &L, int cur\_e, int &next\_e)

{

LNode \*p = L->next;

while(p->next)

{

if(p->data == cur\_e)

{

next\_e = p->next->data;

return true;

}

p = p->next;

}

return false;

}

bool SetElem(LinkList &L, int pos ,int &e)

{

int length = ListLength (L);

if (pos<1 || pos > length )

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next;

for (int i = 1; i < pos; i++) p = p->next;

int temp = p->data;

p->data = e;

e = temp;

return true;

}

bool InsertElem(LinkList &L, int pos, int e)

{

int length = ListLength (L);

if (pos < 1 || pos > length + 1 ) //可以加在最后一个，所以长度要加一

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next,\*q =NULL;

for (int i = 2; i < pos; i ++) p = p->next; //this node is at the position of pos-1

q = (LNode \*)malloc( sizeof(LNode) );

if (pos == 1)

{

q->data = e;

q->next = L->next;

L->next = q;

}

else

{

q->data = e;

q->next = p->next;

p->next = q;

}

return true;

}

bool DeleteElem(LinkList &L, int pos, int &e)

{

int length = ListLength (L);

if (pos < 1 || pos > length ) //可以加在最后一个，所以长度要加一

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next,\*q=p;

for (int i = 2; i < pos; i ++) p=p->next; //找到pos前边的一个，就是位置是pos-1

if (pos == 1)

{

e = L->next->data;

q = L->next;

L->next = L->next->next;

free(q);

return true;

}

e = p->next->data; //e是他下一个元素的值

q = p->next;

p->next = q->next;

free(q);

return true;

}

int main()

{

LinkList L=NULL;

int e =999,temp = -111;

InitList(L);

CreateList (L,8);

ListTraverse (L);

DeleteElem(L,11,e);

ListTraverse(L);

cout <<e<<endl;

return 0;

}

2、

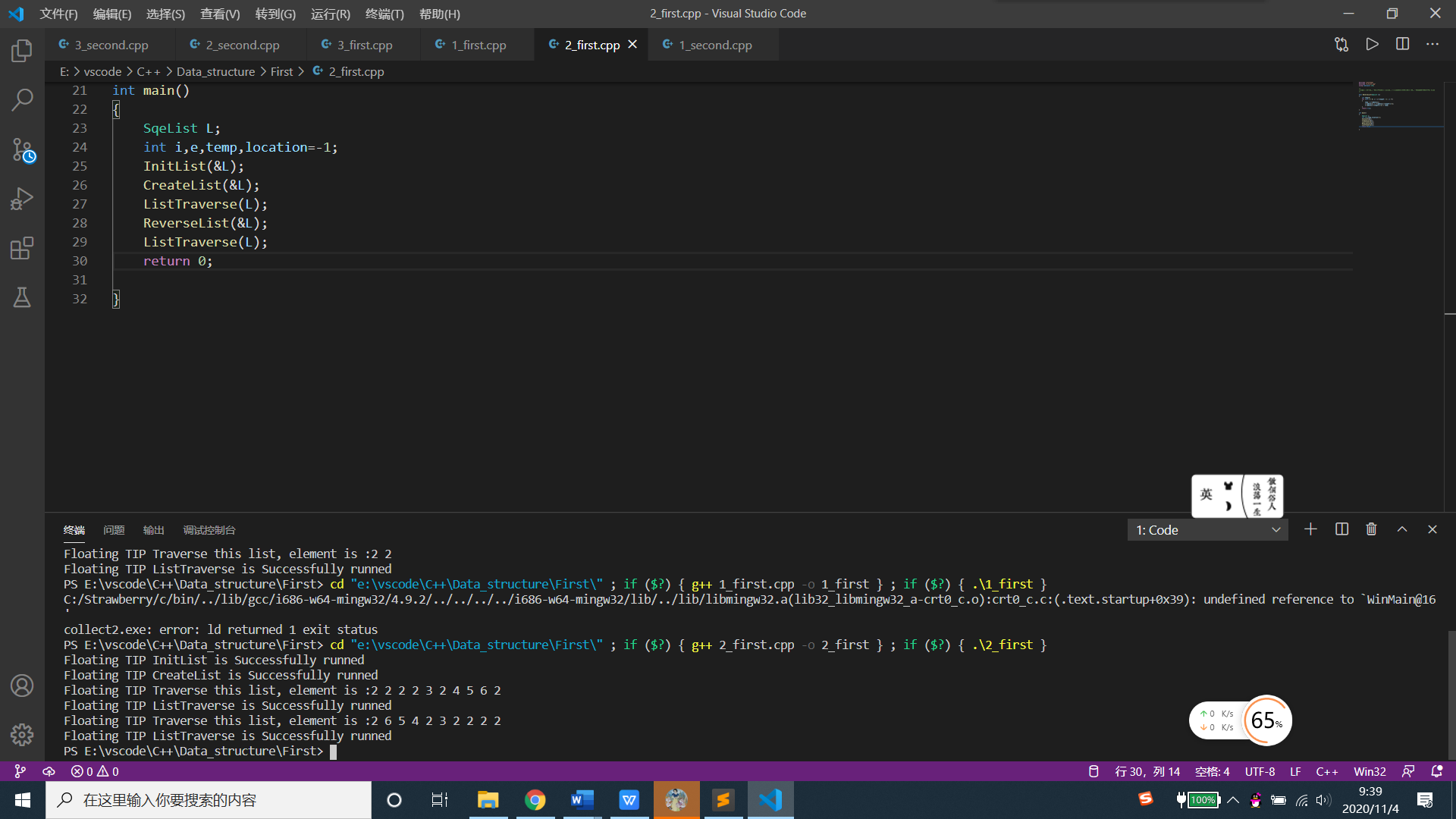
题目：设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，实现线性表的就地逆置（书P31 4）。

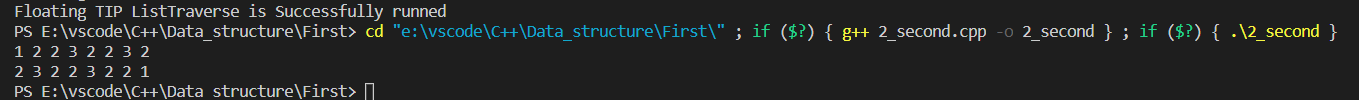
算法思想：

（1）顺序结构。可以直接进行对称，即第i个数字与第n-i个数字直接交换。

（2）链式结构。先实现在头结点之后插入一个新节点的操作，然后设置一个新的头指针，从原链表上按由头到尾的顺序进行摘除节点，并插在新链表的头结点之后。这样就实现了链表的就地逆置。

运行结果：





结果分析：结果正确，实现了线性表的转置，在一个和零个元素时程序依然正确。

附源程序。

#include <iostream>

#include "1\_first.cpp"

using namespace std;

/\*

设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，实现线性表的就地逆置（书P31 4）。

\*/

bool ReverseList(SqeList \*L)

{

int temp=0;

for (int i = 0; i < L->length / 2 ; i ++)

{

temp = L->pData[i];

L->pData[i] = L->pData[L->length-i-1];

L->pData[L->length-i-1] = temp;

}

return true;

}

int main()

{

SqeList L;

int i,e,temp,location=-1;

InitList(&L);

CreateList(&L);

ListTraverse(L);

ReverseList(&L);

ListTraverse(L);

return 0;

}

#include "1\_second.cpp"

/\*

设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，实现线性表的就地逆置（书P31 4）。

\*/

bool ReverseList(LinkList &L)

{

if (ListLength(L)<=1 || L==NULL) return true;

LinkList S=L;

LNode \*p = L->next,\*q = p ;

S->next = NULL;

while (p)

{

q = p->next;

p->next = S->next;

S->next = p;

p = q;

}

return true;

}

int main()

{

LinkList L=NULL;

int e =999,temp = -111;

InitList(L);

CreateList (L,8);

ListTraverse (L);

ReverseList(L);

ListTraverse (L);

DestroyList(L);

ReverseList(L);

ListTraverse(L);

return 0;

}

注意:以上两个函数的测试均调用了第一题线性表的文件，注意测试时不要有两个主函数，应注释掉引用文件中的主函数。

......

3、

题目：设线性表L，元素值为整型的且存在相同值，分别采用顺序结构和链式结构存储，编写函数，利用原空间，删除重复的元素值。

算法思想:

1. 顺序结构。

设置两个指针p1和p2，p1之前维护一个没有重复数字的表，p1指向表的下一个空位,p2指向现在被判定的数字，p2指向一个新的数字时，在表头到p1之间循环查找与p2指向数字相同的，如果有相同的数字，那么p2下移，即去除了当前这个数字，如果没有相同的数字，把p2指向的数字拷贝到p1当前的位置，p1下移，p2下移。

1. 链式结构。

遍历链表中所有的结点，每个结点查找其之前有无相同数字的结点，有则释放掉当前结点，没有则继续查找，复杂度O(n^2)。

注意：表的长度是1或者0时候的特判，直接返回不重复的表。

附源程序:

/\*3．设线性表L，元素值为整型的且存在相同值，分别采用顺序结构和链式结构存储，编写函数，利用原空间，删除重复的元素值。\*/

#include <iostream>

#include "1\_first.cpp" //引入线性结构的表

using namespace std;

bool DeleteRepeteList(SqeList \*L)

{

if (L->length == 0 || L->length == 1) return true;//有零个或者一个元素，那么不会有重复的数据

int p1=1,p2=1;

bool flag=false;

while (p2!=L->length)

{

flag = false;

for (int i = 0; i < p1; i ++)

{

if(L->pData[i] == L->pData[p2])

{

flag = 1; //找到相等的了

break;

}

}

if(flag)

{

p2++;

continue;

}

else

{

L->pData[p1] = L->pData[p2];

p1 ++;

p2 ++;

continue;

}

}

L->length = p1;

return true;

}

int main()

{

SqeList L;

int i,e,temp,location=-1;

InitList(&L);

CreateList(&L);

ListTraverse(L);

DeleteRepeteList(&L);

ListTraverse(L);

}

/\*3．设线性表L，元素值为整型的且存在相同值，分别采用顺序结构和链式结构存储，编写函数，利用原空间，删除重复的元素值。\*/

#include <iostream>

#include "1\_second.cpp"

using namespace std;

bool DeleteRepeteList(LinkList &L)

{

int length = ListLength(L) ;

if (length == 0 || length == 1)

{

return true;

//0 或者 1,直接返回排序成功

}

LNode \*p = L->next, \*q = p,\*r = p;

while (p)

{

r = p;

q = p->next;

while(q!=NULL)

{

if (q->data == p->data)

{

r->next = q->next;

free(q);

q = r;

}

r = q;

q = q->next;

}

p = p->next;

}

return true;

}

int main()

{

LinkList L=NULL;

int e =999,temp = -111;

InitList(L);

CreateList (L,15);

ListTraverse (L);

DeleteRepeteList(L);

ListTraverse(L);

}

4.

题目:

问题描述：一次放学的时候，小明已经规划好了自己回家的路线，并且能够预测经过各个路段的时间。同时，小明通过学校里安装的“智慧光明”终端，看到了**出发时刻路上**经过的所有红绿灯的指示状态。请帮忙计算小明此次回家所需要的时间。

输入格式：

输入的第一行包含空格分隔的三个正整数 r、y、g，表示红绿灯的设置。这三个数均不超过 106。  
 输入的第二行包含一个正整数 n，表示小明总共经过的道路段数和路过的红绿灯数目。  
 接下来的 n 行，每行包含空格分隔的两个整数 k、t。k=0 表示经过了一段道路，将会耗时 t 秒，此处 t 不超过 106；k=1、2、3 时，分别表示**出发时刻**，此处的红绿灯状态是红灯、黄灯、绿灯，且倒计时显示牌上显示的数字是 t，此处 t 分别不会超过 r、y、g。

输出格式：

输出一个数字，表示此次小明放学回家所用的时间。

算法思想：

1. 总时间来自两个分量，一个是通过道路时直接加入的时间，另一个是到一个路口遇到红灯或者黄灯等待的时间。
2. 我们已知在出发时刻，当前路口路灯的情况，那么我们维护一个总时间SumTime，在到达这个路口时路灯的情况就是经过了SumTime后路灯的情况。
3. 根据到达该路口时路灯的情况，我们能够轻易的计算出在该路口需要等待的时间。
4. 计算技巧:用SumTime对一个信号灯完整周期（红灯+黄灯+绿灯）时间取模，得到的时间如果大于当前t[i]，那么判断新的灯情况，小于t[i]依然为当前的灯，只是等待的时间变了。

附源程序:

#include <iostream>

#include <fstream>

using namespace std;

int r,y,g;

int n;

int k[200],t[200];

int main()

{

fstream infile("1.txt",ios::in);

infile>>r>>y>>g;

infile>>n;

for (int i = 1; i <= n; i ++)

infile>>k[i]>>t[i];

int ans=0,temp=0;

int all = r+y+g;

for (int i = 1; i <= n; i ++)

{

if (k[i] == 0)

{

ans+=t[i];

continue;

}

if (ans>t[i])

{

temp = ans - t[i];

temp %= all;

}

else

{

if (k[i] == 1)

{

ans += (t[i] - ans);

}

else if (k[i] == 2)

{

ans += r+(t[i]-ans);

}

continue; //绿灯不加

}

if (k[i]==1)

{

if (temp>g&&temp<g+y) ans += (all-temp);

else if (temp>g+y) ans+=(all-temp);

}

else if (k[i] == 2)

{

if (temp>r+g) ans+=(all-temp+r);

else if (temp<r) ans+=(r-temp);

}

else if (k[i]==3)

{

if (temp < y+r) ans+=(r+y-temp);

}

}

cout <<ans;

return 0;

}

5.题目

问题描述：近来，跳一跳这款小游戏风靡全国，受到不少玩家的喜爱。  
 简化后的跳一跳规则如下：玩家每次从当前方块跳到下一个方块，如果没有跳到下一个方块上则游戏结束。  
 如果跳到了方块上，但没有跳到方块的中心则获得1分；跳到方块中心时，若上一次的得分为1分或这是本局游戏的第一次跳跃则此次得分为2分，否则此次得分比上一次得分多两分（即连续跳到方块中心时，总得分将+2，+4，+6，+8...）。  
 现在给出一个人跳一跳的全过程，请你求出他本局游戏的得分（按照题目描述的规则）。

输入格式：

输入包含多个数字，用空格分隔，每个数字都是1，2，0之一，1表示此次跳跃跳到了方块上但是没有跳到中心，2表示此次跳跃跳到了方块上并且跳到了方块中心，0表示此次跳跃没有跳到方块上（此时游戏结束）。

输出格式：

输出一个整数，为本局游戏的得分（在本题的规则下）。

算法思想:维护一个数组a[i]，记录当前次的得分，那么当跳在边缘时，a[i]=1，跳在中心时，a[i]=a[i-1]+2或者2，根据题目描述可以区分这两种情况，输出记录数组的和即可。

附源程序:

#include <iostream>

#include <string.h>

using namespace std;

int a[50];

int main()

{

int temp = 1,cnt=0,ans = 0;

cin>>temp;

while (temp!=0)

{

cnt ++;

a[cnt] = temp; //从1开始

if (temp == 1)

{

a[cnt] = 1;

}

else if (temp == 2)

{

if(cnt == 1 || a[cnt-1] == 1)

{

a[cnt] = 2;

}

else

{

a[cnt] = a[cnt-1]+2;

}

}

ans += a[cnt];

cin>>temp;

}

cout << ans<<endl;

return 0;

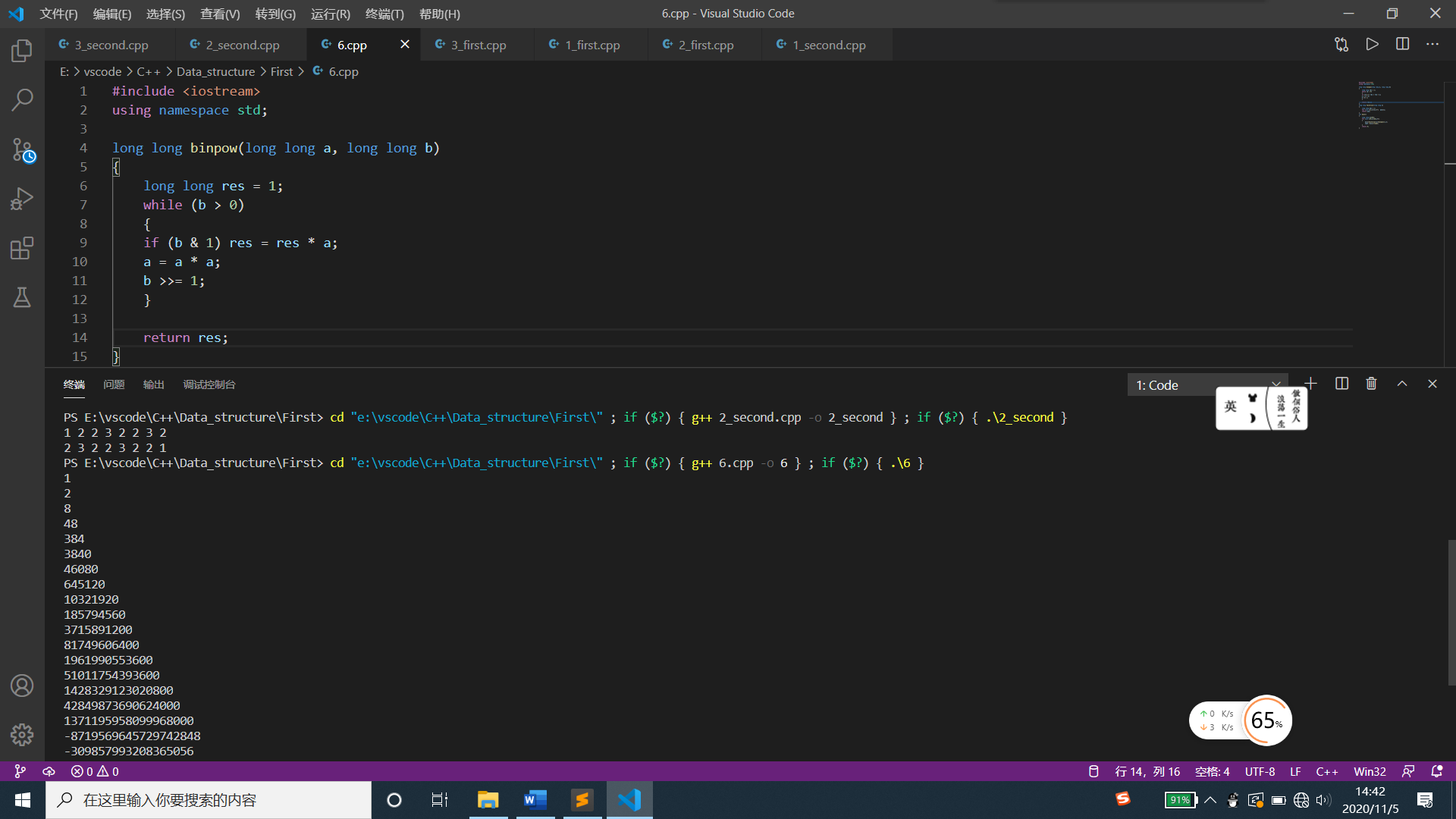
}

6.

题目:试编写算法，计算i!\*2^i(i=0,1,2…n-1)

算法思想:编写两个函数，计算i的阶乘和2^i的值，阶乘可以直接循环，2^n采用快速幂。

运行结果:



附源程序:

#include <iostream>

using namespace std;

long long binpow(long long a, long long b)

{

long long res = 1;

while (b > 0)

{

if (b & 1) res = res \* a;

a = a \* a;

b >>= 1;

}

return res;

}

long long factorial(long long n)

{

long long ans = 1;

for (int i=1;i<=n;i++) ans\*=i;

return ans;

}

int main()

{

long long a[256];

for (int i=0;i<=20;i++)

{

a[i]=factorial(i)\*binpow(2,i);

cout <<a[i]<<endl;

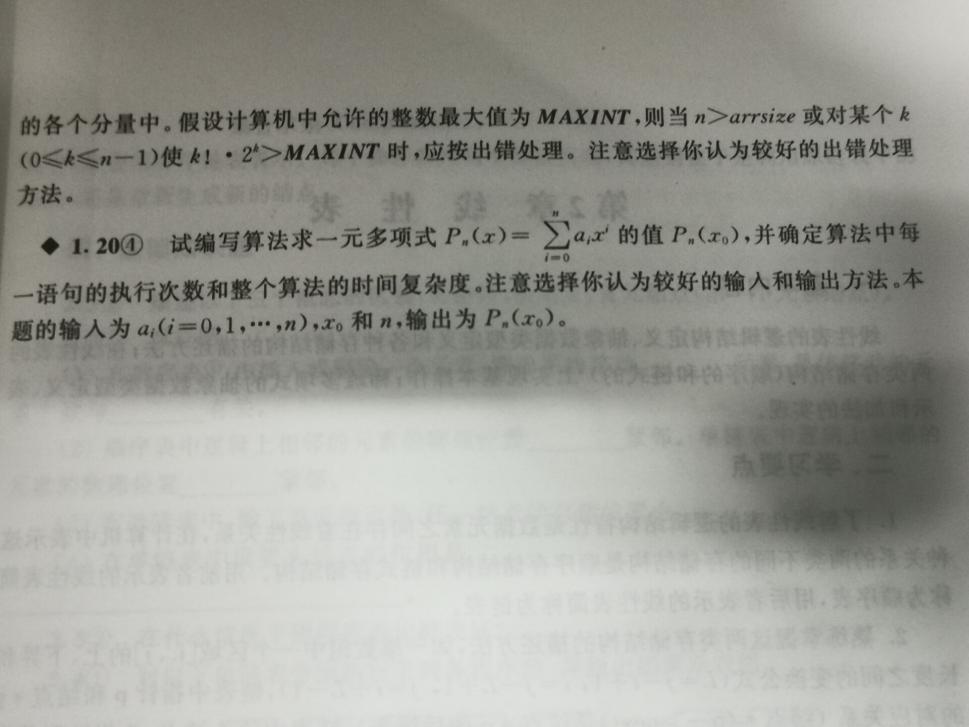
}

return 0;

}

7.

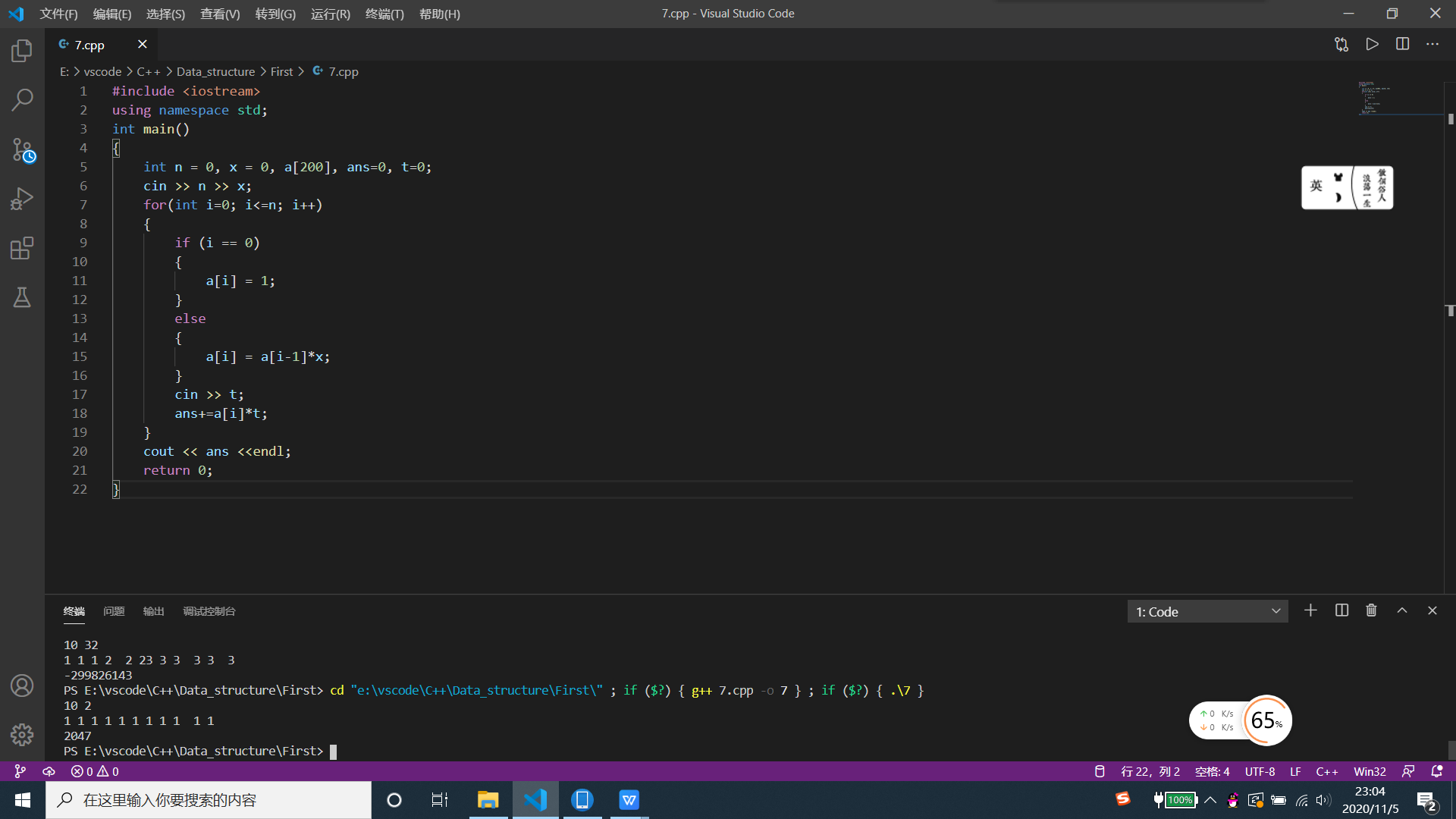
题目:



算法思想:

因为x的n次幂是连续的，所以可以边算x的i次幂边输入当前的系数，x的i次幂就是x的n-1次幂乘以x，输入的系数与之相乘就能得到该项的结果，所有结果累加起来就是最后的计算结果。

运行结果:



附源程序:

#include <iostream>

using namespace std;

int main()

{

int n = 0, x = 0, a[200], ans=0, t=0;

cin >> n >> x;

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

{

if (i == 0)

{

a[i] = 1;

}

else

{

a[i] = a[i-1]\*x;

}

cin >> t;

ans+=a[i]\*t;

}

cout << ans <<endl;

return 0;

}

二、未调试成功程序及说明

1、

题目：暂无

算法思想：

错误原因：

附源程序:

2、

题目：

算法思想：

错误原因：

附源程序。

......

三、代码行数及小结

共七个文件，代码行数共800