

《数据结构专周设计》

课程设计

**学年学期： 2023-2024上学期**

**院 别： 计算机科学与数学学院**

**专 业： 计算机科学与技术**

**班 级： 计算机2204**

**学 生： 蔡琓莹**

**学 号： 3221311409**

**指导教师： 姜春茂**

目录

[1. A类必做 3](#_Toc23660)

[1.1 排序题 4](#_Toc30845)

[1.1.2 题目的关键代码 4](#_Toc23135)

[1.1.3 流程图（如果有） 8](#_Toc19622)

[1.1.4 程序的运行截图 8](#_Toc31138)

[1.1.1 题目的解题思路 9](#_Toc30708)

[1.快速排序：通过一趟排序将要排序的数据分割成独立的两部分。 9](#_Toc1310)

[1.2 散列表 9](#_Toc15727)

[1.2.2 题目的关键代码 9](#_Toc19583)

[1.2.4 程序的运行截图 12](#_Toc12535)

[1.2.1 题目的解题思路 12](#_Toc24967)

[2 B类（较难） 12](#_Toc15648)

[2.1 你选择的题目1 12](#_Toc3309)

[2.2.1 题目的解题思路 12](#_Toc13932)

[2.2.2 题目的关键代码 14](#_Toc27946)

[2.2.3 流程图（如果有） 26](#_Toc30060)

[2.2.4 程序的运行截图 26](#_Toc17570)

[3 C类题目 29](#_Toc14860)

[3.1.1题目的解题思路 29](#_Toc14427)

[3.1.2题目的关键代码 30](#_Toc9558)

[3.1.3 程序的运行截图 61](#_Toc1374)

# 1. A类必做

## 1.1 排序题

### 1.1.2 题目的关键代码

#### 堆排序

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void buildHeap(int arr[], int n, int i) {

int largest = i;

int left = 2\*i + 1;

int right = 2\*i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(&arr[i], &arr[largest]);

buildHeap(arr, n, largest);

}

}

void heapSort(int arr[], int n) {

for (int i = n/2 - 1; i >= 0; i--)

buildHeap(arr, n, i);

for (int i = n-1; i > 0; i--) {

swap(&arr[0], &arr[i]);

buildHeap(arr, i, 0);

}

}

#### 归并排序：

void merge(int arr[], int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (i = 0; i < n1; i++) {

L[i] = arr[left + i];

}

for (j = 0; j < n2; j++) {

R[j] = arr[mid + 1 + j];

}

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

#### 计数排序：

void countSort(int arr[], int n) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

int count[max + 1];

for (int i = 0; i <= max; ++i) {

count[i] = 0;

}

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

count[arr[i]]++;

}

for (int i = 1; i <= max; i++) {

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

}

int output[n];

for (int i = n - 1; i >= 0; i--) {

output[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

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

arr[i] = output[i];

}

}

void printArray(int arr[], int n) {

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

printf("%d ", arr[i]);

}

printf("\n");

}

#### 快速排序：

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition (int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

#### 希尔排序：

void shell(int s[],int n)

{

int i,j,gap,tmp;

for(gap=n/2;gap>0;gap/=2)

{

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

{

tmp=s[i];

for(j=i;j>=gap&&s[j-gap]>tmp;j-=gap)

{

s[j]=s[j-gap];

}

s[j]=tmp;

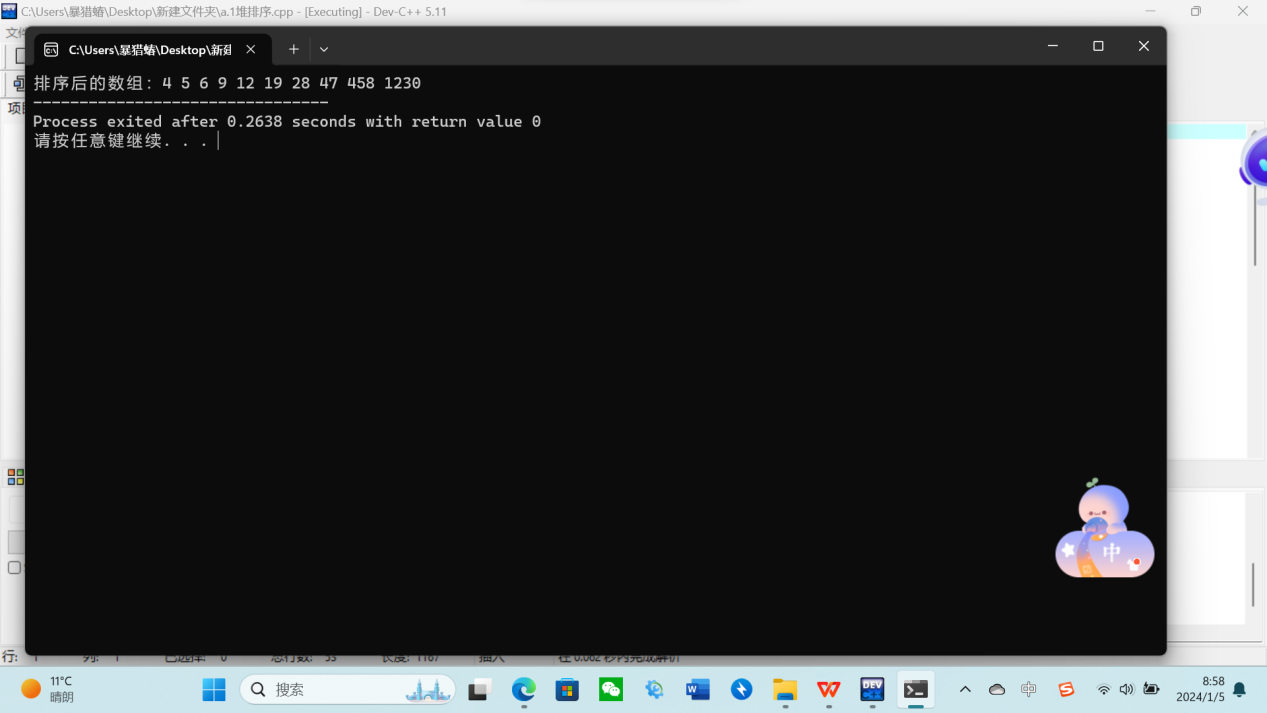
}

}

}

### 1.1.3 流程图（如果有）

### 1.1.4 程序的运行截图



### 1.1.1 题目的解题思路

1.快速排序：通过一趟排序将要排序的数据分割成独立的两部分。

其中一部分的所有数据都比另外一部分的所有数据都要小。

然后再按此方法对这两部分数据分别进行快速排序，整个排序可以递归进行，一次整个数据变成有序序列

2.希尔排序：记录按下标的一定增量分组，对每组使用直接插入排序算法排序，分割待排序记录的目的是减少待排序记录的个数，并使整个序列向基本有序发展。需要采取跳跃分割的策略，将相距某个“增量”的记录组成一个子序列

3.堆排序：首先将待排序的数组构造成一个大根堆，此时，整个数组的最大值就是堆结构的顶端

，然后将顶端的数与末尾的数交换，此时，末尾的数为最大值，剩余待排序数组个数为n-1

，将剩余的n-1个数再构造成大根堆，再将顶端数与n-1位置的数交换，如此反复执行，便能得到有序数组

4.计数排序：根据待排序集合中最大元素和最小元素的差值范围，申请额外空间，遍历待排序集合，将每一个元素出现的次数记录到元素值对应的额外空间内；对额外空间内数据进行计算，得出每一个元素的正确位置；将待排序集合每一个元素移动到计算得出的正确位置上。

5.归并排序：将待排序的线性表不断地切分成若干个子表，直到每个子表只包含一个元素，这时，可以认为只包含一个元素的子表是有序表。将子表两两合并，每合并一次，就会产生一个新的且更长的有序表，重复这一步骤，直到最后只剩下一个子表，这个子表就是排好序的线性表。

## 1.2 散列表

### 1.2.2 题目的关键代码

// 哈希表结构

typedef struct {

HashNode nodes[SIZE];

} HashTable;

HashTable table;

// 哈希函数1

int hashFunc1(char \*phone) {

int sum = 0;

int len = strlen(phone);

for (int i = 0; i < len; i++) {

sum += phone[i];

}

return sum % SIZE;

}

// 在散列表中插入节点

void insertNode(HashTable \*table, char \*phone, char \*name, char \*address) {

int index;

HashNode \*node =(HashNode \*) malloc(sizeof(HashNode));

strcpy(node->phone, phone);

strcpy(node->user.name, name);

strcpy(node->user.address, address);

// 使用拉链法处理冲突

index = hashFunc1(phone);

while (strcmp(table->nodes[index].phone, "") != 0) {

index = (index + 1) % SIZE;

}

table->nodes[index] = \*node;

}

// 根据电话号码查找节点

UserInfo \*searchNode(HashTable \*table, char \*phone) {

int index;

index = hashFunc1(phone);

int i = 0;

while (strcmp(table->nodes[(index + i) % SIZE].phone, phone) != 0) {

i++;

if (i >= SIZE) {

return NULL;

}

}

return &(table->nodes[(index + i) % SIZE].user);

}

int collisionCount = 0; // 冲突计数，用于计算冲突率

void linearProbeInsert(char\* key, UserInfo user) {

int index = hashFunc1(key);

while (table.nodes[index].phone[0] != '\0') {

index = (index + 1) % SIZE;

collisionCount++;

}

strcpy(table.nodes[index].phone, key);

table.nodes[index].user = user;

}

int search(char\* key) {

int index = hashFunc1(key);

int count = 0;

if (strcmp(table.nodes[index].phone, key) == 0)

return 1;

while (table.nodes[index].phone[0] != '\0' && strcmp(table.nodes[index].phone, key) != 0) {

index = (index + 1) % SIZE;

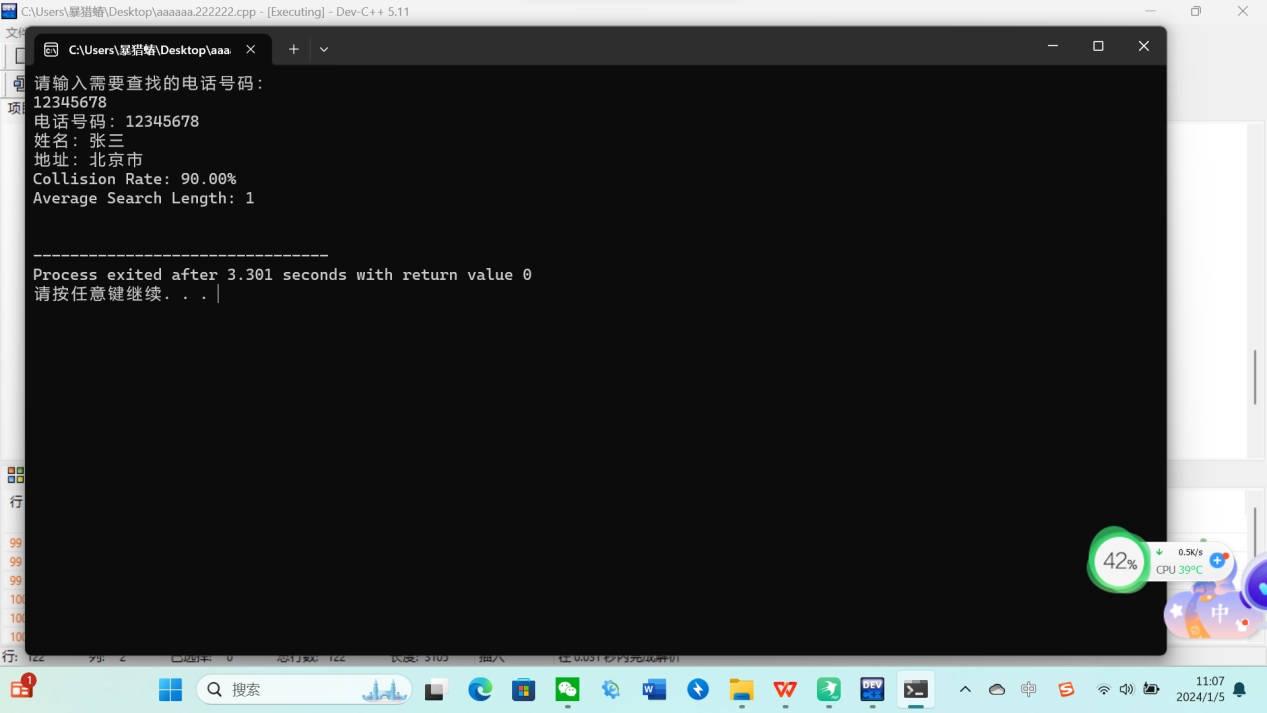
count++;

}

return count;

}

### 1.2.4 程序的运行截图



### 1.2.1 题目的解题思路

构建[哈希表](https://so.csdn.net/so/search?q=%E5%93%88%E5%B8%8C%E8%A1%A8&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/liuyuchen282828/article/details/_blank)（需要构造一个哈希函数，确定一个合适的处理冲突的方法）按照电话号码查找并输出  
将哈希表按序存进文件中

# 2 B类（较难）

## 2.1 你选择的题目1

本节我选择的题目是 4 5 7 9 12 1，分别涉及到栈稀疏矩阵递归，栈和树，排序，链表顺序表查找，查找，递归和查找

### 2.2.1 题目的解题思路

#### 2.1.1

枚举字符串的始位置和末位值，再判断始位置和末位值之间的字符串是否为回文串

#### 2.1.4

先将第一个皇后放在第一行第一列，然后将第二个皇后放在第二行第一列，判断该种摆法是否符合要求。很明显这样摆不行，有两个皇后会在同一列。再将第二个皇后放在第二行第二列，这样也不行，两个皇后会在一条斜线上。将第二个皇后放在第二行第三列，这样满足当前条件。

目前已经有两个皇后满足条件，接下来放第三个，还是从第三行第一列开始放置，不满足条件再放第二列，第三列…一直到第8个皇后也能放在一个不冲突的位置，此时找到一个符合要求的解。

然后我们开始回溯，将第一个皇后放在第一行第二列，后面的就继续按上面的方式循环，一直到回溯完毕，找出所有符合条件的解为止。

#### 2.1.5

前序遍历：根结点 ---> 左子树 ---> 右子树

中序遍历：左子树---> 根结点 ---> 右子树

后序遍历：左子树 ---> 右子树 ---> 根结点

#### 2.1.7

先选定一个根结点，并选定一个数组，先确定未遍历前的初始距离，把距离最短的邻接结点选定为中间结点，并标记访问过，开始往下遍历，挨个访问那个中间结点的邻接结点。计算出根结点到中间结点+中间结点到新邻接结点的距离，作为新距离，对比新距离和旧距离，如果新距离大，则把新距离替换掉旧距离，否则不变。

一轮访问结束后，从未标记的结点中选定距离最短的，把它作为中间结点，继续往下访问。若都标记过，则算法结束。

#### 2.1.9

． 建立AOE网的存储结构。

拓扑排序，并求得ve[]。从源点V0出发，令ve[0]=0,按拓扑有序求其余各顶点的最早发生时间ve[i]。如果得到的拓扑有序序列中顶点个数小于网中顶点数n，则说明网中存在环，不能求关键路径，算法终止；否则执行步骤3。

拓扑逆序，求得vl[]。从汇点Vn出发，令vl[n-1] = ve[n-1]，按逆拓扑有序求其余各顶点的最迟发生时间vl[i]。

求得关键路径。根据各顶点的ve和vl值，求每条弧s的最早开始时间e(s)和最迟开始时间l(s)。若某条弧满足条件e(s) = l(s)，则为关键活动

#### 2.1.12

字符串的匹配和提取

### 2.2.2 题目的关键代码

#### 2.1.1

、int isprime(char s[],int i,int j){

while(i<j)

{

if(s[i]!=s[j]){

return 0;

}

i++;

j--;

}

if(i>=j){

return 1;

}

}

int main()

{

char s[1000];

char max[1000];

char t[1000];

gets(s);

int l=strlen(s),maxl=1,m;

int i,j,k;

if(l<2){//如果长度为1；直接输出

printf("%s",s);

}

for(i=0;i<=l;i++){

for(j=i+1;j<l;j++){

int num=0;

for(k=i;k<=j;k++){

t[num++]=s[k];

}

t[num]='\0';

int lt=strlen(t);

if(isprime(s,i,j)==1&&lt>maxl){ strcpy(max,t);

maxl=lt;

}

strcpy(t,"");

}

}

printf("%s",max);

return 0;

}

#### 2.1.4

int isTrue(int a, int b)

{

int t;

for(t=a-1; t>=0; t--)

{

if(ms[t][b] == 1)

return 0;

}

for(t=b-1; t>=0; t--){

if(ms[a][t] == 1)

return 0;

}

int m, n;

for(m=a-1,n=b-1; m>=0&&n>=0; m--,n--)

{

if(ms[m][n] == 1)

return 0;

}

for(m=a-1,n=b+1; m>=0&&n<length; m--,n++)

{

if(ms[m][n] == 1)

return 0;

}

return 1;

}

int a, b;

for(a=0; a<length; a++)

{

for(b=0; b<length; b++)

{

if(ms[a][b] == 0)

{

printf("\* ");

}

else if(ms[a][b] == 1)

{

printf("# ");

}

}

printf("\n");

}

printf("\n\n");

}

void fun(int m){

if(m == length)

{

print();

return;

}

int i;

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

{

if(isTrue(m, i))

{

ms[m][i] = 1;

fun(m+1);

ms[m][i] = 0;

}

}

}

#### 2.1.5

typedef struct Node {

int value;

int left;

int right;

} Node;

typedef struct Stack {

int capacity;

int top;

Node \*\*array;

} Stack;

Stack\* createStack(int capacity) {

Stack \*stack = (Stack\*) malloc(sizeof(Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (Node\*\*) malloc(capacity \* sizeof(Node\*));

return stack;

}

int isEmpty(Stack \*stack) {

return stack->top == -1;

}

int isFull(Stack \*stack) {

return stack->top == stack->capacity - 1;

}

void push(Stack \*stack, Node \*item) {

if (isFull(stack)) {

return;

}

stack->array[++stack->top] = item;

}

Node\* pop(Stack \*stack) {

if (isEmpty(stack)) {

return NULL;

}

return stack->array[stack->top--];

}

Node\* peek(Stack \*stack) {

if (isEmpty(stack)) {

return NULL;

}

return stack->array[stack->top];

}

void preorderTraversal(Node \*nodes, int n) {

if (n <= 0) {

return;

}

Stack \*stack = createStack(n);

push(stack, &nodes[0]);

while (!isEmpty(stack)) {

Node \*current = pop(stack);

printf("%d ", current->value);

if (current->right != -1) {

push(stack, &nodes[current->right]);

}

if (current->left != -1) {

push(stack, &nodes[current->left]);

}

}

printf("\n");

}

void inorderTraversal(Node \*nodes, int n) {

if (n <= 0) {

return;

}

Stack \*stack = createStack(n);

int currentIndex = 0;

while (!isEmpty(stack) || currentIndex != -1) {

while (currentIndex != -1) {

push(stack, &nodes[currentIndex]);

currentIndex = nodes[currentIndex].left;

}

if (!isEmpty(stack)) {

Node \*current = pop(stack);

printf("%d ", current->value);

currentIndex = current->right;

}

}

printf("\n");

}

void postorderTraversal(Node \*nodes, int n) {

if (n <= 0) {

return;

}

Stack \*stack1 = createStack(n);

Stack \*stack2 = createStack(n);

push(stack1, &nodes[0]);

while (!isEmpty(stack1)) {

Node \*current = pop(stack1);

push(stack2, current);

if (current->left != -1) {

push(stack1, &nodes[current->left]);

}

if (current->right != -1) {

push(stack1, &nodes[current->right]);

}

}

while (!isEmpty(stack2)) {

Node \*current = pop(stack2);

printf("%d ", current->value);

}

printf("\n");

}

#### 2.1.7

int minimumSteps(int array[], int N) {

int count = 0;

for (int i = 0; i < N; i++) {

if (array[i] != i + 1) {

int j = i + 1;

while (array[j] != i + 1) {

j++;

}

int temp = array[i];

array[i] = array[j];

array[j] = temp;

count++;

}

}

return count;

}

#### 2.1.9

typedef int EdgeType;

typedef char VertexType;

typedef struct EdgeNode

{

int adjvex;

EdgeType edgeData;

struct EdgeNode \*next;

}EdgeNode, \*EdgeList;

typedef struct

{

VertexType vextexData;

EdgeList edgeList;

int indegree;

EdgeType earliest;

EdgeType latest;

}VertexNode, \*VertexList;

typedef struct

{

VertexList vertexList;

int numVertexs, numEdges;

}ALGraph;

void InitALGraph(ALGraph \*G)

{

G->numVertexs = 0;

G->numEdges = 0;

G->vertexList = (VertexNode \*)malloc(MaxVex \* sizeof(VertexNode));

int i;

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

{

G->vertexList[i].edgeList = (EdgeNode \*)malloc(sizeof(EdgeNode));

G->vertexList[i].edgeList->next = NULL;

G->vertexList[i].indegree = 0;

G->vertexList[i].earliest = 0;

G->vertexList[i].latest = 0;

}

printf("已初始化邻接表\n\n");

}

void CreateALGraph(ALGraph \*G)

{

printf("请输入顶点数和边数:");

scanf("%d %d", &G->numVertexs, &G->numEdges);

int i, j, k;

EdgeType w;

for (i = 0; i < G->numVertexs; i++)

{

fflush(stdin);

printf("请输入第%d个顶点数据:", i + 1);

scanf("%c", &G->vertexList[i].vextexData);

}

for (k = 0; k < G->numEdges; k++)

{

printf("请输入边(vi, vj)上的顶点序号及权值:");

scanf("%d %d %d", &i, &j, &w);

EdgeNode \*p1;

p1 = (EdgeNode \*)malloc(sizeof(EdgeNode));

p1->edgeData = w;

p1->next = G->vertexList[i - 1].edgeList->next;

G->vertexList[i - 1].edgeList->next = p1;

p1->adjvex = j - 1;

G->vertexList[j - 1].indegree++;

}

printf("已创建邻接表!\n\n");

}

void KeyPath(ALGraph G)

{

int top1 = 0, Stack[MaxVex];

int top2 = 0, RStack[MaxVex];

int i, indegree[MaxVex];

for (i = 0; i < G.numVertexs; i++)

{

indegree[i] = G.vertexList[i].indegree;

if (indegree[i] == 0)

{

Stack[top1++] = i;

}

}

int v;

EdgeNode \*p;

printf("拓扑排序:");

while (top1 > 0)

{

i = Stack[--top1];

RStack[top2++] = i;

printf("%c--->", G.vertexList[i].vextexData);

for (p = G.vertexList[i].edgeList->next; p ; p = p->next)

{

v = p->adjvex;

indegree[v]--;

if (!indegree[v])

{

Stack[top1++] = v;

}

if (G.vertexList[i].earliest + p->edgeData > G.vertexList[v].earliest)

{

G.vertexList[v].earliest = G.vertexList[i].earliest + p->edgeData;

}

}

}

printf("\n");

if (top2 < G.numVertexs)

{

printf("排序失败!存在回路\n");

}

else

{

v = RStack[--top2];

for (i = 0; i < G.numVertexs; i++)

{

G.vertexList[i].latest = G.vertexList[v].earliest;

}

while (top2 > 0)

{

i = RStack[--top2];

for (p = G.vertexList[i].edgeList->next; p ; p = p->next)

{

v = p->adjvex;

if (G.vertexList[v].latest - p->edgeData < G.vertexList[i].latest)

{

G.vertexList[i].latest = G.vertexList[v].latest - p->edgeData;

}

}

}

}

}

void Display(ALGraph G)

{

int i;

printf("顶点\t");

for (i = 0; i < G.numVertexs; i++)

{

printf("%c\t", G.vertexList[i].vextexData);

}

printf("\n最早\t");

for (i = 0; i < G.numVertexs; i++)

{

printf("%d\t", G.vertexList[i].earliest);

}

printf("\n最迟\t");

for (i = 0; i < G.numVertexs; i++)

{

printf("%d\t", G.vertexList[i].latest);

}

printf("\n关键路径:");

for (i = 0; i < G.numVertexs; i++)

{

if (G.vertexList[i].latest - G.vertexList[i].earliest == 0)

{

printf("%C--->", G.vertexList[i].vextexData);

}

}

printf("\n");

}

#### 2.1.12

int main()

{

char s[2],ss[2];

char ch;

int i=0;

while((ch=getchar())!='\n')

{

s[i]=ch;

i++;

}

i=0;

while((ch=getchar())!='\n')

{

ss[i]=ch;

i++;

}

if(s[0]==ss[0]||ss[0]=='.')

{

if(s[1]==ss[1]||ss[1]=='\*')

printf("true\n");

else

printf("false");

}

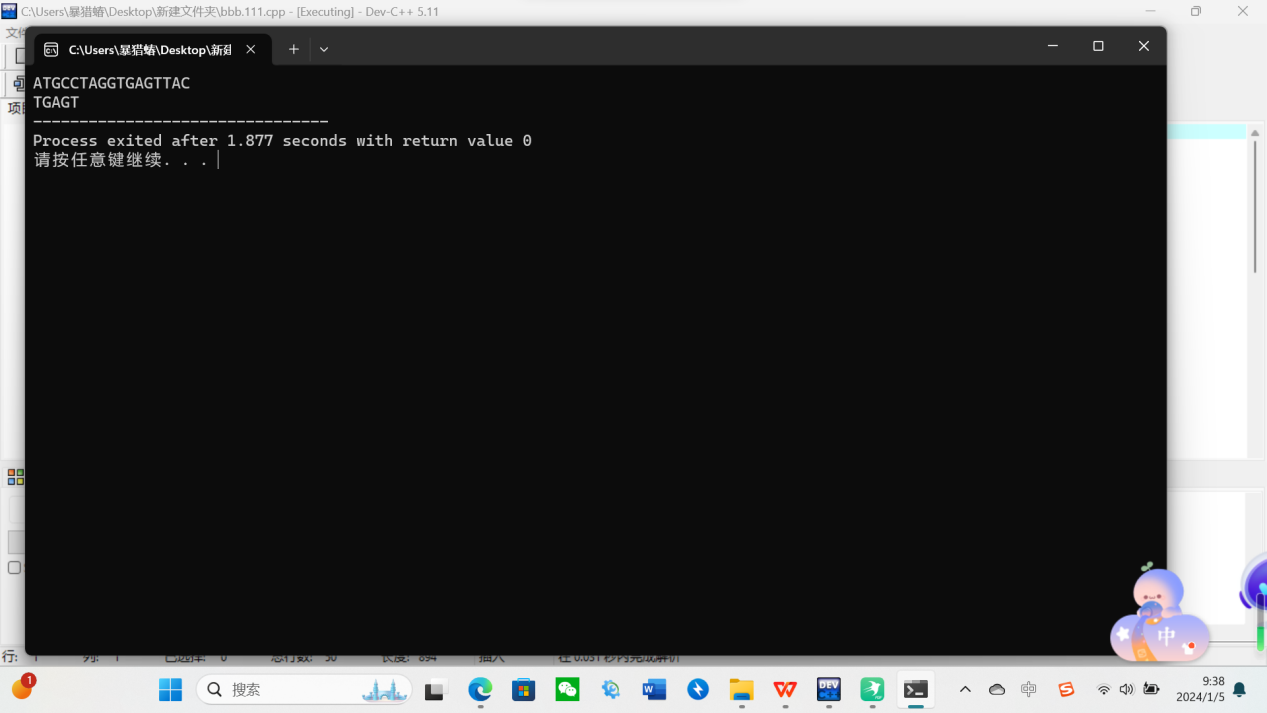
return 0;

}

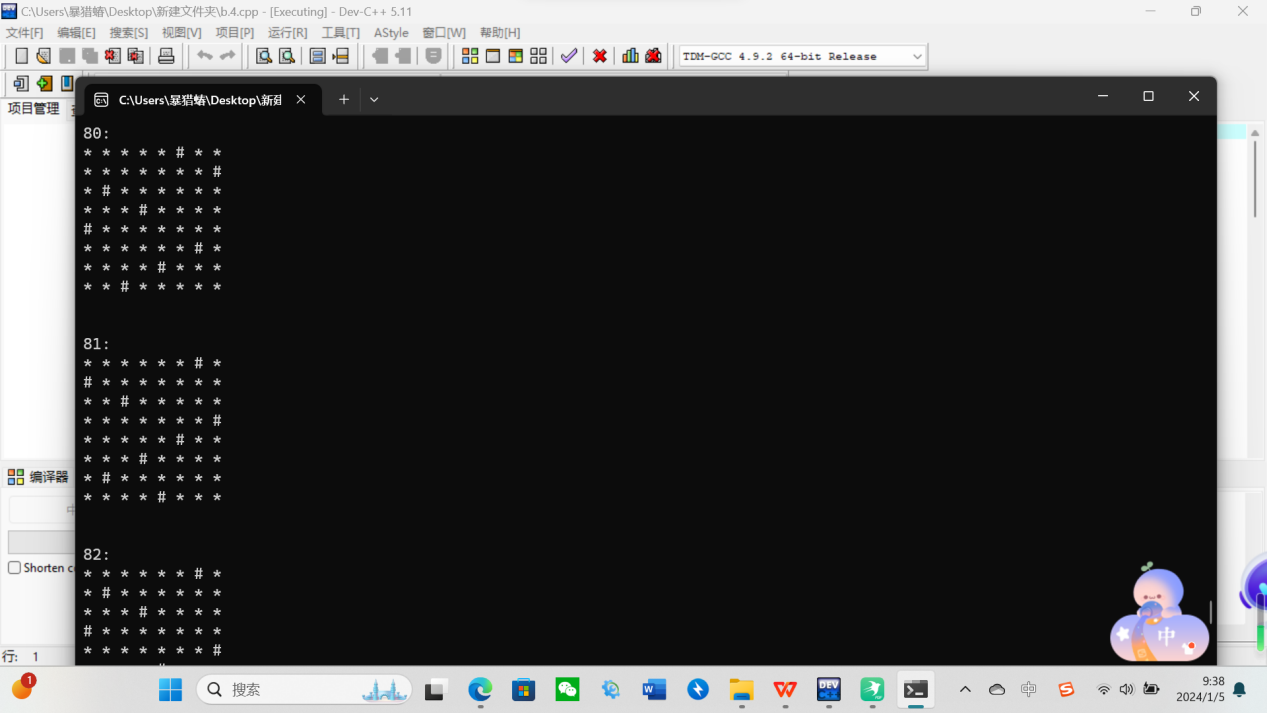
### 2.2.3 流程图（如果有）

### 2.2.4 程序的运行截图

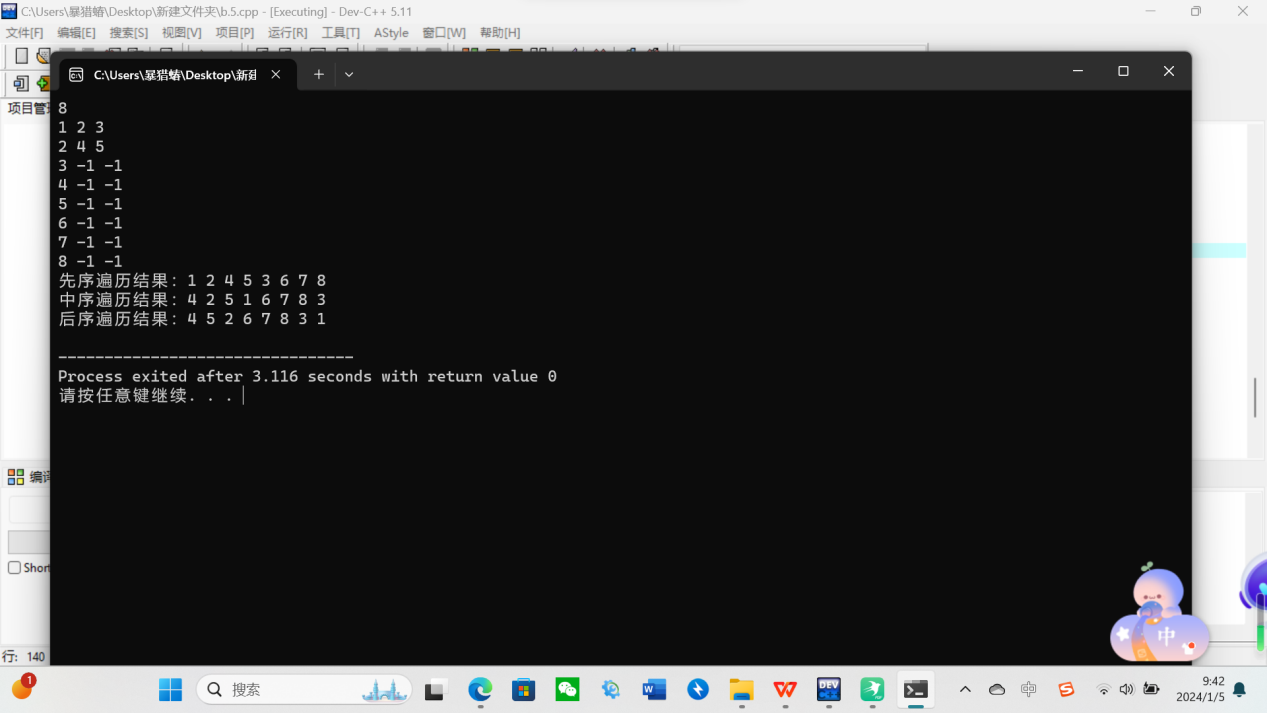
#### 2.1.1



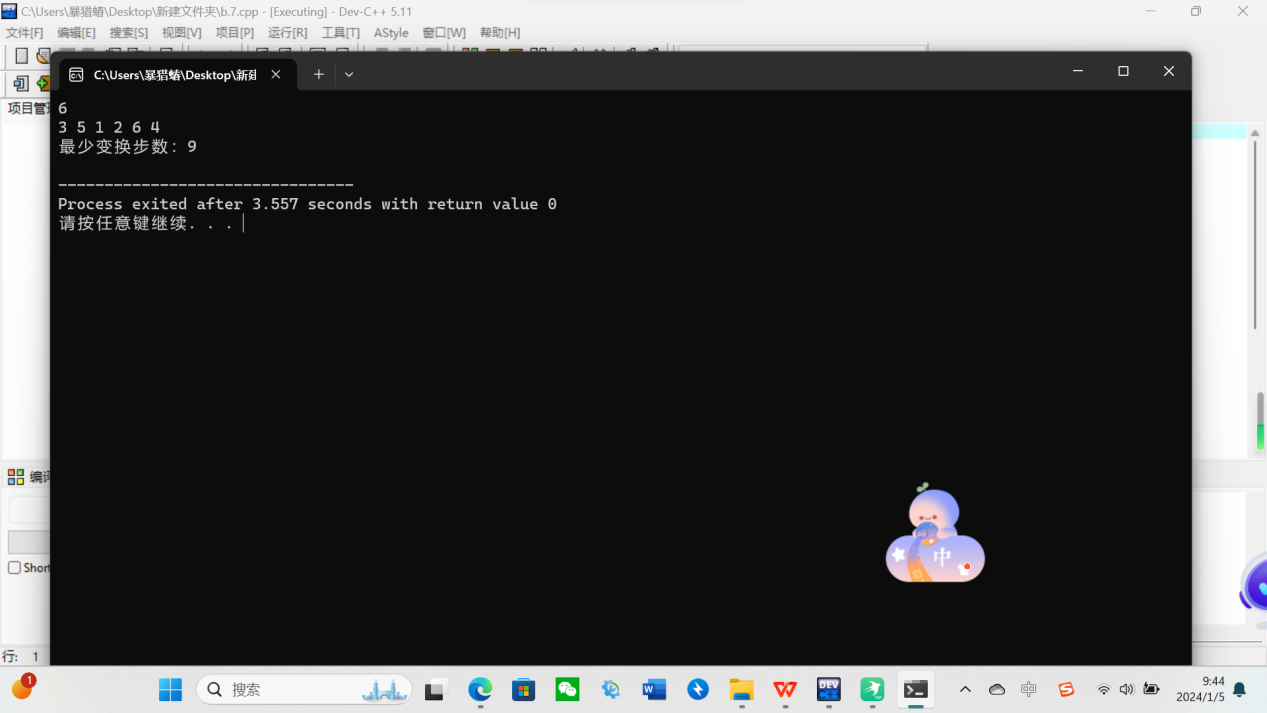
#### 2.1.4



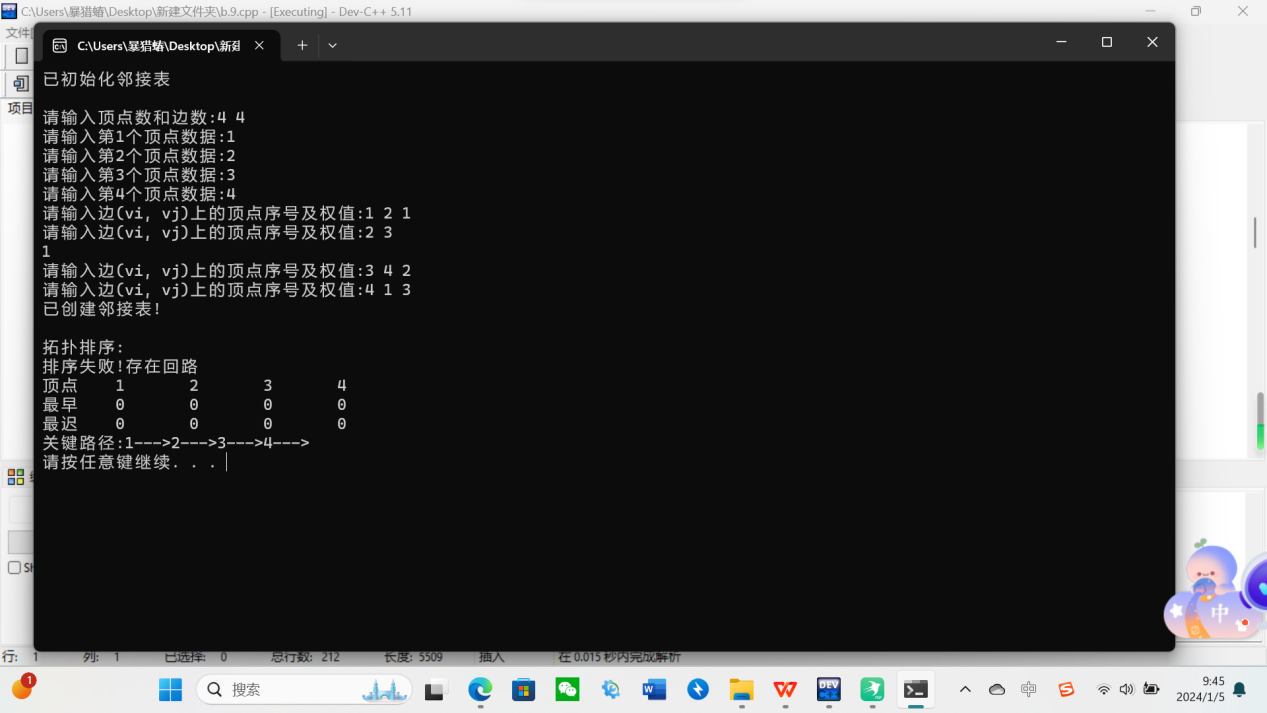
#### 2.1.5



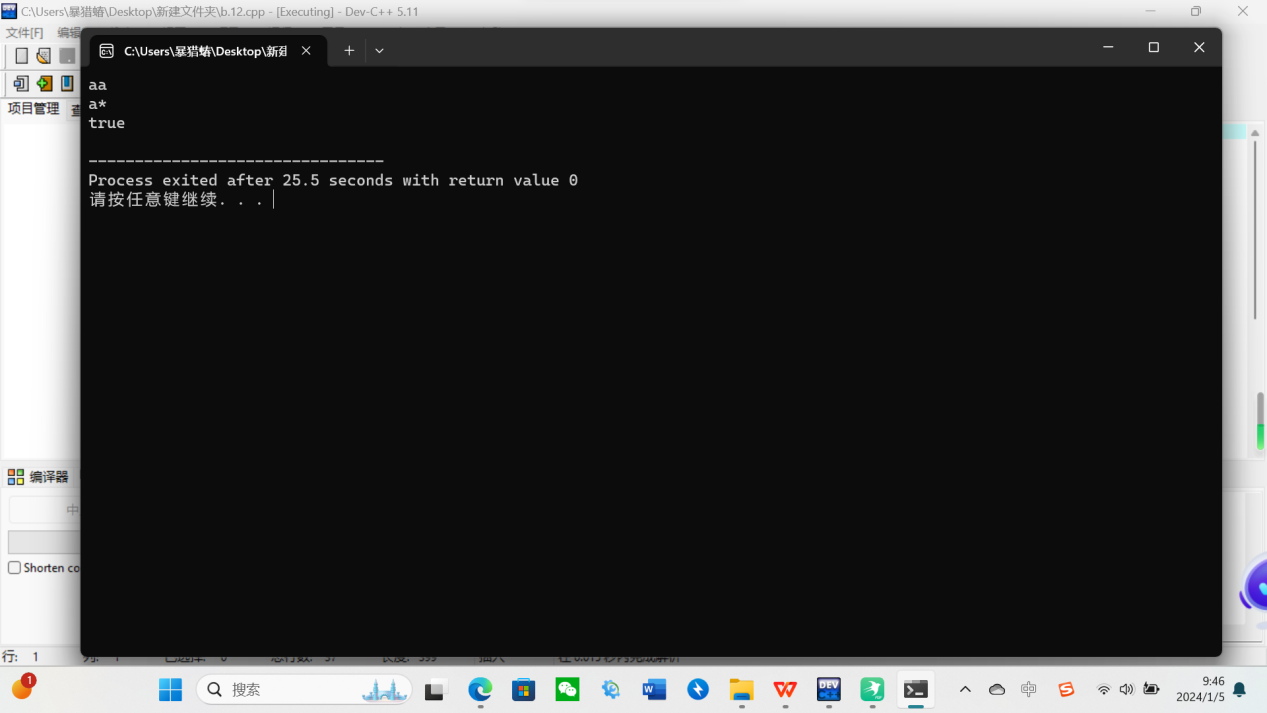
#### 2.1.7



#### 2.1.9



#### 2.1.12



# 3 C类题目

### 3.1.1题目的解题思路

#### 3.2.20

函数的参数采用单链表的引用，返回值类型为链表，返回其中一个链表。

创建一个新的头结点，并且将形参传过来的链表置空，开始遍历，设置一个计数器，如果为奇数将节点添加到链表A的尾部，如果为偶数添加到链表B的尾部。

#### 3.2.21

使用快速排序法找到中位数，相加除以2

#### 3.2.23

后进先出

#### 3.2.24

将**十进制**数**转换为其他进制**的方法是通过不断除以目标**进制**的基数，并将余数保存下来，直到商**为**0为止

#### 3.2.27

直接对三元组进行相关运算  
先对稀疏矩阵进行，生成三元组

#### 3.2.28

一种使用顺序存储，另一种使用二叉链存储，来求二叉树中两个指定节点

的最近公共祖先

#### 3.2.29

只要用中序遍历的过程是单调升序就代表是二叉搜索树

#### 3.2.31

因为二叉排序树可以看成是一个有序表，所以在二叉排序树上进行查找和折半查找类似，也是一个逐步缩小查找范围的过程。 此处我们采用递归查找。

无论是否查找成功，我们都将查找的最终地址返回，查找成功返回查到的元素的地址；查找失败则返回空地址NULL

#### 3.2.32

[深度优先搜索](https://so.csdn.net/so/search?q=%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/ff_gogo/article/details/_blank)，递归，迭代也可以实现深度优先搜索，用到了回溯。

找到叶子节点时，就生成一条路径，并放到结果集中，然后回溯到上一个节点，开始遍历上一个节点的右节点。

#### 3.2.41

在树中，每个节点都有其的深度，深度是相对于根节点而言的，而要找到x和y的LCA深度是唯一的，当x与y的深度相同时，每一次上跳父节点可保证深度是相同的，这时候只需要保证父节点重合了则找到了LCA。 所以在存储的时候，需要存储每个节点的深度，同时要存储每个节点的父节点

#### 3.2.25

利用已经部分匹配这个有效信息，保持i指针（主串）不回溯，通过修改j指针，让模式串尽量地移动到有效的位置

### 3.1.2题目的关键代码

#### 3.2.19

int main()

{

int s[100]={0};

int ss[100]={0};

int x[100]={0};

int i=0;

while(1){

scanf("%d",&s[i]);

i++;

if(getchar()=='\n')

break;

}

i=0;

while(1){

scanf("%d",&ss[i]);

i++;

if(getchar()=='\n')

break;

}

int len=i;

int j;

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

{

for(j=0;j<len;j++)

{

if(s[i]==ss[j])

printf("%d ",s[i]);

}

}

printf("\n");

int flag;

int n=0;

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

x[i]=s[i];

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

{

flag=0;

for(j=0;j<len;j++)

{

if(ss[i]!=x[j])

{

flag++;

}

if(flag==len)

{

x[len+(n++)]=ss[i];

}

}

}

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

printf("%d ",x[i]);

return 0;

}

#### 3.2.20

typedef struct s

{

int num;

struct s \*next;

}ss;

ss\*creat ()

{

ss \*head;

ss \*p,\*q;

p=q=(ss\*)malloc(sizeof(ss));

head=(ss\*)malloc(sizeof(ss));

scanf("%d",&p->num);

int i=0;

while(p->num!=0)

{

i++;

if(i==1)

{

head->next=p;

}

else

q->next=p;

q=p;

p=(ss\*)malloc(sizeof(ss));

scanf("%d",&p->num);

if(i==10)

break;

}

q->next=NULL;

return head;

}

void sep(ss \*a,ss \*&b)

{

ss \*heada=a->next;

ss \*pre=a;

b=(ss\*)malloc(sizeof(ss));

b->next=NULL;

ss \*headb=b;

ss \*dele;

while(heada!=NULL)

{

if(heada->num%2!=0)

{

pre=heada;

heada=heada->next;

}

else

{

dele=heada;

pre->next=dele->next;

heada=heada->next;

headb->next=dele;

headb=dele;

}

}

headb->next=NULL;

}

#### 3.2.21

int main()

{

int s[100]={0};

int ss[100]={0};

int i=0;

while(1){

scanf("%d",&s[i]);

i++;

if(getchar()=='\n')

break;

}

i=0;

while(1){

scanf("%d",&ss[i]);

i++;

if(getchar()=='\n')

break;

}

int t=i/2;

printf("%d\n",t);

float x=(s[t]+ss[t])/2.0;

printf("%.1f",x);

return 0;

}

#### 3.2.23

int check(int n)

{

int po=0;

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

{

for(int j=po+1;j<=out[i];j++)

{

po=j;

stack[top++]=j;

}

if(stack[--top]!=out[i])

return 0;

}

return 1;

}

#### 3.2.24

typedef struct {

int data[max];

int top;

} ss;

void initss(ss\* ss) {

ss->top = -1;

}

int Empty(ss\* ss) {

return ss->top == -1;

}

void push(ss\* ss, int value) {

if (ss->top < max - 1) {

ss->data[++ss->top] = value;

} else {

printf("栈已满，无法入栈\n");

}

}

int pop(ss\* ss) {

if (!Empty(ss)) {

return ss->data[ss->top--];

} else {

printf("栈为空，无法出栈\n");

return -1;

}

}

void decimal(int num) {

ss ss;

initss(&ss);

while (num != 0) {

push(&ss, num % 2);

num /= 2;

}

printf("二进制:");

while (!Empty(&ss)) {

printf("%d", pop(&ss));

}

printf("\n");

}

void decimalt(int num) {

ss ss;

initss(&ss);

while (num != 0) {

push(&ss, num % 8);

num /= 8;

}

printf("八进制:");

while (!Empty(&ss)) {

printf("%d", pop(&ss));

}

printf("\n");

}

void decimalT(int num) {

ss ss;

initss(&ss);

while (num != 0) {

int remain = num % 16;

if (remain < 10) {

push(&ss, remain);

} else {

push(&ss, 'A' + remain - 10);

}

num /= 16;

}

printf("十六进制:");

while (!Empty(&ss)) {

int value = pop(&ss);

if (value >= 0 && value <= 9) {

printf("%d", value);

} else {

printf("%c", value);

}

}

printf("\n");

}

#### 3.2.26

void sentence(char\*p1)

{

char\*p2;

if(p1[0]>='a'&&p1[0]<='z')

p1[0]-=32;

while(\*p1)

{

p2=p1+1;

if(\*p1==' ')

{

while(\*p2==' ')

{p2++;}

strcpy(p1,p2-1);

}

p1++;

}

\*p1='.';

\*(p1+1)='\0';

}

#### 3.2.27

typedef struct {

int row;

int col;

int value;

} tt;

void sparse(tt mata[], tt matb[], int rows, int cols) {

int a = 0, b = 0, c = 0;

tt matc[max];

while (a < mata[0].value && b < matb[0].value) {

int indexa = mata[a].row \* cols + mata[a].col;

int indexb = matb[b].row \* cols + matb[b].col;

if (indexa < indexb) {

matc[c++] = mata[a++];

} else if (indexa > indexb) {

matc[c++] = matb[b++];

} else {

matc[c].row = mata[a].row;

matc[c].col = mata[a].col;

matc[c++].value = mata[a++].value + matb[b++].value;

}

}

for (; a < mata[0].value; a++) {

matc[c++] = mata[a];

}

for (; b < matb[0].value; b++) {

matc[c++] = matb[b];

}

matc[0].row = rows;

matc[0].col = cols;

matc[0].value = c - 1;

printf("相加结果：\n");

for (int i = 0; i <= matc[0].value; i++) {

printf("%d %d %d\n", matc[i].row, matc[i].col, matc[i].value);

}

}

void spa(tt mata[], tt matb[], int numRowsa, int numcolsa, int numcolsb) {

tt matc[max];

if (numcolsa != matb[0].row) {

printf("无法相乘\n");

return;

}

// 元素初始化为0

for (int i = 0; i < numRowsa; i++) {

for (int j = 0; j < numcolsb; j++) {

matc[i \* numcolsb + j].row = i;

matc[i \* numcolsb + j].col = j;

matc[i \* numcolsb + j].value = 0;

}

}

for (int i = 0; i < numRowsa; i++) {

int startPosa = mata[i].value;

int endPosa = (i == numRowsa - 1) ? mata[0].value : mata[i + 1].value;

for (int j = 0; j < numcolsb; j++) {

int sum = 0;

for (int k = startPosa; k < endPosa; k++) {

int colIdxa = mata[k].col;

for (int l = 0; l < matb[0].value; l++) {

int rowIdxb = matb[l].row;

int colIdxb = matb[l].col;

int valueb = matb[l].value;

if (colIdxa == rowIdxb && j == colIdxb) {

sum += mata[k].value \* valueb;

break;

}

}

}

if (sum != 0) {

matc[i \* numcolsb + j].row = i;

matc[i \* numcolsb + j].col = j;

matc[i \* numcolsb + j].value = sum;

}

}

}

matc[0].row = numRowsa;

matc[0].col = numcolsb;

matc[0].value = numRowsa \* numcolsb;

printf("相乘结果：\n");

for (int i = 0; i <= matc[0].value; i++) {

printf("%d %d %d\n", matc[i].row, matc[i].col, matc[i].value);

}

}

int main() {

// 矩阵a

tt mata[max];

mata[0].row = 3;

mata[0].col = 3;

mata[0].value = 4;

mata[1].row = 0;

mata[1].col = 0;

mata[1].value = 1;

mata[2].row = 1;

mata[2].col = 1;

mata[2].value = 2;

mata[3].row = 1;

mata[3].col = 2;

mata[3].value = 3;

mata[4].row = 2;

mata[4].col = 0;

mata[4].value = 4;

// 矩阵b

tt matb[max];

matb[0].row = 3;

matb[0].col = 3;

matb[0].value = 4;

matb[1].row = 0;

matb[1].col = 1;

matb[1].value = -1;

matb[2].row = 1;

matb[2].col = 0;

matb[2].value = 2;

matb[3].row = 1;

matb[3].col = 2;

matb[3].value = 3;

matb[4].row = 2;

matb[4].col = 1;

matb[4].value = 4;

sparse(mata, matb, 3, 3);

spa(mata, matb, 3, 3, 3);

return 0;

}

#### 3.2.28

#define ver int

void find(int num[],int n,int a,int b){

int index1=-1,index2=-1;

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

if(num[i]==a)

index1=i;

if(num[i]==b)

index2=i;

}

if(index1==-1 || index2==-1){

printf("They don`t have a same gradeparents.\n");

return ;

}

while(num[index1]!=num[index2]){

index1/=2;

index2/=2;

}

if(num[index1] == num[index2])

printf("这两个节点的最近公共祖先的编号是 %d\n",num[index1]);

}

#### 3.2.29

typedef struct{

char \*base;

char \*top;

int stacksize;//栈的存储容量

}OPRTstack;

typedef struct{

double \*base;

double \*top;

int stacksize;

}NUMstack;

int createStack(OPRTstack\*s)

{

s->base=(char\*)malloc(sizeof(char)\*defaultsize);

if(!s->base)return 0;

s->top=s->base;

s->stacksize=10;

return 1;

}

int pop(OPRTstack \*s,char \*e)

{

if(s->top==s->base)return 0;

s->top--;

\*e=\*(s->top);

return 1;

}

int push(OPRTstack\*s,char e)

{

if(s->top-s->base>=s->stacksize)

{

s->base=(char\*)realloc(s->base,sizeof(char)\*(s->stacksize+increasesize));

if(!s->base)return 0;

s->top=s->base+s->stacksize;

s->stacksize+=increasesize;

}

\*(s->top)=e;

s->top++;

}

int isEmpty(OPRTstack \*s)

{

if(s->top==s->base)return 1;

else return 0;

}

char GetTop(OPRTstack \*s)

{

if(!isEmpty(s))

{

char\*temp=s->top;

temp--;

return \*(temp);

}

else return '!';//这样定义的话，栈里面不能存储！这个数据

}

void showStack(OPRTstack\*s)

{

if(isEmpty(s))return ;

for(int i=0;i<s->top-s->base;i++)

{

printf("%c ",s->base[i]);

}

printf(" ");

}

int createStack(NUMstack\*s)

{

s->base=(double\*)malloc(sizeof(double)\*defaultsize);

if(!s->base)return 0;

s->top=s->base;

s->stacksize=10;

return 1;

}

int pop(NUMstack \*s,double \*e)

{

if(s->top==s->base)return 0;

s->top--;

\*e=\*(s->top);

return 1;

}

int push(NUMstack\*s,double e)

{

if(s->top-s->base>=s->stacksize)

{

s->base=(double\*)realloc(s->base,sizeof(double)\*(s->stacksize+increasesize));

if(!s->base)return 0;

s->top=s->base+s->stacksize;

s->stacksize+=increasesize;

}

\*(s->top)=e;

s->top++;

}

int isEmpty(NUMstack \*s)

{

if(s->top==s->base)return 1;

else return 0;

}

double GetTop(NUMstack \*s)

{

if(!isEmpty(s))

{

double \*temp=s->top;

temp--;

return \*(temp);

}

else return -1;//这样定义的话，栈里面不能存储！这个数据

}

void showStack(NUMstack\*s)

{

if(isEmpty(s))return ;

for(int i=0;i<s->top-s->base;i++)

{

printf("%f ",s->base[i]);

}

printf(" ");

}

int isOPRT(char c)//判断c是不是运算符

{

if(c=='+'||c=='-'||c=='\*'||c=='/'||c=='('||c==')'||c=='#')return 1;

else return 0;

}

char compare(char a,char b)

{

if(a=='+')

{

if(b=='\*'||b=='/'||b=='(') return '<';

else return '>';

}

else if(a=='-')

{

if(b=='\*'||b=='/'||b=='(') return '<';

else return '>';

}

else if(a=='\*')

{

if(b=='(')return '<';

else return '>';

}

else if(a=='/')

{

if(b=='(')return '<';

else return '>';

}

else if(a=='(')

{

if(b==')')return '=';

else if(b=='#') return '!';

else return '<';

}

else if(a==')')

{

if(b=='(')return '!';

else return '>';

}

else if(a=='#')

{

if(b==')')return '!';

if(b=='#')return '=';

else return '<';

}

}

double calculate(double left,double right, char operators)

{

switch(operators)

{

case '+':

return left+right;

case '-':

return 1.0\*left-right;

case '\*':

return left\*right;

case '/':

return 1.0\*left/right;

}

}

#### 3.2.31

typedef int KeyType;

typedef struct node

{

KeyType key ; /\*关键字的值\*/

struct node \*lchild,\*rchild;/\*左右指针\*/

}BSTNode, \*BSTree;

void InsertBST(BSTree \*bst, KeyType key)

/\*若在二叉排序树中不存在关键字等于key的元素，插入该元素\*/

{

BSTree s;

if (\*bst == NULL)/\*递归结束条件\*/

{

s=(BSTree)malloc(sizeof(BSTNode));/\*申请新的结点s\*/

s-> key=key;

s->lchild=NULL;

s->rchild=NULL;

\*bst=s;

}

else

if (key < (\*bst)->key)

InsertBST(&((\*bst)->lchild), key);/\*将s插入左子树\*/

else

if (key > (\*bst)->key)

InsertBST(&((\*bst)->rchild), key); /\*将s插入右子树\*/

}

void CreateBST(BSTree \*bst)

/\*从键盘输入元素的值，创建相应的二叉排序树\*/

{

KeyType key;

\*bst=NULL;

scanf("%d", &key);

while (key!=ENDKEY) /\*ENDKEY为自定义常量\*/

{

InsertBST(bst, key);

scanf("%d", &key);

}

}

void PreOrder(BSTree root)

/\*先序遍历二叉树, root为指向二叉树根结点的指针\*/

{

if (root!=NULL)

{

printf("%d ",root->key); /\*输出结点\*/

PreOrder(root->lchild); /\*先序遍历左子树\*/

PreOrder(root->rchild); /\*先序遍历右子树\*/

}

}

/\*

BSTree SearchBST(BSTree bst, KeyType key)

/ \*在根指针bst所指二叉排序树中，递归查找某关键字等于key的元素，若查找成功，返回指向该元素结点指针，否则返回空指针\* /

{

if (!bst)

return NULL;

else

if (bst->key == key)

return bst;/ \*查找成功\* /

else

if (bst->key > key)

return SearchBST(bst->lchild, key);/ \*在左子树继续查找\* /

else

return SearchBST(bst->rchild, key);/ \*在右子树继续查找\* /

}\*/

BSTree SearchBST(BSTree bst, KeyType key)

/\*在根指针bst所指二叉排序树bst上，查找关键字等于key的结点，若查找成功，返回指向该元素结点指针，否则返回空指针\*/

{

BSTree q;

q=bst;

while(q)

{

if (q->key == key)

return q; /\*查找成功\*/

if (q->key > key)

q=q->lchild; /\*在左子树中查找\*/

else

q=q->rchild; /\*在右子树中查找\*/

}

return NULL; /\*查找失败\*/

}/\*SearchBST\*/

BSTNode \* DelBST(BSTree t, KeyType k) /\*在二叉排序树t中删去关键字为k的结点\*/

{

BSTNode \*p, \*f,\*s ,\*q;

p=t;

f=NULL;

while(p) /\*查找关键字为k的待删结点p\*/

{

if(p->key==k ) break; /\*找到则跳出循环\*/

f=p; /\*f指向p结点的双亲结点\*/

if(p->key>k)

p=p->lchild;

else

p=p->rchild;

}

if(p==NULL) return t; /\*若找不到，返回原来的二叉排序树\*/

if(p->lchild==NULL) /\*p无左子树\*/

{

if(f==NULL)

t=p->rchild; /\*p是原二叉排序树的根\*/

else

if(f->lchild==p) /\*p是f的左孩子\*/

f->lchild=p->rchild ; /\*将p的右子树链到f的左链上\*/

else /\*p是f的右孩子\*/

f->rchild=p->rchild ; /\*将p的右子树链到f的右链上\*/

free(p); /\*释放被删除的结点p\*/

}

else /\*p有左子树\*/

{

q=p;

s=p->lchild;

while(s->rchild) /\*在p的左子树中查找最右下结点\*/

{

q=s;

s=s->rchild;

}

if(q==p)

q->lchild=s->lchild ; /\*将s的左子树链到q上\*/

else

q->rchild=s->lchild;

p->key=s->key; /\*将s的值赋给p\*/

free(s);

}

return t;

} /\*DelBST\*/

#### 3.2.32

struct TreeNode {

char val;

struct TreeNode\* left;

struct TreeNode\* right;

};

// 初始化二叉树节点

struct TreeNode\* createNode(char val) {

struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

node->val = val;

node->left = NULL;

node->right = NULL;

return node;

}

// 查找从根节点到目标叶子节点的路径

int findPath(struct TreeNode\* root, char target, char\* path, int index) {

if (root == NULL) {

return 0;

}

path[index] = root->val;

index++;

if (root->val == target && root->left == NULL && root->right == NULL) {

return 1;

}

if (findPath(root->left, target, path, index) || findPath(root->right, target, path, index)) {

return 1;

}

return 0;

}

#### 3.2.34

int main()

{

float x;

int ch;

do

{

scanf("%f",&x);

x=x\*100;

if(x==3.00)

printf("A ");

if(x==6.00)

printf("B ");

if(x==9.00)

printf("C ");

if(x==15.00)

printf("D ");

if(x==17.00)

printf("E ");

}while((ch=getchar())!='\n');

return 0;

}

#### 3.2.39

int a[1000],b[1000],c[1000];

//返回两者较大值

int max(int a,int b){

if(a>b)return a;

else return b;

}

//将数组设置为0

int setZero(int\*num,int len){

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

num[i]=0;

}

//比较两个数组形式数字大小

int compare(int\*num1,int\*num2){

int i,j;

for(i=999;num1[i]==0;i--);

for(j=999;num2[j]==0;j--);

if(i<j)return -1;

else if(i>j)return 1;

else{

for(;i>=0;i--){

if(num1[i]>num2[i])

return 1;

else if(num1[i]<num2[i])

return -1;

}

return 0;

}

}

//将一个数组赋给另一个数组

int copy(int\*num1,int\*num2){

int i,j;

for(i=0;i<1000;i++)

num1[i]=0;

for(i=999;num2[i]==0;i--);

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

num1[j]=num2[j];

return i+1;

}

//大数加法

void add(char\*str1,char\*str2){

setZero(a,1000);

setZero(b,1000);

setZero(c,1000);

printf("%s + %s = ",str1,str2);

int i,carry=0;

int len1=strlen(str1),len2=strlen(str2),len=max(len1,len2);

for(i=0;i<len1;i++)

a[i]=str1[len1-1-i]-'0';

for(i=0;i<len2;i++)

b[i]=str2[len2-1-i]-'0';

for(i=0;i<len;i++){

c[i]=(a[i]+b[i]+carry)%10;

carry=(a[i]+b[i]+carry)/10;

}

if(carry!=0)

c[len++]=1;

for(i=len-1;i>=0;i--)

printf("%d",c[i]);

printf("\n");

}

//大数减法

void sub(char\*str1,char\*str2){

setZero(a,1000);

setZero(b,1000);

setZero(c,1000);

printf("%s - %s = ",str1,str2);

int i,borrow=0;

int len1=strlen(str1),len2=strlen(str2),len=max(len1,len2);

for(i=0;i<len1;i++)

a[i]=str1[len1-1-i]-'0';

for(i=0;i<len2;i++)

b[i]=str2[len2-1-i]-'0';

//初步根据位数判断两数大小

if(len2>len1){

printf("-");

for(i=0;i<len;i++){

c[i]=b[i]-a[i]-borrow;

if(c[i]<0){

borrow=1;

c[i]+=10;

}

else

borrow=0;

}

}

else{

for(i=0;i<len;i++){

c[i]=a[i]-b[i]-borrow;

if(c[i]<0&&i+1<len){

borrow=1;

c[i]+=10;

}

else

borrow=0;

}

}

//如果发现被减数小于减数，则重新计算，用减数减去被减数，输出负号

if(c[len-1]<0){

printf("-");

for(i=0;i<len;i++){

c[i]=b[i]-a[i]-borrow;

if(c[i]<0&&i+1<len){

borrow=1;

c[i]+=10;

}

else

borrow=0;

}

}

while(c[len-1]==0&&len>1)

len--;

for(i=len-1;i>=0;i--)

printf("%d",c[i]);

printf("\n");

}

//大数乘法

void mul(char\*str1,char\*str2) {

setZero(a,1000);

setZero(b,1000);

setZero(c,1000);

printf("%s \* %s = ",str1,str2);

int i,j,k,carry=0,tmp;

int len1=strlen(str1),len2=strlen(str2);

for(i=0;i<len1;i++)

a[i]=str1[len1-1-i]-'0';

for(i=0;i<len2;i++)

b[i]=str2[len2-1-i]-'0';

for(i=0;i<len1;i++){

for(k=i,j=0;j<len2;j++,k++){

c[k]+=a[i]\*b[j];

}

}

for(i=0;i<=k;i++) {

c[i]+=carry;

tmp=c[i];

c[i]=tmp%10;

carry=tmp/10;

}

for(i=k;c[i]==0;i--);

for(;i>=0;i--)

printf("%d",c[i]);

printf("\n");

}

//大数除法

void div(char\*str1,char\*str2){

setZero(a,1000);

setZero(b,1000);

setZero(c,1000);

printf("%s / %s = ",str1,str2);

int i,j,tmp,tmp2=0,borrow=0,temp[1000]={0},temp2[1000]={0};

int len1=strlen(str1),len2=strlen(str2);

for(i=0;i<len1;i++)

a[i]=str1[len1-1-i]-'0';

for(i=0;i<len2;i++)

b[i]=str2[len2-1-i]-'0';

if(compare(a,b)<0){

printf("0\n");

return;

}

while(compare(a,b)>=0){

tmp=len1-len2;

if(tmp==tmp2&&tmp>0)

tmp--;

tmp2=tmp;

setZero(temp2,1000);

for(i=len1-1;i>=tmp;i--)

temp2[i]=b[i-tmp];

copy(temp,a);

if(compare(temp,temp2)<0)

continue;

for(j=1;;j++){

borrow=0;

for(i=tmp;i<len1;i++){

temp[i]=a[i]-temp2[i]-borrow;

if(temp[i]<0){

borrow=1;

temp[i]+=10;

}

else

borrow=0;

}

len1=copy(a,temp);

c[tmp]=j;

if(compare(temp,temp2)<0)

break;

}

}

for(i=999;c[i]==0;i--);

for(;i>=0;i--)

printf("%d",c[i]);

printf("\n");

}

#### 3.2.41

int i;

void reverse(int num[],int n,int a,int b)

{

int index=-1,index1=-1;

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

{

if(num[i]==a)

index=i;

if(num[i]==b)

index1=i;

}

if(index==-1||index1==-1)

{

printf("error\n");

return ;

}

while(num[index]!=num[index1]){

index/=2;

index1/=2;

}

if(num[index] == num[index1])

printf("它们的公共节点是 %d\n",num[index1]);

}

#### 3.2.42

int i;

void reverse(int num[],int n,int a,int b)

{

int c=0;

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

{

if(num[i]>a&&num[i]<b)

c=c+num[i];

}

printf("%d\n",c);

}

#### 3.2.15

int main()

{

int s[100]={0};

int i=0;

char ch;

while((ch=getchar())!='\n')

{

s[i]=ch;

i++;

}

int t=i;

int x=0;

for(i=0;i<t-1;i++)

{

if(s[i]>s[i+1])

x++;

else

x--;

}

if(x==t-1||x+t==1)

printf("不是摆动序列，需要删除 %d 个元素",t-2);

else

printf("是摆动序列");

return 0;

}

#### 3.2.25

int main()

{

char s[200]={0};

char ss[100]={0};

char ch;

int i=0;

while((ch=getchar())!='\n')

{

s[i]=ch;

i++;

}

printf("%d\n",i);

i=0;

int len1=strlen(s);

while((ch=getchar())!='\n')

{

ss[i]=ch;

i++;

}

int len2=strlen(ss);

int j;

int t,x;

int flag;

for(i=0;i<len1;i++)

{

flag=0;

j=0;

if(s[i]==ss[0])

{

t=i;

x=t;

flag=1;

for(j=0;j<len2;j++)

{

if(flag==len2)

printf("%d",t+1);

if(s[x+1]==ss[j+1])

{

x++;

flag++;

}

else

break;

}

}

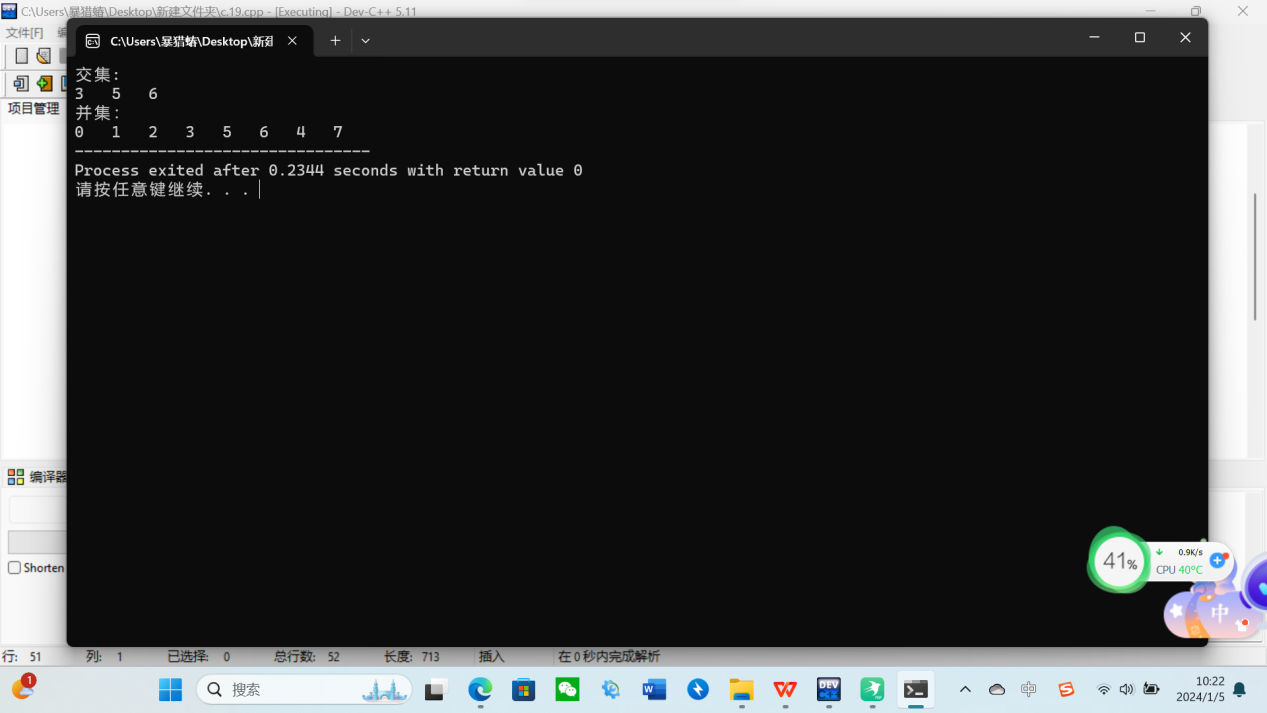
}

return 0;

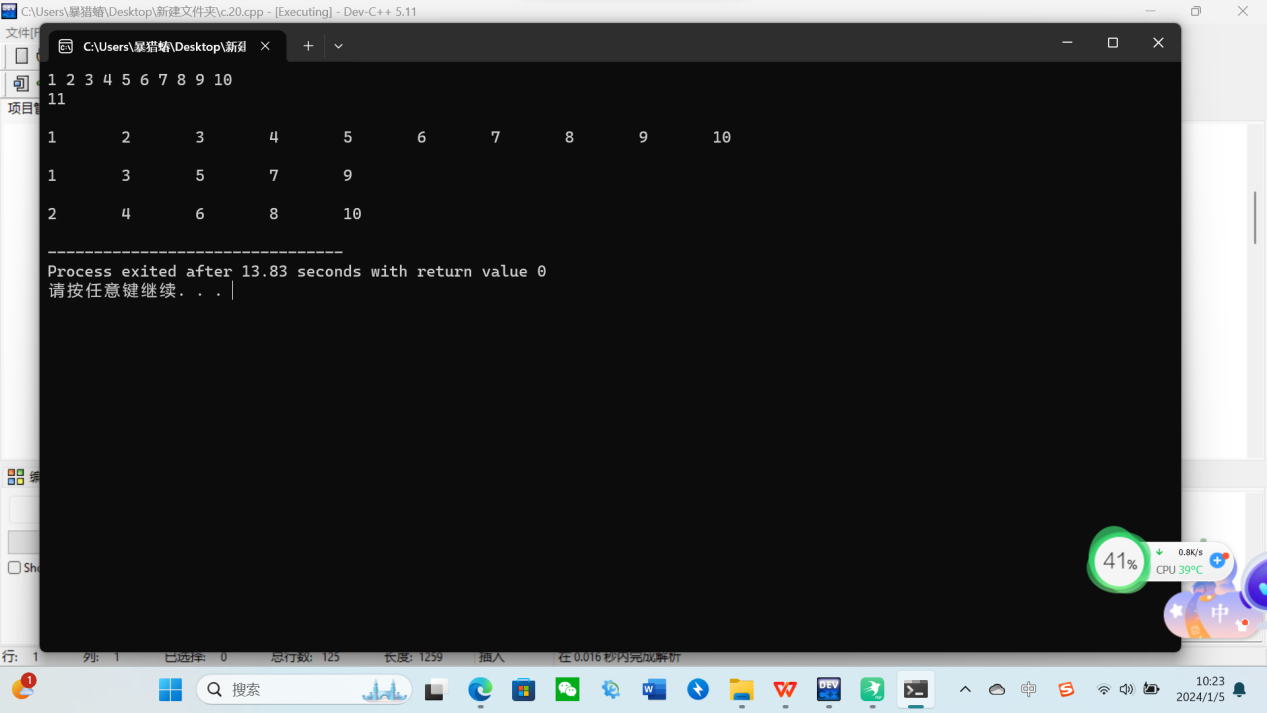
}

### 3.1.3 程序的运行截图

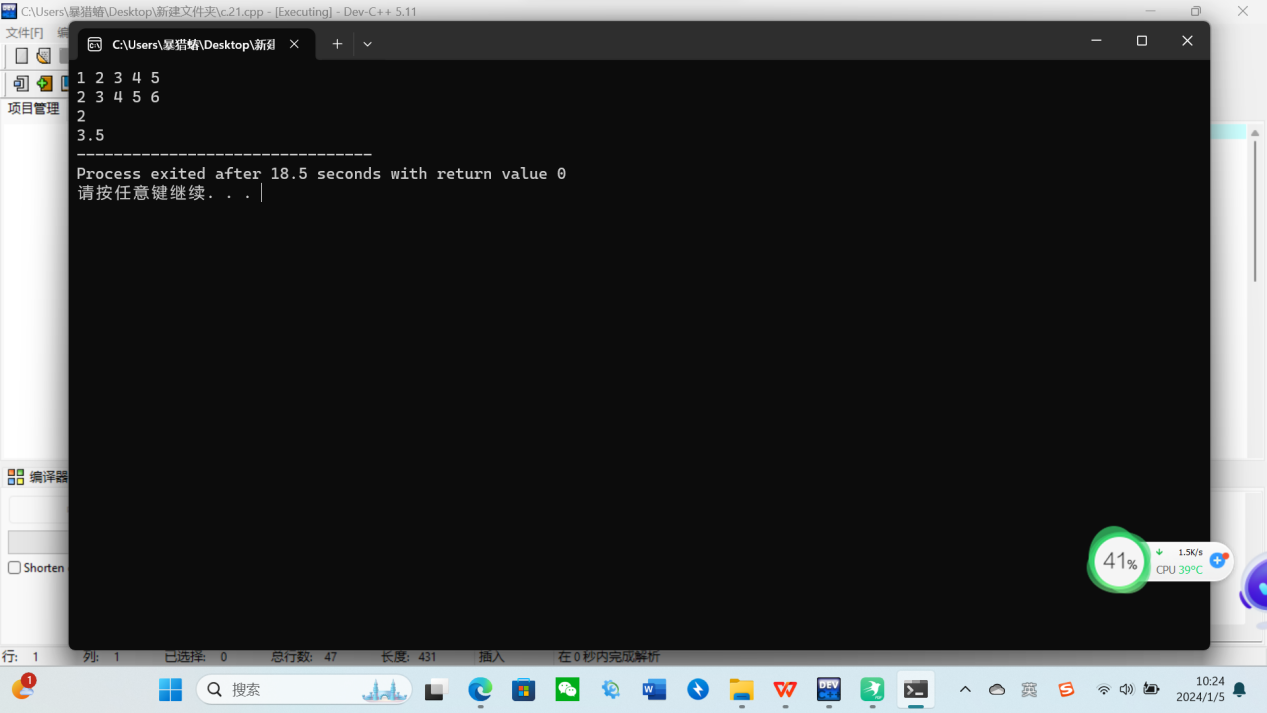
#### 3.2.19



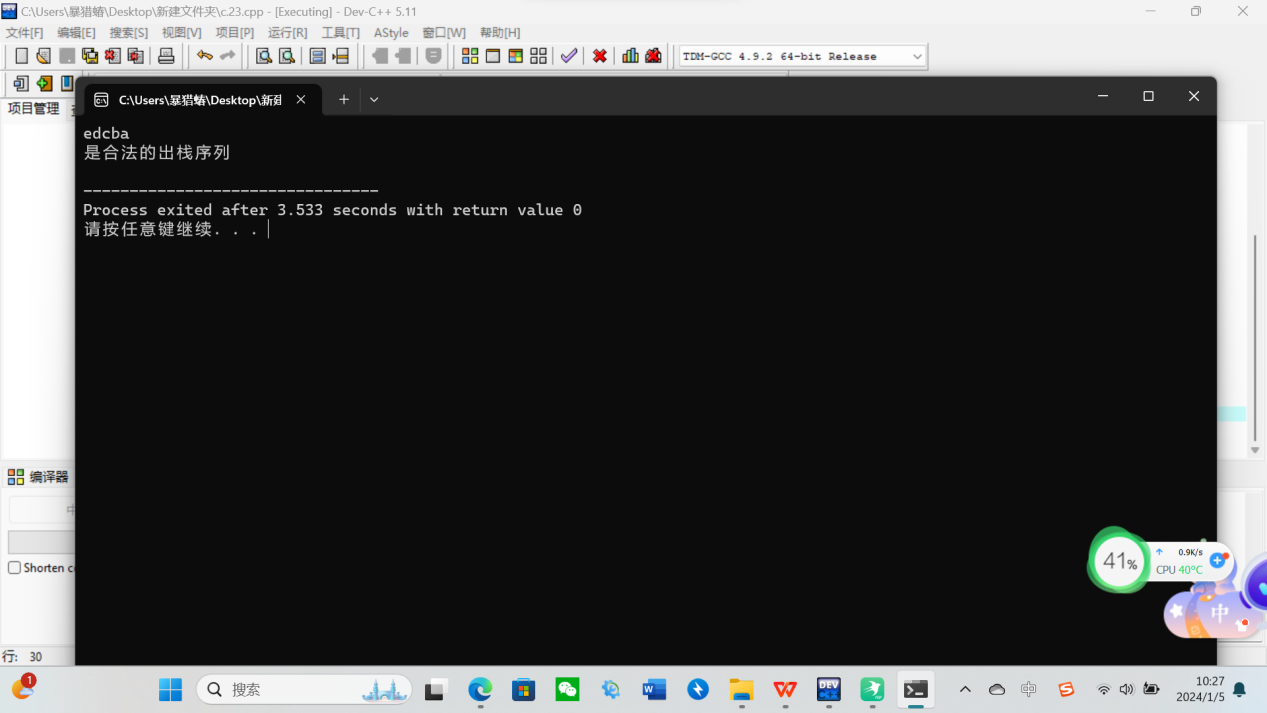
#### 3.2.20



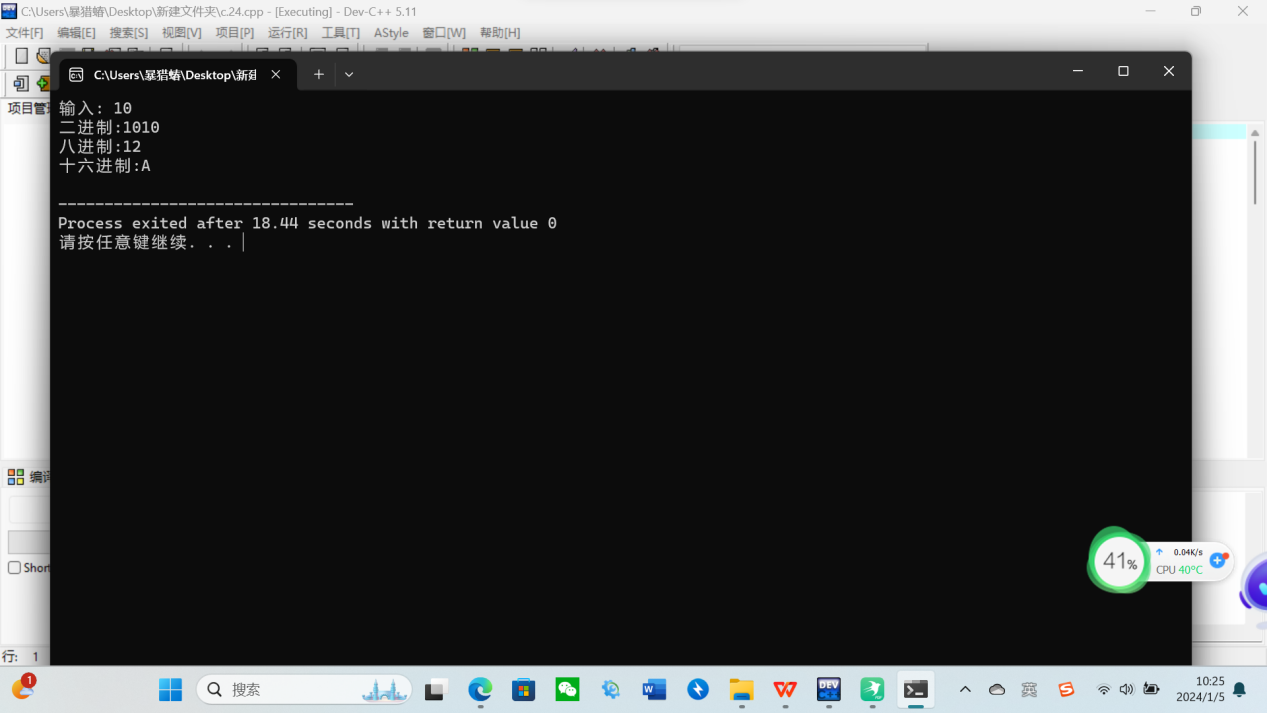
#### 3.2.21



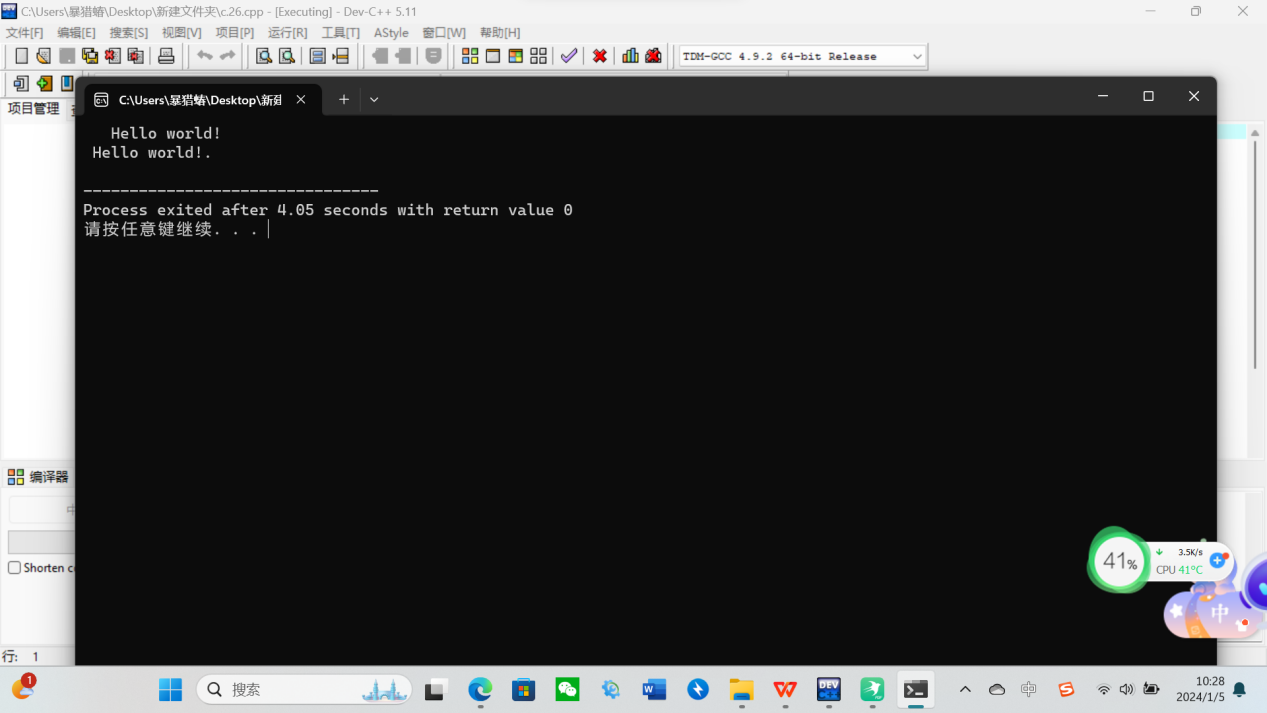
#### 3.2.23



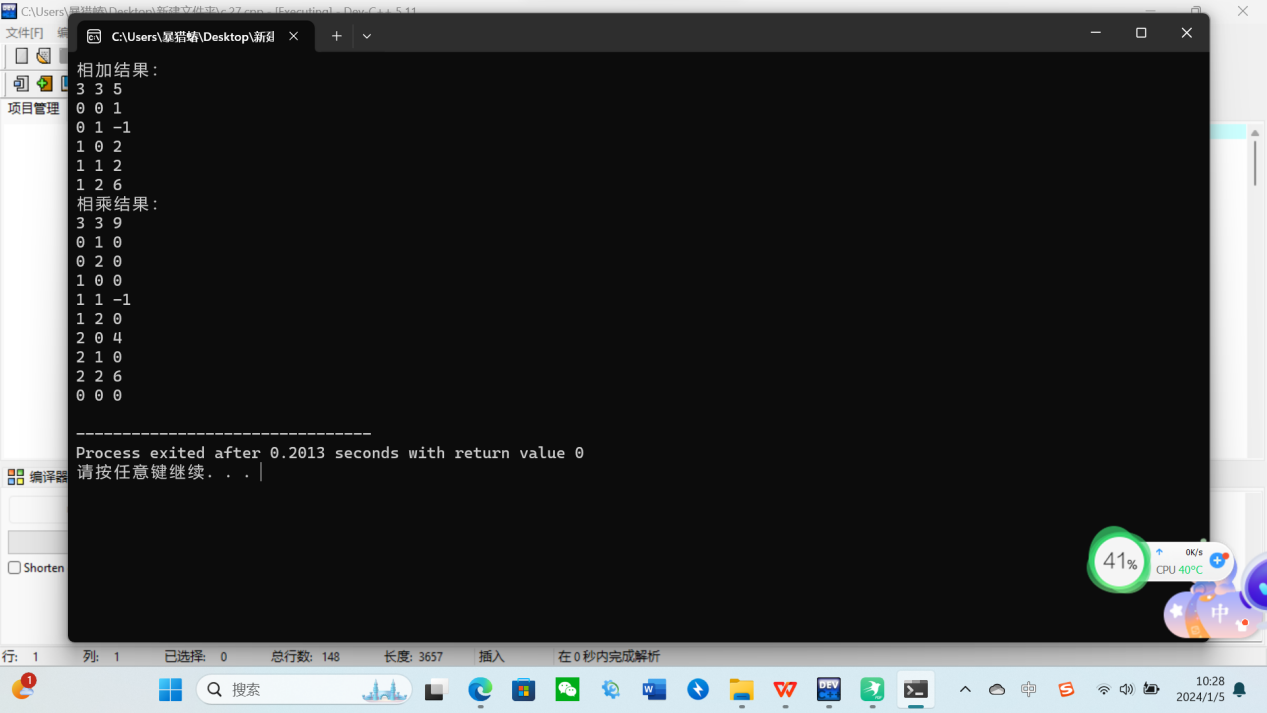
#### 3.2.24



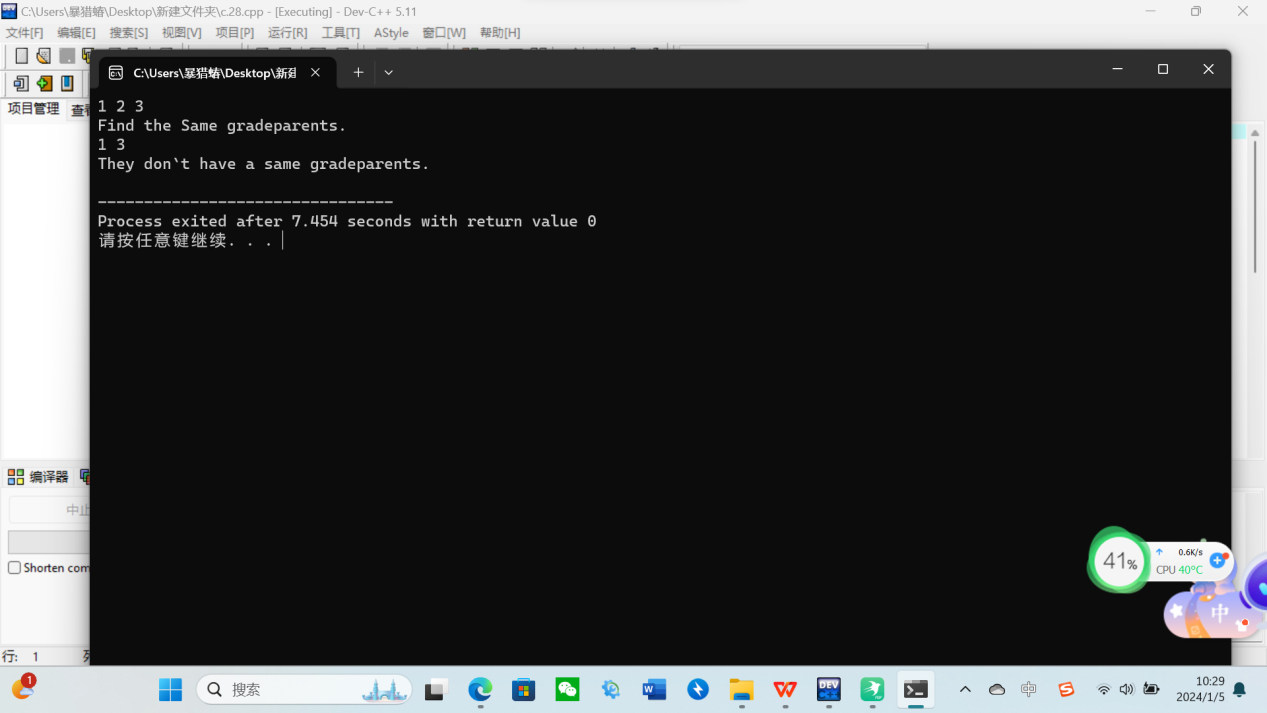
#### 3.2.26



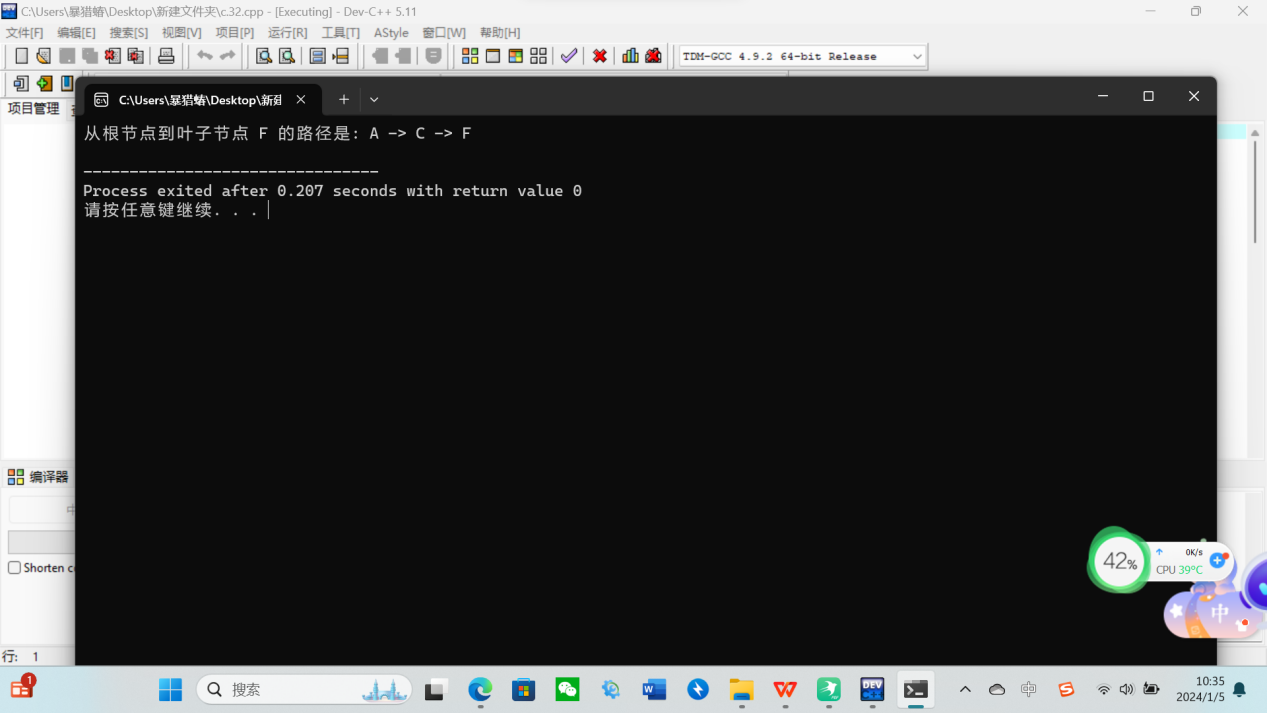
#### 3.2.27



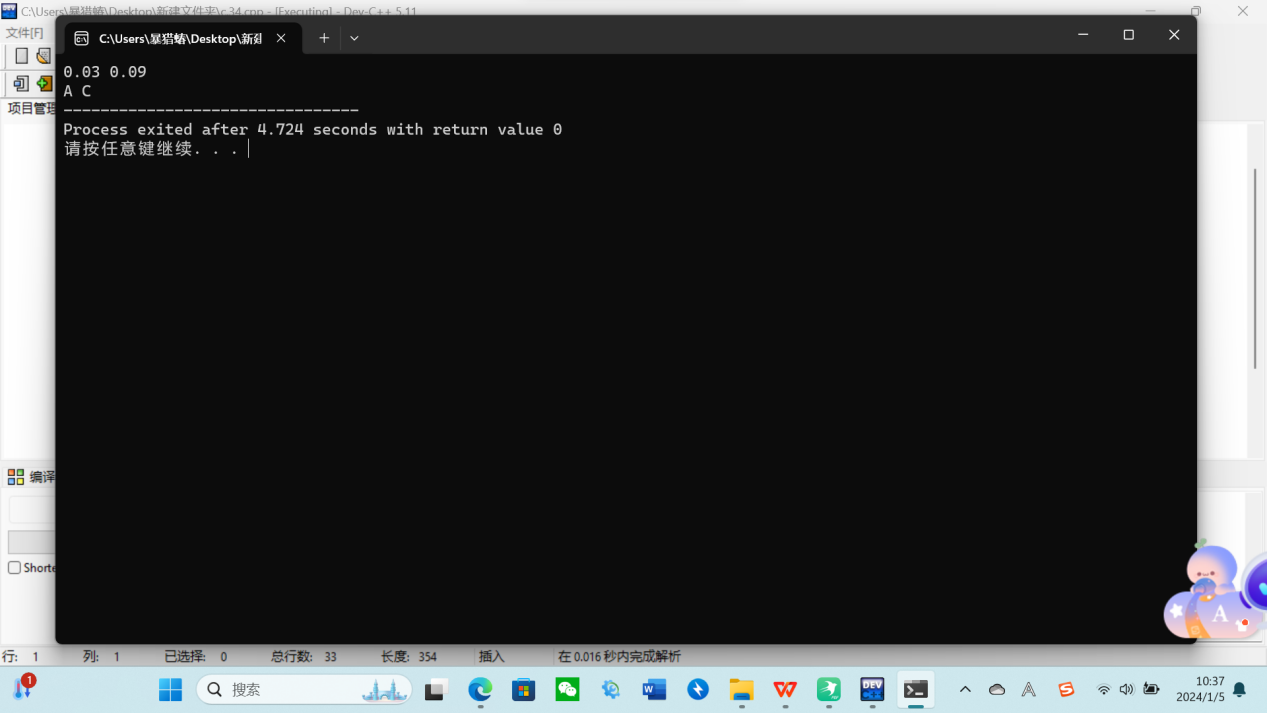
#### 3.2.28



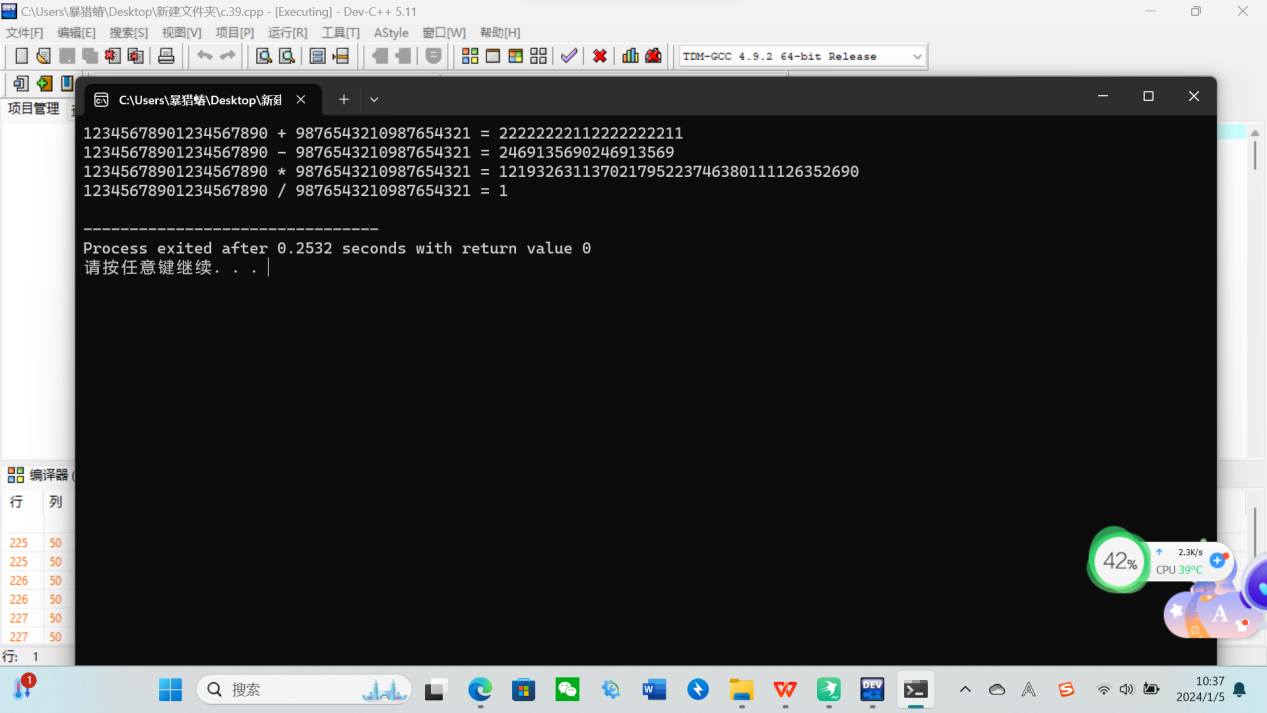
#### 3.2.32



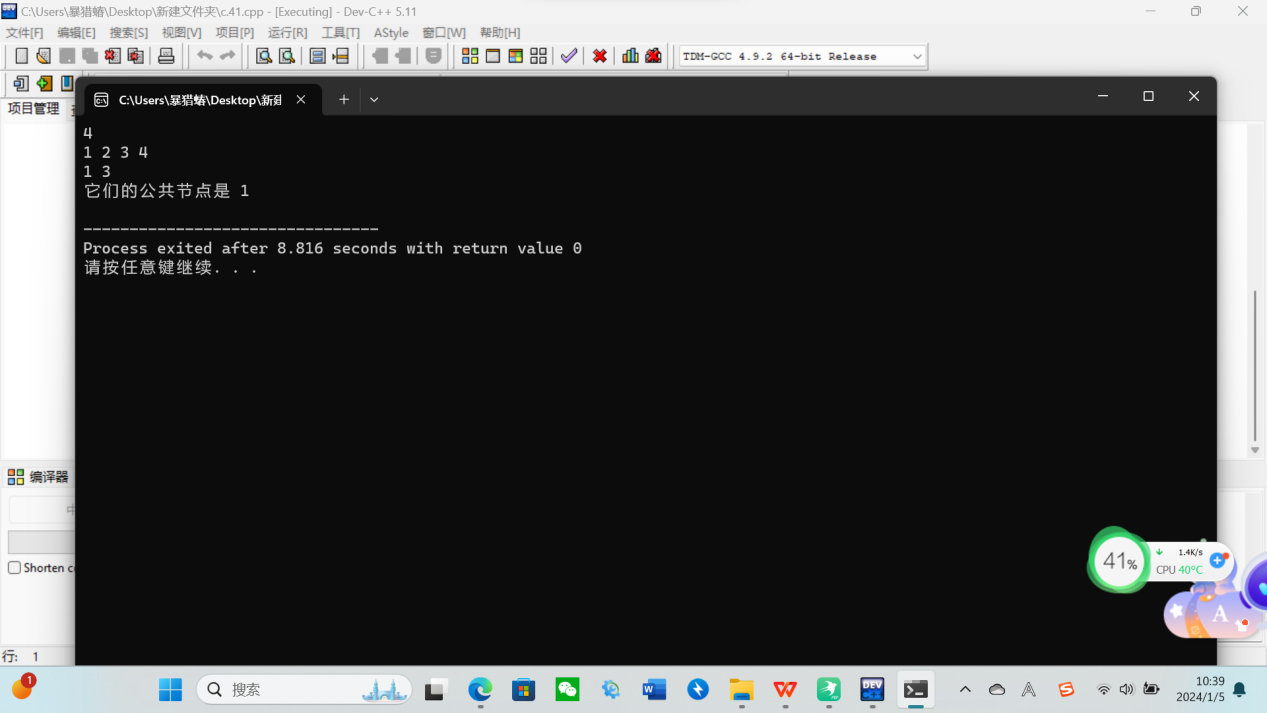
#### 3.2.34



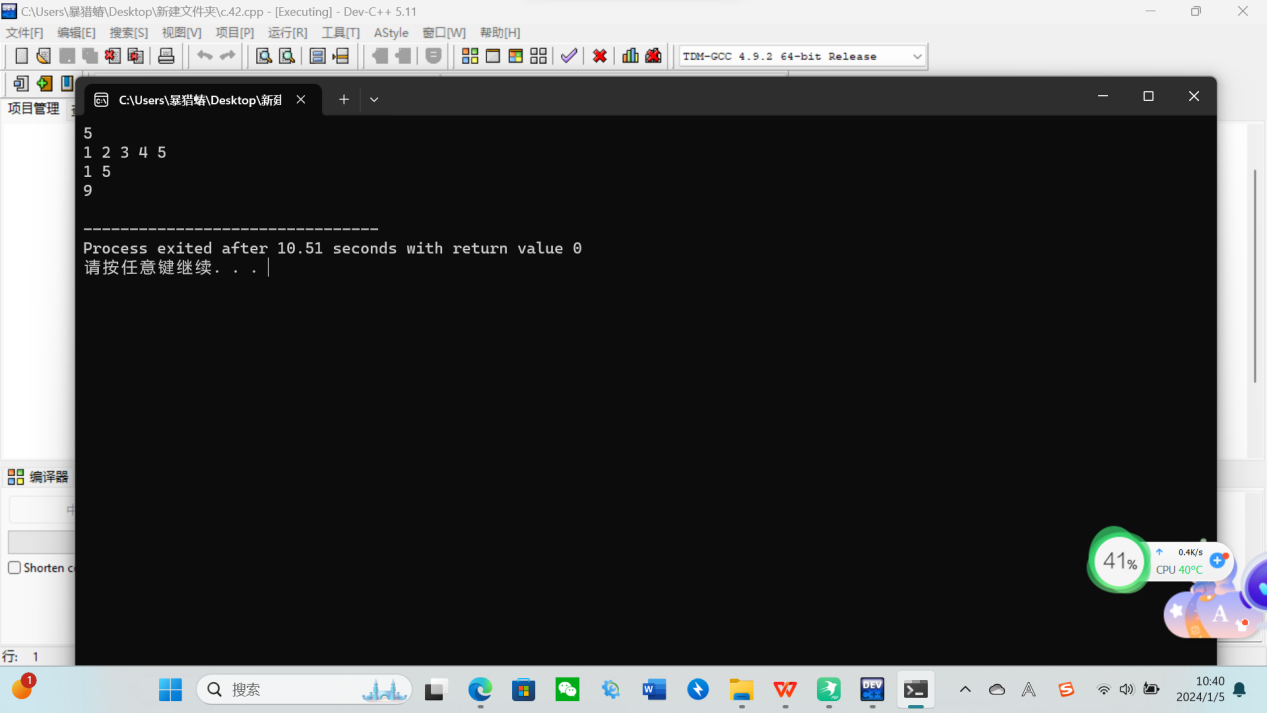
#### 3.2.39



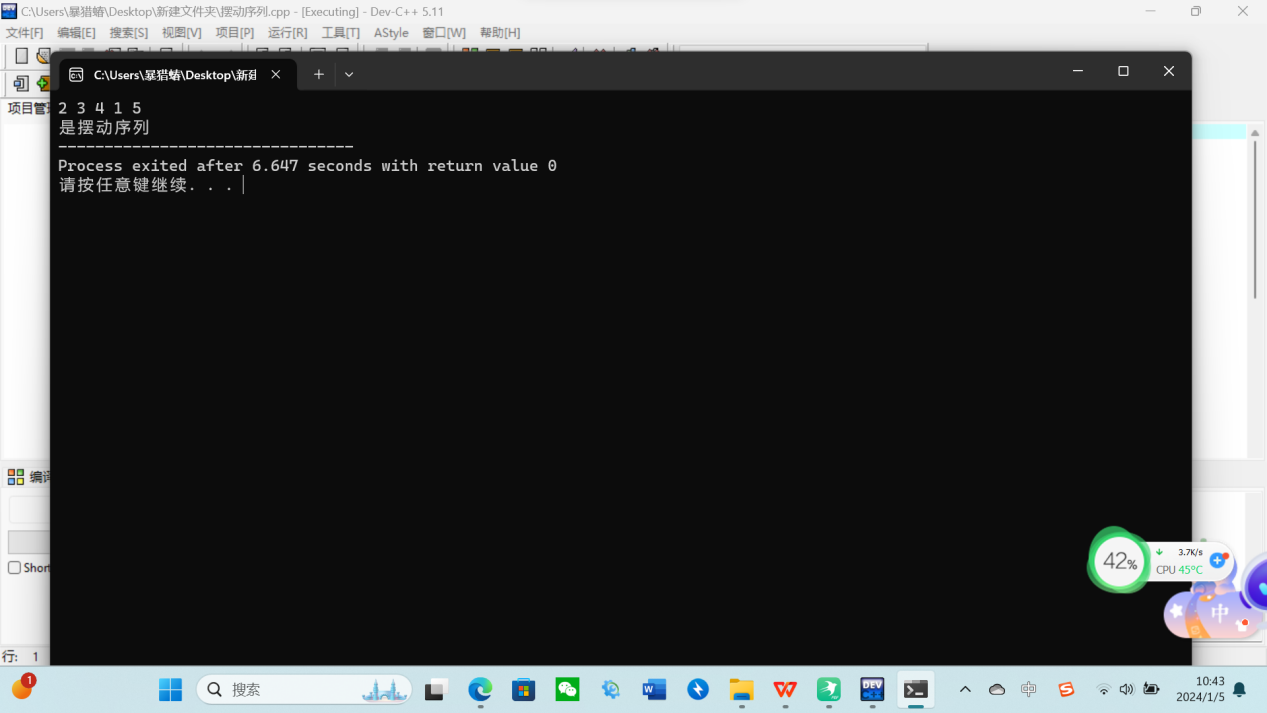
#### 3.2.41



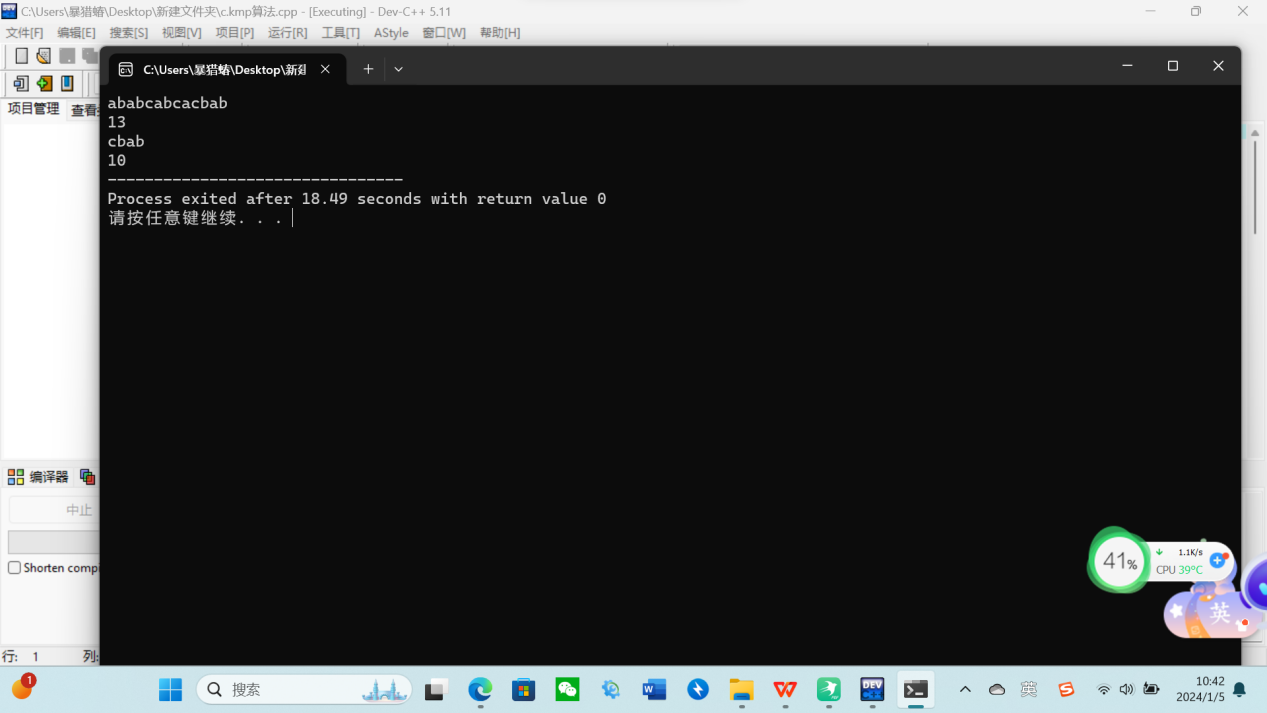
#### 3.2.42



#### 3.2.15



#### 3.2.25



本人承诺，本报告系本人独立完成，没有成篇和成段的内容拷贝其他同学的成果。如有不实，除本报告按零分计外，愿意接受纪律处分。

（签字）

|  |
| --- |
| 最终成绩 |
|  |