数据结构

实验报告

学　　号：	**20188068**
姓　　名：	**孔天欣**
提交日期：	**2019-12-01**
成　　绩：	

东北大学秦皇岛分校

计算机与通信工程学院

【实验内容】

1. 线性表-约瑟夫环

```
1.  LinkList ListBuild()
2.  {
3.      LinkList head = (LinkList)malloc(sizeof(LNode));
4.      LinkList p = head;
5.      cout << "有多少人（n值）?" << endl;
6.      int n;
7.      cin >> n;
8.      cout << "输入每个人的密码：" << endl;
9.      for (int i = 1; i <= n; i++)
10.     {
11.         p->next = (LinkList)malloc(sizeof(LNode));
12.         p = p->next;
13.         cin >> p->data;
14.         p->num = i;
15.     }
16.     p->next = head->next;
17.     p = p->next;
18.     free(head);
19.     return p;
20. }
21.
22. void Joseph(LinkList p)
23. {
24.     cout << "请输入m值 : " << endl;
25.     int num;
26.     cin >> num;
27.     LinkList q = p;
28.     while (p)
29.     {
30.         for (int i = 1; i < num; i++)
31.         {
32.             q = p;
33.             p = p->next;
34.         }
35.         cout << p->num << " ";
36.         if (p->next == p)     //就剩一个人没出列
37.         {
38.             free(p);
39.             break;
40.         }
41.         q->next = p->next;
42.         num = p->data;
43.         free(p);
```

```
        p = q->next;
    }
}
int main()

{
    LinkList p = ListBuild();
    Joseph(p);
}
```

运行结果：

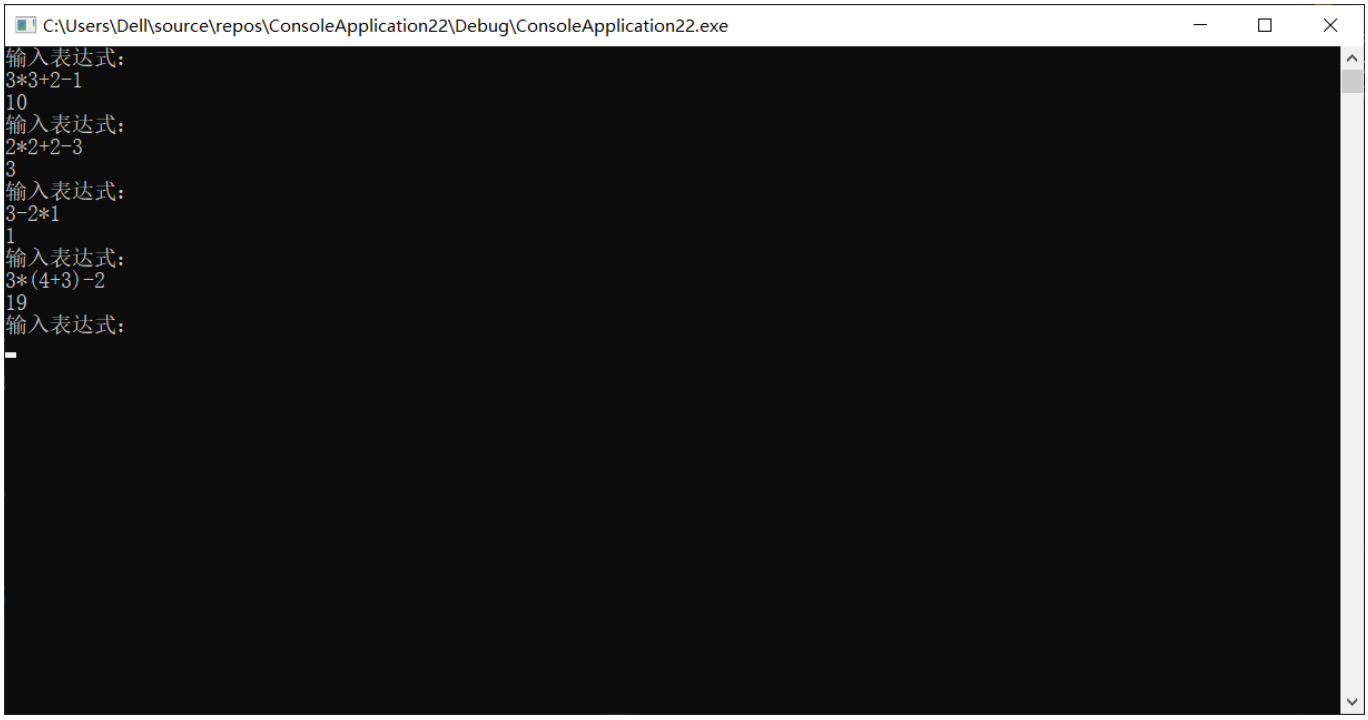
2. 栈-表达式求值

```
#include <Stack.h>

int char_to_int(char s)   //转 int
{
    return int(s) - 48;
}

char PriorityComparison(Stack *S, char c)  //运算符优先级比较
{
    char *p = S->top - 1;
    char prior[7][7] =
    { {'>','>','<','<','<','>','>'},
    {'>','>','<','<','<','>','>'},
     {'>','>','>','>','<','>','>'},
     {'>','>','>','>','<','>','>'},
     {'<','<','<','<','<','=',' '},
     {'>','>','>','>',' ','>','>'},
     {'<','<','<','<','<',' ','='} };

    char Operator[7] = { '+','-','*','/','(',')','#' };
    int m = 0; int n = 0;
    while (Operator[n] != c) { n++; }
    while (Operator[m] != *p) { m++; };
    return prior[m][n];
}

int Operate(int num1,char s,int num2)  //计算
{
    switch (s)
    {
    case '+':return num1 + num2; break;
    case'-':return num1 - num2; break;
    case'*':return num1 * num2; break;
    case'/':return num1 / num2; break;
    }
}


int main()
{
    Stack OPND, OPTR;
    string s;
    while (1)
    {
        cout << "输入表达式: " << endl;
```

```
46.         cin >> s;
47.         InitStack(&OPND); InitStack(&OPTR);
48.         StackPush(&OPTR, '#', 0);
49.         for (int i = 0; i < s.length(); i++)
50.         {
51.             if (s[i] >= '0'&&s[i] <= '9')
52.             {
53.                 StackPush(&OPND, s[i], 0);
54.             }
55.             else if (PriorityComparison(&OPTR, s[i]) == '<')  //优先度低的存入
56.             {
57.                 StackPush(&OPTR, s[i], 0);
58.             }
59.             else if (PriorityComparison(&OPTR, s[i]) == '=')
60.             {
61.                 StackPop(&OPTR);
62.             }
63.             else if (PriorityComparison(&OPTR, s[i]) == '>') //优先度高的立刻运算（这里指去括号的运
    算）
64.             {
65.                 if (OPND.top - 2)
66.                 {
67.                     char num1 = *(OPND.top - 2);
68.                     char num2 = *(OPND.top - 1);
69.                     StackPush(&OPND, '0' + char(Operate(char_to_int(num1), *(OPTR.top - 1), cha
    r_to_int(num2))), 1);
70.                     StackPop(&OPTR);
71.                     if (s[i] == ')')  //括号多删一次，因为多了个符号。(+)
72.                     {
73.                         StackPop(&OPTR);
74.                     }
75.                     else
76.                     {
77.                         StackPush(&OPTR, s[i], 0);
78.                     }
79.                 }
80.
81.             }
82.         }
83.         //完毕后计算剩下的值
84.         while (*(OPTR.top) != '#' &&*(OPTR.top - 1) != '#')
85.         {
86.             if (OPND.top - 2)
87.             {
88.                 char num1 = *(OPND.top - 2);
89.                 char num2 = *(OPND.top - 1);
```

```
90.                 StackPush(&OPND, '0' + (Operate(char_to_int(num1), *(OPTR.top - 1), char_to_int
    (num2))), 1);
91.                 // int to char
92.                 StackPop(&OPTR);
93.             }
94.         }
95.         //char to int
96.         cout << int(*(OPND.top - 1)) - 48 << endl;
97.         StackDestory(&OPND,&OPTR);
98.
99.     }
100.
101. }
```

运行结果：



3. 队列-猴子分桃

```
1.  void QueneBuild(Quene *Q)
2.  {
3.      cout << "有多少只猴子？" << endl;
4.      int n;
5.      cin >> n;
6.      Q->num = (int*)malloc(100 * sizeof(int));
7.      Q->Pch = (int*)malloc(100 * sizeof(int));
8.      Q->isfull = (bool*)malloc(100 * sizeof(bool));
9.      Q->front = Q->rear = 0;
10.     for (int i = 0; i < n; i++)
11.     {
12.         Q->num[i] = i + 1;
13.         Q->Pch[i] = 0;
14.         Q->isfull[i] = false;
15.         Q->rear++;
16.     }
17. }
18.
19. void DividePeach(Quene *Q)
20. {
21.     cout << "每只猴子可以分到多少桃？" << endl;
22.     int m;
23.     cin >> m;
24.     cout << "筐里最大的桃子数是？ " << endl;
25.     int k;
26.     cin >> k;
27.     int elemnum = Q->rear;
28.     int b = 0;   //从0开始
29.     int isfullnum = 0;
30.     int BOXnum = 0;
31.     while (isfullnum != Q->rear)
32.     {
33.         for (int n = 1; n <= k; n++)
34.         {
35.             BOXnum += n;
36.             if (b == Q->rear)
37.             {
38.                 b = 0;
39.             }
40.             if (!(Q->Pch[b] + BOXnum >= m) && !(Q->isfull[b]))
41.             {
42.                 Q->Pch[b] += n;
43.                 BOXnum = 0;
44.             }
45.             else if (!(Q->isfull[b]))
```

```
            {
                BOXnum -= m - Q->Pch[b];
                Q->Pch[b] = m;
                cout << Q->num[b] << " ";
                Q->isfull[b] = true;
                isfullnum++;
            }
            b++;
        }
    }
}
void QueneShow(Quene *Q)
{
    for (int i = 0; i < Q->rear; i++)
    {
        cout << "Q.num: " << Q->num[i] << " Q.Pch: " << Q->Pch[i] << " ";
        cout << endl;
    }
}
int main()
{
    Quene Q;
    QueneBuild(&Q);
    DividePeach(&Q);
}
```

运行结果：

4. 字符串和数组-三元组表稀疏矩阵

```
typedef struct {
    int row, col;   //该数据所在 row 行 col 列
    int elem;      //存储的数据
}Triple;

typedef struct {
    Triple data[100];   //非零元的数据构成的数组
    int rows, cols, num;  //该压缩前矩阵的行数，列数，非零元个数
}TSMatrix, *TSM;

void TSMatrixInit(TSMatrix *M, int a[100][100])
{
    cout << "输入矩阵的行和列 : " << endl;
    int m, n, i, j;
    cin >> m;
    cin >> n;
    cout << "输入 " << m << "行 " << n << "列的矩阵" << endl;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
    M->rows = i;
    M->cols = j;
    int k = 0;      //计数器
    for (i = 0; i < M->rows; i++)
    {
        for (j = 0; j < M->cols; j++)
        {
            if (a[i][j])  //如果非 0
            {
                M->data[k].elem = a[i][j];
                M->data[k].row = i;
                M->data[k].col = j;
                k++;
            }
        }
    }
    M->num = k;

}

void TSMatrixAdd(TSMatrix *M, TSMatrix *T,TSMatrix *S)  //矩阵相加
```

```
{
    int k = M->num;
    S->num = M->num; S->rows = M->rows; S->cols = M->cols;
    for (int i = 0; i < M->num; i++)
    {
        S->data[i].elem = M->data[i].elem;
        S->data[i].row = M->data[i].row;
        S->data[i].col = M->data[i].col;
    }
    bool flag = 1;
    for (int i = 0; i < M->num; i++)
    {
        flag = 1;
        for (int j = 0; j < k; j++)
        {
            if (T->data[i].row == S->data[j].row&&T->data[i].col==S->data[j].col)
            {
                S->data[j].elem += T->data[i].elem;
                flag = 0;
                break;
            }
        }
        if (flag)
        {
            S->data[k].elem = T->data[i].elem;
            S->data[k].col = T->data[i].col;
            S->data[k].row = T->data[i].row;
            k++;
            S->num++;
        }

    }
}

void TSMatrixShow(TSMatrix *T)
{
    cout << "相加后为 : " << endl;
    bool sign = 1;
    for (int i = 0; i < T->rows; i++)
    {
        for (int j = 0; j < T->cols; j++)
        {
            sign = 1;
            for (int k = 0; k < T->num; k++)
            {
                if (T->data[k].row == i && T->data[k].col == j)
                {
```

```
93.                     cout << T->data[k].elem << " ";
94.                     sign = 0;
95.                 }
96.             }
97.             if (sign)
98.             {
99.                 cout << int(0) << " ";
100.             }
101.
102.         }
103.         cout << endl;
104.     }
105. }
106.
107. int main()
108. {
109.     int a[100][100];
110.     TSM M, T,S;
111.     M = new TSMatrix;
112.     T = new TSMatrix;
113.     S = new TSMatrix;
114.     TSMatrixInit(M, a);
115.     TSMatrixInit(T, a);
116.     TSMatrixAdd(M, T, S);
117.     TSMatrixShow(S);
118.
119. }
```

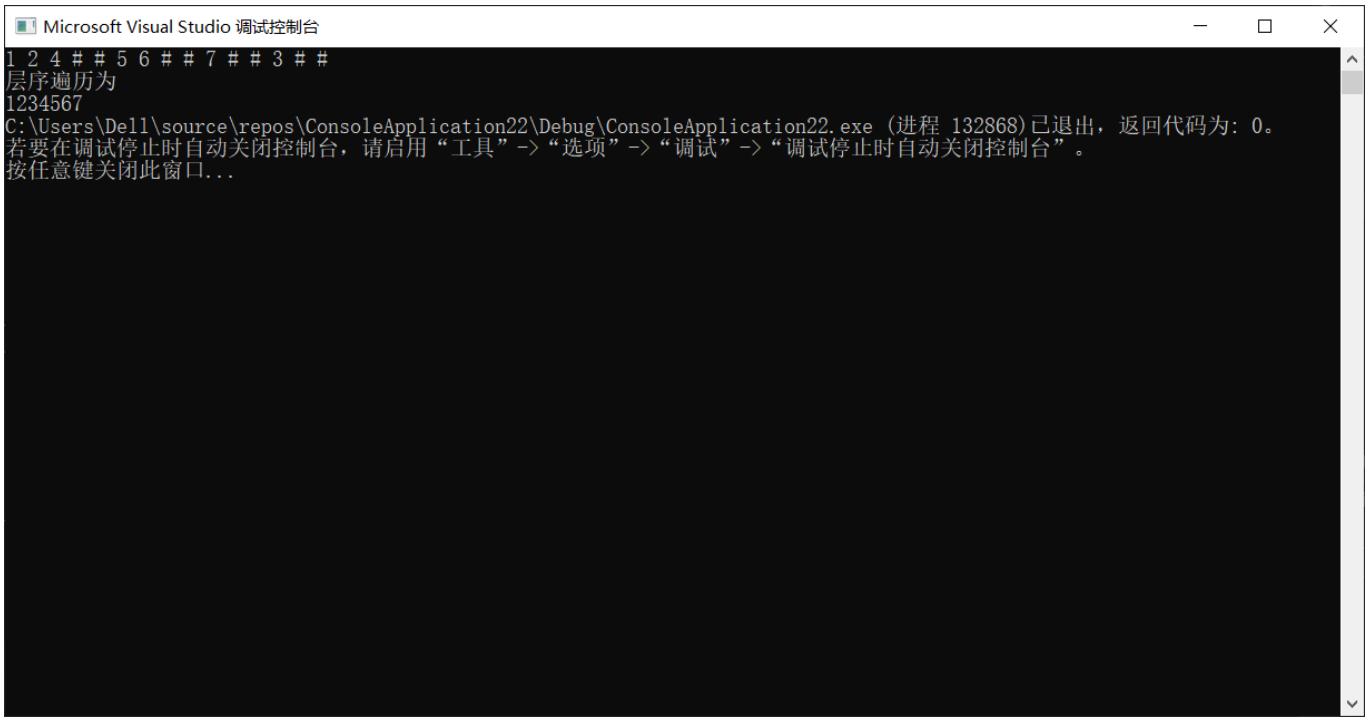
运行结果：

```
Microsoft Visual Studio 调试控制台
输入矩阵的行和列 :
3
3
输入 3行 3列的矩阵
0 1 1
0 0 1
1 0 0
输入矩阵的行和列 :
3
3
输入 3行 3列的矩阵
0 0 1
1 0 0
0 0 1
相加后为 :
0 1 2
1 0 1
1 0 1

C:\Users\Dell\source\repos\ConsoleApplication22\Debug\ConsoleApplication22.exe (进程 155204)已退出，返回代码为: 0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

5. 树-层序遍历

```
1.  void LevelOrderTraverse(BTreeNode *&T)
2.  {
3.      BTreeNode* s[100];
4.      int front=0;
5.      int rear=0;
6.      s[rear++]=T;
7.      while(front<rear)
8.  {
9.      cout<<s[front]->data<<” “;
10.     if(s[front]->lchild)
11.     {
12.         s[rear++]=s[front]->lchild;
13.     }
14.     if(s[front]->rchild)
15.     {
16.         s[rear++]=s[front]->rchild;
17.     }
18.     front++;
19. }
```

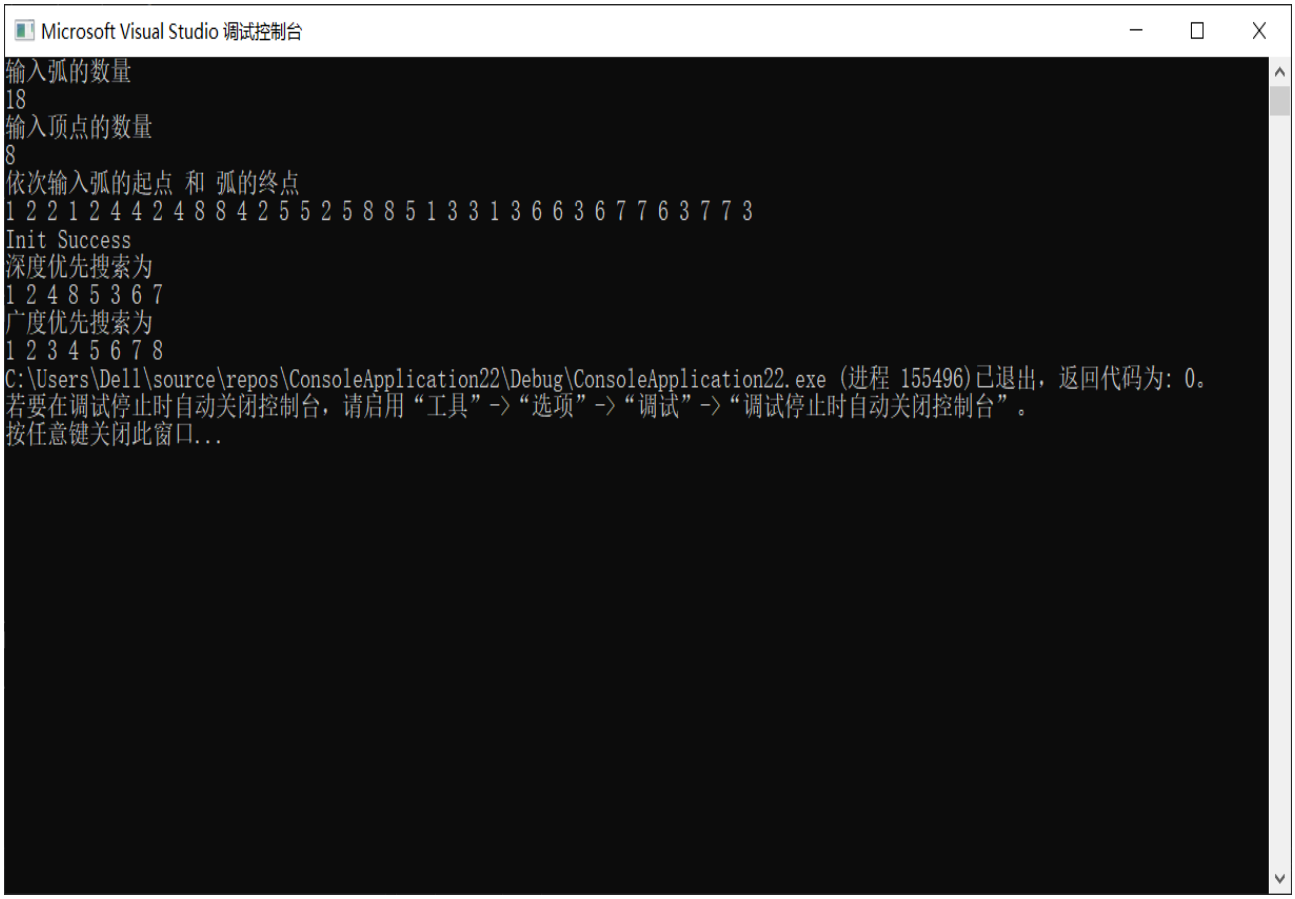
运行结果：



6. 图-深度优先搜索

```
1.  void DFS(Graph &G, int visited[100], int i)
2.  {
3.      if (!visited[i])
4.      {
5.          visited[i] = 1;
6.          cout << G.vertices[i].data << " ";
7.          ArcNode *p = G.vertices[i].firstout;
8.          while (p)
9.          {
10.             if (!visited[p->adj])
11.             {
12.                 DFS(G, visited, p->adj);
13.              }
14.             p = p->next;
15.         }
16.     }
17. }
18.
19. void showDFS(Graph &G)
20. {
21.     int visited[100];
22.     for (int i = G.vexnum - 1; i >= 0; i--)
23.     {
24.         visited[i] = 0;
25.     }
26.     for (int i = 0; i < G.vexnum; i++)
27.     {
28.         if (!visited[i])
29.         {
30.             DFS(G, visited, i);
31.         }
32.     }
33. }
```

运行结果：



7. 查找-二叉排序树

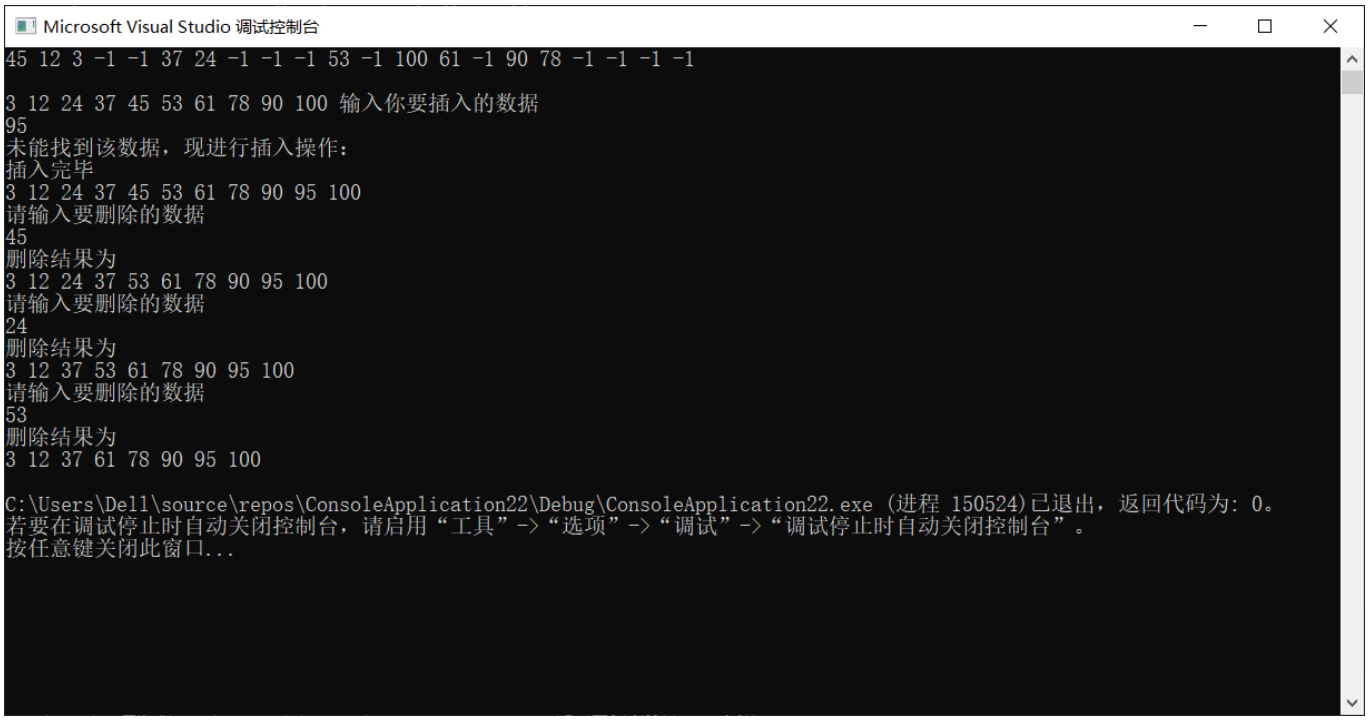
```
1.  bool SearchBTreeNode(BTreeNode *T, int num,BTreeNode *&q,BTreeNode *&cur)      //q:双亲结点
2.  {
3.      if (T)
4.      {
5.          if (T->data == num)
6.          {
7.              cur = T;                                                          //cur:该数据结
    点
8.              return true;
9.          }
10.         if (T->data > num)
11.         {
12.             q = T;
13.             return SearchBTreeNode(T->lchild, num, q,cur);
14.         }
15.         else
16.         {
```

```
            q = T;
            return SearchBTreeNode(T->rchild, num, q,cur);
        }
    }
    else
    {
        return false;
    }
}

void BTreeNodeInsert(BTreeNode *&q,int num)
{
    if (q->data > num)
    {
        q->lchild = new BTreeNode;
        q->lchild->data = num;
        q->lchild->lchild = q->lchild->rchild = NULL;
    }
    else
    {
        q->rchild = new BTreeNode;
        q->rchild->data = num;
        q->rchild->lchild = q->rchild->rchild = NULL;
    }
}

void BTreeNodeDelete(BTreeNode *&p,BTreeNode *&cur, int num)  //删除节点
   {
    if (!cur->lchild)//左子树不在就接右子树
    {
        p->rchild = cur->rchild;
    }
    else if (!cur->rchild)//右子树不在就接左子树
    {
        p->lchild = cur->lchild;
    }
    else if (cur->lchild && cur->rchild)//左右子树都有
    {
        BTreeNode *pre = cur->lchild;   //找 cur 的中序前驱
        //如果这个前驱的右子树存在
        if (pre->rchild)
        {
            BTreeNode *ppre = pre;
            while (pre->rchild)
            {
                ppre = pre;
                pre = pre->rchild;
```

```
64.             }
65.             cur->data = pre->data;      //前驱替换该点
66.             if (pre->lchild)            //如果这个前驱也有前驱
67.             {
68.                 ppre->rchild = pre->lchild;  //这个点的前驱作为它下面的一系列前驱的前驱的后继，这样就不会丢失这个点下面的所有数据。
69.             }
70.             pre = NULL;
71.         }
72.         //如果不存在，也就是这个前驱就是该点的左子树根结点
73.         else
74.         {
75.             p->lchild = pre->lchild;
76.             free(pre);
77.         }
78.     }
79.
80. }
81. int main()
82. {
83.     BTreeNode *T;
84.     T = new BTreeNode;
85.     BTN ptr = T;
86.     BTN q = T;
87.     BTN cur = NULL;
88.     BTreeBuild(ptr);
89.     BTreeShow(ptr);
90.     int num;
91.     cout << "输入你要插入的数据" << endl;
92.     cin >> num;
93.     if (!SearchBTreeNode(ptr, num,q,cur))
94.     {
95.         cout << "未能找到该数据，现进行插入操作：" << endl;
96.         BTreeNodeInsert(q, num);
97.         cout << "插入完毕" << endl;
98.     }
99.     BTreeShow(ptr);
100.     cout << endl;
101.     for (int i = 0; i < 3; i++)
102.     {
103.         q = ptr;  //重置
104.         cout << "请输入要删除的数据" << endl;
105.         int delnum;
106.         cin >> delnum;
107.         if (!SearchBTreeNode(ptr, delnum, q, cur))
108.         {
109.             cout << "该数据不存在，无法删除" << endl;
```

```
110.         }
111.         else
112.         {
113.             cout << "删除结果为" << endl;
114.             BTreeNodeDelete(q, cur, delnum);
115.             BTreeShow(ptr);
116.             cout << endl;
117.         }
118.     }
119. }
120. //45 12 3 -1 -1 37 24 -1 -1 -1 53 -1 100 61 -1 90 78 -1 -1 -1 -1
```

运行结果：



8. 排序-快速排序算法

```
1.  void QuickSort(SqList &L,int low,int high)      //快排
2.  {
3.      if(low<high)
4.      {
5.          int m;
6.          L.r[0].key=L.r[low].key;
7.          m=L.r[0].key;
```

```
8.          int i=low;
9.          int j=high;
10.         while(i<j)
11.         {
12.             while(i<j&&L.r[j].key>=m)
13.             {
14.                 j--;
15.             }
16.             while(i<j&&L.r[i].key<=m)
17.             {
18.                 i++;
19.             }
20.             if(i<j)
21.             {
22.                 int temp;
23.                 temp=L.r[i].key;
24.                 L.r[i].key=L.r[j].key;
25.                 L.r[j].key=temp;
26.             }
27.         }
28.         //此时 low=high,调换中枢和第一个
29.         L.r[low].key=L.r[i].key;
30.         L.r[i].key=L.r[0].key;
31.
32.         QuickSort(L,i+1,high);
33.         QuickSort(L,low,i-1);
34.     }
35. }
```

运行结果：