

华为上机题总结

1. 选秀节目打分，分为专家评委和大众评委，`score[]` 数组里面存储每个评委打的分数，`judge_type[]` 里存储与 `score[]` 数组对应的评委类别，`judge_type[i] == 1`，表示专家评委，`judge_type[i] == 2`，表示大众评委，`n` 表示评委总数。打分规则如下：专家评委和大众评委的分数先分别取一个平均分（平均分取整），然后，总分 = 专家评委平均分 * 0.6 + 大众评委 * 0.4，总分取整。如果没有大众评委，则 总分 = 专家评委平均分，总分取整。函数最终返回选手得分。

函数接口 `int cal_score(int score[], int judge_type[], int n)`

[view plain](#)

```
1. #include "iostream"
2. using namespace std;
3.
4. int cal_score(int score[], int judge_type[], int n)
5. {
6.     if (NULL==score||NULL==judge_type||0==n)
7.         return 0;
8.
9.     int sum=0;
10.    int sum1=0,count1=0;
11.    int sum2=0,count2=0;
12.    for(int i=0;i<n;i++)
13.    {
14.        if (judge_type[i]==1)
15.        {
16.            sum1=sum1+score[i];
17.            count1++;
18.        }
19.        else
20.        {
21.            sum2=sum2+score[i];
22.            count2++;
23.        }
24.    }
25.    if(0==count2)
26.        sum=sum1/count1;
27.    else
28.        sum=(sum1/count1)*0.6+(sum2/count2)*0.4;
29.    return sum;
30. }
```

```

31. void main()
32. {
33.     int score[3]={12,13,15};
34.     int judge_type[3]={1,1,2};
35.     printf("%d",cal_score(score, judge_type, 3) );
36.
37. }
38.
39.

```

//2. 给定一个数组 `input[]`，如果数组长度 `n` 为奇数，则将数组中最大的元素放到 `output[]` 数组最中间的位置，如果数组长度 `n` 为偶数，则将数组中最大的元素放到 `output[]` 数组中间两个位置偏右的那个位置上，然后再按从大到小的顺序，依次在第一个位置的两边，按照一左一右的顺序，依次存放剩下的数。例如：`input[] = {3, 6, 1, 9, 7}` `output[] = {3, 7, 9, 6, 1}`;
`input[] = {3, 6, 1, 9, 7, 8}` `output[] = {1, 6, 8, 9, 7, 3}`

函数接口 `void sort(int input[], int n, int output[])`

[view plain](#)

```

1. #include "iostream"
2. using namespace std;
3. void bubblesort(int data[],int n)
4. {
5.     int temp=0;
6.     for(int i=0;i<n;i++ )
7.     {
8.         for (int j=i+1;j<n;j++)
9.         {
10.            if (data[i]<data[j])
11.            {
12.                temp=data[i];
13.                data[i]=data[j];
14.                data[j]=temp;
15.            }
16.        }
17.    }
18. }
19.
20. void sort(int input[], int n, int output[])
21. {
22.     int *sort_input=new int[n];
23.     for(int i=0;i<n;i++)
24.     {

```

```

25.  sort_input[i]=input[i];
26. }
27. bubblesort(sort_input,n);
28. if (1==n%2)
29. {
30.     int mid=n/2;
31.     int k=0;
32.     output[mid]=sort_input[k++];
33.     for(int j=1;j<=n/2;j++)
34.     {
35.         output[mid-j]=sort_input[k++];
36.         output[mid+j]=sort_input[k++];
37.     }
38.
39. }
40. else
41. {
42.     int mid=n/2;
43.     int k=0;
44.     output[mid]=sort_input[k++];
45.     for(int j=1;j<n/2;j++)
46.     {
47.         output[mid-j]=sort_input[k++];
48.         output[mid+j]=sort_input[k++];
49.     }
50.     output[0]=sort_input[k++];
51.
52. }
53.
54.     delete sort_input;
55. }
56.
57.
58. void main()
59. {
60.     int input1[] = {3, 6, 1, 9, 7};
61.     int output1[5];
62.     memset(output1,0,5*sizeof(int));
63.     int input2[] = {3, 6, 1, 9, 7, 8} ;
64.     int output2[6];
65.     memset(output2,0,6*sizeof(int));
66.
67.     sort(input1, 5, output1);
68.     sort(input2, 6, output2);

```

```

69. for(int k=0;k<5;k++)
70.     printf("%d ",output1[k]);
71. for(k=0;k<6;k++)
72.     printf("%d ",output2[k]);
73. }

```

// 3.操作系统任务调度问题。操作系统任务分为系统任务和用户任务两种。其中，系统任务的优先级 < 50 ，用户任务的优先级 ≥ 50 且 ≤ 255 。优先级大于 255 的为非法任

务，应予以剔除。现有一任务队列 `task[]`，长度为 `n`，`task` 中的元素值表示任务的优先级，数值越小，优先级越高。函数 `scheduler` 实现如下功能，将 `task[]` 中的任务按照系统任

务用户任务依次存放到 `system_task[]` 数组和 `user_task[]` 数组中（数组中元素的值是任务在 `task[]` 数组中的下标），并且优先级高的任务排在前面，优先级相同的任务按

照入队顺序排列（即先入队的任务排在前面），数组元素为 -1 表示结束。例如：`task[] = {0, 30, 155, 1, 80, 300, 170, 40, 99}` `system_task[] = {0, 3, 1, 7, -1}` `user_task[] = {4, 8, 2, 6, -1}`

// 函数接口 `void scheduler(int task[], int n, int system_task[], int user_task[])`

[view plain](#)

```

1. #include "iostream"
2. using namespace std;
3. void change(int *a,int *b)
4. {
5.     int temp=*a;
6.     *a=*b;
7.     *b=temp;
8. }
9.
10. void bubblesort(int data[],int n,int index[])//冒泡排序并记录排序后下标
11. {
12.     int temp=0;
13.     for(int j=0;j<n;j++)
14.         index[j]=j;
15.     for(int i=0;i<n;i++ )
16.     {
17.         for (int j=i+1;j<n;j++)
18.         {
19.             if (data[i]>data[j])
20.             {
21.                 change(&data[i],&data[j]);

```

```

22.     change(&index[i],&index[j]);
23. }
24. }
25.
26. }
27. }
28.
29. void scheduler(int task[], int n, int system_task[], int user_task[])
30. {
31.
32.     int *sort_task=new int[n];
33.     int *index=new int[n];
34.     for(int i=0;i<n;i++)
35.     {
36.         sort_task[i]=task[i];
37.     }
38.     bubblesort(sort_task,n,index);
39.     i=0;
40.     while (sort_task[i]<50)
41.     {
42.         system_task[i]=index[i];
43.         i++;
44.     }
45.
46.     system_task[i]=-1;
47.
48.     int k=0;
49.     while (sort_task[i]>50&&sort_task[i]<=255)
50.     {
51.         user_task[k++]=index[i++];
52.     }
53.
54.     user_task[k]=-1;
55.     delete sort_task;
56.     delete index;
57.
58.
59. }
60. void main()
61. {
62.     int task[] = {0, 30, 155, 1, 80, 300, 170, 40, 99} ;
63.     int n=sizeof(task)/sizeof(int);
64.     int *system_task=new int[n];
65.     int *user_task=new int[n];

```

```

66.
67.     scheduler(task, n,  system_task, user_task);
68.
69.     for(int k=0;k<n;k++)
70.     {
71.
72.         printf("%d  ",system_task[k]);
73.         if (system_task[k]!=-1)
74.         {
75.             printf("%d  ",system_task[k]);
76.         }
77.         else
78.         {
79.             printf("%d  ",system_task[k]);
80.             break;}
81.
82.     }
83.     printf("\n");
84.     for(k=0;k<n;k++)
85.     {
86.         printf("%d  ",user_task[k]);
87.         if (user_task[k]!=-1)
88.         {
89.             printf("%d  ",user_task[k]);
90.         }
91.         else{
92.             printf("%d  ",system_task[k]);
93.             break;}
94.
95.     }
96. }

```

4.删除字符串中所有给定的子串

问题描述：

在给定字符串中查找所有特定子串并删除，如果没有找到相应子串，则不作任何操作。

要求实现函数：

int delete_sub_str(const char *str, const char *sub_str, char *result_str)

【输入】 str: 输入的被操作字符串

sub_str: 需要查找并删除的特定子字符串

【输出】 result_str: 在 str 字符串中删除所有 sub_str 子字符串后的结果

【返回】 删除的子字符串的个数

[view plain](#)

```
1. #include <stdio.h>
2. #include <string.h>
3. #include "string"
4. int delete_sub_str(const char *str, const char *sub_str, char *result_str)
5. {
6.     int count = 0;
7.     int k=0;
8.     int n=strlen(str);
9.     for (int i=0;i<n;i++)
10.    {
11.        while (str[i]!=sub_str[0]&&str[i]!='\0')
12.        {
13.            result_str[k++]=str[i];
14.            i++;
15.        }
16.        int j=0;
17.        int temp=i;
18.        while (str[i]==sub_str[j]&&str[i]!='\0'&&sub_str[j]!='\0')
19.        {
20.            i++;
21.            j++;
22.        }
23.        if (sub_str[j]=='\0')
24.        {
25.            count++;
26.        }
27.        else
28.        {
29.            while (temp<i)
30.            {
31.                result_str[k++]=str[temp];
32.                temp++;
33.            }
34.
35.
36.        }
37.
38.        i--;
39.    }
```

```

40.     result_str[k]='\0';
41.     return count;
42.
43. }
44.
45. int main()
46. {
47.     char *str = "aaddde";
48.     char *sub = "ad";
49.     char res[50] = "";
50.     int count = delete_sub_str(str, sub, res);
51.     printf("子字符串的个数是: %d\n", count);
52.     printf("删除子字符串后: %s\n", res);
53.     return 0;
54. }
55.
56.

```

5. 高精度整数加法

问题描述:

在计算机中, 由于处理器位宽限制, 只能处理有限精度的十进制整数加减法, 比如在 32 位宽处理器计算机中,

参与运算的操作数和结果必须在 $-2^{31} \sim 2^{31}-1$ 之间。如果需要进行更大范围的十进制整数加法, 需要使用特殊

的方式实现, 比如使用字符串保存操作数和结果, 采取逐位运算的方式。如下:

$9876543210 + 1234567890 = ?$

让字符串 num1="9876543210", 字符串 num2="1234567890", 结果保存在字符串 result = "11111111100"。

$-9876543210 + (-1234567890) = ?$

让字符串 num1="-9876543210", 字符串 num2="-1234567890", 结果保存在字符串 result = "-11111111100"。

要求编程实现上述高精度的十进制加法。

要求实现函数:

```
void add (const char *num1, const char *num2, char *result)
```

【输入】num1: 字符串形式操作数 1, 如果操作数为负, 则 num1[0]为符号位 '-'

num2: 字符串形式操作数 2, 如果操作数为负, 则 num2[0]为符号位 '-'

【输出】result: 保存加法计算结果字符串, 如果结果为负, 则 result[0]为符号位 '-'。

注:

1、 当输入为正数时, '+'不会出现在输入字符串中; 当输入为负数时, '-'会出现在输入字符

串中，且一定在输入字符串最左边位置；

II、 输入字符串所有位均代表有效数字，即不存在由'0'开始的输入字符串，比如"0012"，"-0012"不会出现；

III、 要求输出字符串所有位均为有效数字，结果为正或 0 时 '+' 不出现在输出字符串，结果为负时输出字符串最左边位置为 '-'。

[view plain](#)

```
1. #include "string.h"
2. #include "iostream"
3. using namespace std;
4. void reverse(char *str)
5. {
6.     int i=0;
7.     int len=strlen(str);
8.     int temp=0;
9.     while (i<(len+1)/2)
10.    {
11.        temp=str[i];
12.        str[i]=str[len-1-i];
13.        str[len-1-i]=temp;
14.        i++;
15.    }
16. }
17.
18. void sub (const char *num1, const char *num2, char *result) //num1-num2
19. {
20.     int num1len=strlen(num1)-1;
21.     int num2len=strlen(num2)-1;
22.     int t1=0;
23.     int t2=0;
24.     if (num1[0]=='-')
25.         t1=1;
26.     else
27.         t2=1;
28.     int k=0;
29.     int t=0;
30.     int *subnum1=new int[num1len+1];    ////借位
31.     memset(subnum1,0,(num1len+1)*sizeof(int));
32.     while (num1len>=t1&&num2len>=t2)
33.     {
34.         if (num1[num1len]+subnum1[num1len]<num2[num2len])
35.         {
36.             int q=num1len-1;
```

```

37.         while (num1[q]=='0')
38.         {
39.             subnum1[q]=9;           //借位
40.             q--;
41.         }
42.         subnum1[q]-=1;
43.         result[k++]=(num1[num1len]-'0')+subnum1[num1len]+10-(num2[num2le
n]-'0')+ '0';
44.     }
45.     else
46.         result[k++]=(num1[num1len]-'0')+subnum1[num1len]-(num2[num2len]-
'0')+ '0';
47.     num1len--;
48.     num2len--;
49. }
50.
51. while (num1len>=t1)
52. {
53.     result[k++]=num1[num1len]+subnum1[num1len];
54.     num1len--;
55. }
56. k--;
57. while (result[k]-'0'==0)
58. {
59.     k--;
60. }
61. k++;
62. if (num1[0]=='-')
63.     result[k++]=num1[0];
64. result[k]='\0';
65.
66. }
67.
68.
69. void add (const char *num1, const char *num2, char *result)
70. {
71.     int num1len=strlen(num1)-1;
72.     int num2len=strlen(num2)-1;
73.     int t=0;           //进位
74.     int k=0;
75.     if (num1[0]=='-'&&num2[0]=='-' || (num1[0]!='-'&&num2[0]!='-')) //作加法
运算
76.     {
77.         int t1=num1len;

```

```

78.     int t2=num2len;
79.     int t11=0;
80.     if (num1[0]=='-'&&num2[0]=='-')
81.         t11=1;
82.     while (t1>=t11&&t2>=t11)
83.     {
84.         result[k++]=((num1[t1]-'0')+(num2[t2]-'0')+t)%10+'0';
85.         t=((num1[t1]-'0')+(num2[t2]-'0')+t)/10;
86.         t1--;
87.         t2--;
88.     }
89.     while (t2>=t11)
90.     {
91.         result[k++]=((num2[t2]-'0')+t)%10+'0';
92.         t=((num2[t2]-'0')+t)/10;
93.         t2--;
94.     }
95.
96.     while (t1>=t11)
97.     {
98.         result[k++]=((num1[t1]-'0')+t)%10+'0';
99.         t=((num1[t1]-'0')+t)/10;
100.        t1--;
101.    }
102.
103.    if (t!=0)
104.        result[k++]=t+'0';
105.    if (num1[0]=='-'&&num2[0]=='-')
106.    {
107.        result[k++]='-';
108.    }
109.    result[k]='\0';
110.    reverse(result);
111. }
112. else
113. {
114.     if (num1[0]=='-')
115.     {
116.         if (num1len==num2len+1)
117.         {
118.             int j=0;
119.             while (j<num1len)
120.             {
121.                 if (num1[j+1]>num2[j])

```

```

122.         {
123.             sub(num1,num2, result);
124.             reverse(result);
125.             break;
126.         }
127.         else if(num1[j+1]<num2[j])
128.         {
129.             sub(num2,num1, result);
130.             reverse(result);
131.             break;
132.         }
133.         else
134.             j++;
135.     }
136.
137. }
138. else if(num1len>num2len+1)
139. {
140.     sub(num1,num2, result);
141.     reverse(result);
142. }
143. else if(num1len<num2len+1)
144. {
145.     sub(num2,num1, result);
146.     reverse(result);
147. }
148.
149. }
150. else if (num2[0]=='-')
151. {
152.     if (num1len+1==num2len)
153.     {
154.         int j=0;
155.         while (j<num2len)
156.         {
157.             if (num1[j]>num2[j+1])
158.             {
159.                 sub(num1,num2, result);
160.                 reverse(result);
161.                 break;
162.             }
163.             else if(num1[j+1]<num2[j])
164.             {
165.                 sub(num2,num1, result);

```

```

166.         reverse(result);
167.         break;
168.     }
169.     else
170.         j++;
171.     }
172.
173.     }
174.     else if(num1len+1>num2len)
175.     {
176.         sub(num1,num2, result);
177.         reverse(result);
178.     }
179.     else if(num1len+1<num2len)
180.     {
181.         sub(num2,num1, result);
182.         reverse(result);
183.     }
184.
185.     }
186.
187.
188.     }
189. }
190.
191. int main()
192. {
193.     char *a = "-10000";
194.     char *b = "33";
195.     char res[200];
196.     add(a, b, res);
197.     printf("%s\n", res);
198.     return 0;
199. }

```

6.数组比较

问题描述：

比较两个数组，要求从数组最后一个元素开始逐个元素向前比较，如果 2 个数组长度不等，则只比较较短长度数组个数元素。请编程实现上述比较，并返回比较中**发现的不相等**

元素的个数

比如：

数组{1,3,5}和数组{77,21,1,3,5}按题述要求比较，不相等元素个数为 0

数组{1,3,5}和数组{77,21,1,3,5,7}按题述要求比较，不相等元素个数为 3

要求实现函数：

`int array_compare(int len1, int array1[], int len2, int array2[])`

【输入】 `int len1`：输入被比较数组 1 的元素个数；

`int array1[]`：输入被比较数组 1；

`int len2`：输入被比较数组 2 的元素个数；

`int array2[]`：输入被比较数组 2；

【输出】 无

【返回】 不相等元素的个数，类型为 `int`

示例

1) 输入：`int array1[] = {1,3,5}`，`int len1 = 3`，`int array2[] = {77,21,1,3,5}`，`int len2 = 5`

函数返回：0

2) 输入：`int array1[] = {1,3,5}`，`int len1 = 3`，`int array2[] = {77,21,1,3,5,7}`，`int len2 = 6`

函数返回：3

[view plain](#)

```
1. #include <stdio.h>
2. #include <string.h>
3. #include "string"
4. #include "assert.h"
5. int array_compare(int len1, int array1[], int len2, int array2[])
6. {
7.     assert(array1!=NULL&&array1!=NULL);
8.     int count=0;
9.     while (len1!=0&&len2!=0)
10.    {
11.        if (array1[len1-1]!=array2[len2-1])
12.        {
13.            count++;
14.        }
15.        len1--;
16.        len2--;
17.    }
18.    return count;
19.
20. }
21.
22. int main()
23. {
24.     int array1[] = {1,3,5};
```

```

25.     int len1 = 3;
26.     int array2[] = {77,21,1,3,5};
27.     int len2 = 5;
28.     printf("%d",array_compare(len1, array1,len2, array2));
29.     int array3[] = {1,3,5};
30.     len1 = 3;
31.     int array4[] = {77,21,1,3,5,7};
32.     len2 = 6;
33.     printf("%d",array_compare(len1, array3,len2, array4));
34. }

```

7.约瑟夫问题

问题描述：

输入一个由随机数组成的数列（数列中每个数均是大于 0 的整数，长度已知），和初始计数值 m 。从数列首位置开始计数，计数到 m 后，将数列该位置数值替换计数值 m ，并将数列

该位置数值出列，然后从下一位置从新开始计数，直到数列所有数值出列为止。如果计数到达数列尾段，则返回数列首位置继续计数。请编程实现上述计数过程，同时输出数

值出列的顺序

比如： 输入的随机数列为：3,1,2,4，初始计数值 $m=7$ ，从数列首位置开始计数（数值 3 所在位置）

第一轮计数出列数字为 2，计数值更新 $m=2$ ，出列后数列为 3,1,4，从数值 4 所在位置从新开始计数

第二轮计数出列数字为 3，计数值更新 $m=3$ ，出列后数列为 1,4，从数值 1 所在位置开始计数

第三轮计数出列数字为 1，计数值更新 $m=1$ ，出列后数列为 4，从数值 4 所在位置开始计数

最后一轮计数出列数字为 4，计数过程完成。

输出数值出列顺序为：2,3,1,4。

要求实现函数：

`void array_iterate(int len, int input_array[], int m, int output_array[])`

【输入】 `int len`：输入数列的长度；

`int input_array[]`：输入的初始数列

`int m`：初始计数值

【输出】 `int output_array[]`：输出的数值出列顺序

【返回】 无

示例

输入: int input_array[] = {3,1,2,4}, int len = 4, m=7

输出: output_array[] = {2,3,1,4}

[view plain](#)

```
1. #include "iostream"
2. using namespace std;
3. struct Node
4. {
5.     int data;
6.     Node *next;
7. };
8. void array_iterate(int len, int input_array[], int m, int output_array[])
9. {
10.    if (NULL==input_array||0==m||0==len)
11.        return 0;
12.    Node *node=new Node;
13.    Node *head=new Node;
14.    head->data=input_array[0];
15.    Node *p=head;
16.    int k=0;
17.    for (int i=1;i<len;i++)
18.    {
19.        node=new Node;
20.        node->data=input_array[i];
21.        p->next=node;
22.        p=node;
23.    }
24.    p->next=head;
25.    p=head;
26.    Node *q=new Node;
27.    int count;
28.    while (p->next!=p)
29.    {
30.        count=1;
31.        while (count<m)
32.        {
33.            q=p;
34.            p=p->next;
35.            count++;
36.        }
37.        m=p->data;
```



```

38.         output_array[k++]=m;
39.         q->next=p->next;
40.         p=q->next;
41.     }
42.     output_array[k]=p->data;
43. }
44. void main()
45. {
46.     int a[]={3,1,2,4};
47.     int b[4];
48.     memset(b,0,4*sizeof(int));
49.     array_iterate(4, a, 7,b);
50.     printf("%d,%d,%d,%d",b[0],b[1],b[2],b[3]);
51. }

```

8. 简单四则运算

问题描述:

输入一个只包含个位数字的简单四则运算表达式字符串，计算该表达式的值

注： 1、表达式只含 +, -, *, / 四则运算符，不含括号

2、表达式数值只包含个位整数(0-9)，且不会出现 0 作为除数的情况

3、要考虑加减乘除按通常四则运算规定的计算优先级

4、除法用整数除法，即仅保留除法运算结果的整数部分。比如 $8/3=2$ 。输入表达式保证无 0 作为除数情况发生

5、输入字符串一定是符合题意合法的表达式，其中只包括数字字符和四则运算符字符，除此之外不含其它任何字符，不会出现计算溢出情况

要求实现函数：

int calculate(int len,char *expStr)

【输入】 int len: 字符串长度;

char *expStr: 表达式字符串;

【输出】 无

【返回】 计算结果

示例

1) 输入: char *expStr = "1+4*5-8/3"

函数返回: 19

2) 输入: char *expStr = "8/3*3"

函数返回: 6

[view plain](#)

```
1. #include <stdio.h>
2. #include <string.h>
3. #include "string"
4. #include "assert.h"
5. struct stack{           //存放后续排列的表达式，模拟栈
6.     char str[80];
7.     int top;
8. };
9. struct sstack{          //存放计算表达式的值，模拟栈
10.    int str[80];
11.    int top;
12. };
13. int calculate(int len,char *expStr)
14. {
15.     char *postexp=new char[len+1];
16.     stack opstack;
17.     sstack calstack;
18.     calstack.top=-1;
19.     opstack.top=-1;
20.     int i=0;
21.     int k=0;
22.     while(expStr[i]!='\0')
23.     {
24.         if (expStr[i]>='0'&&expStr[i]<='9')
25.         {
26.             postexp[k++]=expStr[i];
27.         }
28.         else if (expStr[i]=='+'||expStr[i]=='-')
29.         {
30.             while (opstack.top>=0)
31.             {
32.                 postexp[k++]=opstack.str[opstack.top--];
33.             }
34.             opstack.top++;
35.             opstack.str[opstack.top]=expStr[i];
36.         }
37.     }
38.     else if (expStr[i]=='*'||expStr[i]=='/')
39.     {
```

```

40.         while (opstack.top>=0&&(opstack.str[opstack.top]=='*' || opstack.s
tr[opstack.top]=='/'))
41.         {
42.             postexp[k++]=opstack.str[opstack.top--];
43.         }
44.         opstack.top++;
45.         opstack.str[opstack.top]=expStr[i];
46.     }
47.
48.     i++;
49. }
50. while (opstack.top>=0)
51. {
52.     postexp[k++]=opstack.str[opstack.top--];
53. }
54. int temp1=0;
55. int temp2=0;
56. for (i=0;i<len;i++)
57. {
58.     if (postexp[i]>='0'&&postexp[i]<='9')
59.     {
60.         calstack.top++;
61.         calstack.str[calstack.top]=postexp[i]-'0';
62.     }
63.     else if (postexp[i]=='+')
64.     {
65.         temp1=calstack.str[calstack.top--];
66.         temp2=calstack.str[calstack.top];
67.         calstack.str[calstack.top]=temp2+temp1;
68.     }
69.     else if (postexp[i]=='-')
70.     {
71.         temp1=calstack.str[calstack.top--];
72.         temp2=calstack.str[calstack.top];
73.         calstack.str[calstack.top]=temp2-temp1;
74.     }
75.     else if (postexp[i]=='*')
76.     {
77.         temp1=calstack.str[calstack.top--];
78.         temp2=calstack.str[calstack.top];
79.         calstack.str[calstack.top]=temp2*temp1;
80.     }
81.     else if (postexp[i]=='/')
82.     {

```

```

83.         temp1=calstack.str[calstack.top--];
84.         temp2=calstack.str[calstack.top];
85.         calstack.str[calstack.top]=temp2/temp1;
86.     }
87. }
88. printf("%d",calstack.str[calstack.top]);
89. return calstack.str[calstack.top];
90.
91. }
92. int main()
93. {
94.     char *expStr = "1+4*5-8/3";
95.     int len=strlen(expStr);
96.     calculate(len,expStr);
97.
98. }

```

9.

一副牌中发五张扑克牌给你：让你判断数字的组成：

有以下几种情况：

- 1: 四条：即四张一样数值的牌（牌均不论花色）
- 2: 三条带 一对
- 3: 三条带两张不相同数值的牌
- 4: 两对
- 5: 顺子 包括 10, J, Q, K, A
- 6: 什么都不是
- 7: 只有一对

[view plain](#)

```

1. void sort(int data[],int n)
2. {
3.     int temp=0;
4.     for(int i=0;i<n;i++ )
5.     {
6.         for (int j=i+1;j<n;j++)
7.         {
8.             if (data[i]<data[j])
9.             {
10.                 temp=data[i];
11.                 data[i]=data[j];

```

```

12.         data[j]=temp;
13.     }
14. }
15. }
16. }
17.
18. void test(int a[],int len)
19. {
20.     int *b=new int[len];
21.     int count=0;
22.     bool temp=false;
23.     for (int i=0;i<len;i++)
24.     {
25.         b[i]=a[i];
26.     }
27.     sort(b,5);
28.     for (i=0;i<len-1;i++)
29.     {
30.         if (b[i]==b[i+1])
31.             count++;
32.     }
33.     switch (count)
34.     {
35.     case 0:
36.         if (b[0]-b[4]==4||(b[0]-b[3]==3&&b[4]==1))
37.         {
38.             printf("顺子");
39.         }
40.         else
41.             printf("什么都不是");
42.         break;
43.     case 1:
44.         printf("只有一对");
45.         break;
46.     case 2:
47.         for (i=0;i<3;i++)
48.         {
49.             if (b[i]==b[i+2])
50.             {
51.                 printf("三条带两张不相同数值的牌");
52.                 temp=true;
53.                 break;
54.             }
55.         }

```

```
56.     if (!temp)
57.     {
58.         printf("两对");
59.     }
60.     break;
61. case 3:
62.     if (b[1]==b[3])
63.         printf("四条: 即四张一样数值的牌");
64.     else
65.         printf("三条带 一对");
66.
67.     break;
68.
69. }
70. }
71.
72. int main()
73. {
74.     int a[5]={1,10,11,12,13};
75.     test(a,5);
76. }
```