二维数组与字符串数组的指针实现

2022 年 8 月 16 日

1. 有一篇文章，共有若干行文字，每行有若干个字符。要求分别统计出其中英文大写字母、小写字母、数字、空格以及其它字符的个数。每行文字都以字符串的形态存储。文章的数据声明和定义如下。

```
const char PASSAGE[][1024] = {
       "Once upon a time, there was a wolf living on a grassland in west China.",
       "It loved eating sheep.",
       "It often drank water at a river.",
       "There were also 25 houses there."};
```

```
#include <stdio.h>
#include <string.h>
const char PASSAGE[][1024] = {
                     "Once upon a time, there was a wolf living on a grassland in
                         west China.",
                     "It loved eating sheep.",
                     "It often drank water at a river.",
                     "There were also 25 houses there."};

int main()
{
    int upper_case_letter_num = 0, lower_case_letter_num = 0;
    int digit_num = 0, space_num = 0, other_num = 0;
    for(int i = 0; i < 4; i++)
    {
        // PASSAGE[i]为i号行的数组名，
        // 因为数组名自动转化为首元素地址，所以PASSAGE[i]自动转化为i号行的首元素地址
        char *p = PASSAGE[i]; // p被赋值为i号行首元素地址，也就是p指向i号行的首元素
```

```
        while(*p) // p未指向字符串的末端
        {
            // 注意p[j]和*(p+j)完全等价
            char ch = p[j]; //p[j]表示从p开始后移j个单元得到的元素（字符）
            if(ch >= 'A' && ch <= 'Z') // 大写字母
            {
                upper_case_letter_num++;
            }
            else if(ch >= 'a' && ch <= 'z') // 小写字母
            {
                lower_case_letter_num++;
            }
            else if(ch >= '0' && ch <= '9') // 数字
            {
                digit_num++;
            }
            else if(ch == ' ') // 空格
            {
                space_num++;
            }
            else
            {
                other_num++;
            }
        }
        p++; // p指向下一个元素（字符）
    }
    printf("big letter num: %d\n", upper_case_letter_num);
    printf("small letter num: %d\n", lower_case_letter_num);
    printf("digit num: %d\n", digit_num);
    printf("space num: %d\n", space_num);
    printf("other num: %d\n", other_num);
    return 0;
}
```

2. 定义两个 5×4 的二维矩阵 A 和 B，分别对这两个列表输入数据，求 $A+B$ 和 $A-B$ 的值。$A+B$ 就是把所有的 a_{ij} 和 b_{ij} 对应相加，并且把所得的和保存在另一矩阵的第 i 行第 j 列。减法是类似的。

```c
#include <stdio.h>
int main()
{
    int a[5][4]; // 定义矩阵A
    int b[5][4]; // 定义矩阵B
    int sum_mtx[5][4]; // 定义和矩阵
    int diff_mtx[5][4]; // 定义差矩阵

    printf("Input the first 5*4 matrix:\n");
    for(int i = 0; i < 5; i++)
    {
        printf("Input the %d-th row:\n", i + 1);
        for(int j = 0; j < 4; j++)
        {
            scanf("%d", *(a + i) + j); // 输入a[i][j]的值
        }
    }

    printf("Input the second 5*4 matrix:\n");
    for(int i = 0; i < 5; i++)
    {
        printf("Input the %d-th row:\n", i + 1);
        for(int j = 0; j < 4; j++)
        {
            scanf("%d", *(b + i) + j); // 输入b[i][j]的值
        }
    }

    printf("The first 5*4 matrix:\n");
    for(int i = 0; i < 5; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%d\t", *(*(a + i) + j));
        }
        printf("\n");
    }

    printf("The second 5*4 matrix:\n");
```

```
40      for(int i = 0; i < 5; i++)
41      {
42          for(int j = 0; j < 4; j++)
43          {
44              printf("%d\t", *(*(b + i) + j));
45          }
46          printf("\n");
47      }
48
49      // 求和
50      for(int i = 0; i < 5; i++)
51      {
52          for(int j = 0; j < 4; j++)
53          {
54              *(*(sum_mtx + i) + j) = *(*(a + i) + j) + *(*(b + i) + j);
55          }
56      }
57
58      // 求差
59      for(int i = 0; i < 5; i++)
60      {
61          for(int j = 0; j < 4; j++)
62          {
63              *(*(diff_mtx + i) + j) = *(*(a + i) + j) - *(*(b + i) + j);
64          }
65      }
66
67      printf("The sum matrix:\n");
68      for(int i = 0; i < 5; i++)
69      {
70          for(int j = 0; j < 4; j++)
71          {
72              printf("%d\t", *(*(sum_mtx + i) + j));
73          }
74          printf("\n");
75      }
76
77      printf("The diff matrix:\n");
78      for(int i = 0; i < 5; i++)
```

```
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%d\t", *(*(diff_mtx + i) + j));
        }
        printf("\n");
    }
    return 0;
}
```

3. 编制程序，将 $M \times N$ 的矩阵转置。矩阵 $A = (a_{ij})_{m \times n}$ 的转置就是把所有的 a_{ij} 和相应的 a_{ji} 进行对换。设 $M = 3$，$N = 4$。

```
#include <stdio.h>
int main()
{
    int a[3][4];

    printf("Input a 3*4 matrix:\n");
    int *p = &a[0][0];
    // 输入3*4的矩阵
    while(p < (*a) + 3 * 4)
    {
        if((p - (*a)) % 4 == 0)
        {
           printf("Input the %d-th row:\n", (p - (*a)) / 4 + 1);
        }
        scanf("%d", p);
        p++;
    }

    printf("The input matrix:");
    printf("The transposed matrix is:\n");
    while(p < (*a) + 3 * 4)
    {
        printf("%d\t", *p);
        p++;
        if((p - (*a)) % 4 == 0)
        {
```

```
            printf("\n");
        }
    }

    // 转置
    int b[4][3];
    p = &a[0][0];
    for(int j = 0; j < 3; j++)
    {
        for(int i = 0; i < 4; i++)
        {
            b[i][j] = *p++; // 复制到恰当位置
        }
    }

    printf("The transposed matrix is:\n");
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            printf("%d\t", b[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

4. 有一个用户输入的 5×4 的二维矩阵，找出其中最大和最小元素，并指出它们所在的行和列。若存在多个最大或者最小元素，则只返回第一个所在的行和列。先按行看，再按列看。

```
#include <stdio.h>
int main()
{
    int a[5][4];
    printf("Input elements for a 5*4 matrix:\n");

    int (*p)[4] = &a[0]; // 注意到p的类型为int (*)[4]
    while(p < a + 5)
```

```
    {
        printf("Input the %d-th row:\n", p - a + 1);
        for(int j = 0; j < 4; j++)
        {
            scanf("%d", &(*p)[j]);
        }
        p++;
    }

    printf("The input matrix is:\n");
    p = &a[0];
    while(p < a + 5)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%d\t", (*p)[j]);
        }
        printf("\n");
        p++;
    }

    // 先把最大和最小值的行标和列标初始化为0
    int row_for_maxi = 0, column_for_maxi = 0;
    int row_for_mini = 0, column_for_mini = 0;

    p = &a[0];
    for(int i = 0; i < 5; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            if(*(p + i)[j] > *(a + row_for_maxi)[column_for_maxi]) // 找到更大的
            {
                // 更新行标和列标
                row_for_maxi = i;
                column_for_maxi = j;
            }

            if(*(p + i)[j] < *(a + row_for_mini)[column_for_mini]) // 找到更小的
            {
```

```
48                  // 更新行标和列标
49                  row_for_mini = i;
50                  column_for_mini = j;
51              }
52
53              p++;
54          }
55      }
56
57      printf("The first maxi is at (%d, %d)\n", row_for_maxi, column_for_maxi);
58      printf("The first mini is at (%d, %d)\n", row_for_mini, column_for_mini);
59
60      return 0;
61  }
```

5. 求 M 行 N 列矩阵中各行最大值中最小的数。设 $M = 5$，$N = 4$.

```
1   #include <stdio.h>
2   int main()
3   {
4       int a[5][4]; // 用于存储输入矩阵
5       int maxi_for_each_row[5]; // 用于存储各行的最大值
6
7       printf("Please input elements for a 5*4 matrix:\n");
8
9       int *p = &a[0][0];
10      int *p_out = p + 5 * 4;
11      while(p < p_out) // 按次序枚举每一个元素
12      {
13          if((p - (*a)) % 4 == 0)
14          {
15            printf("Input the %d-th row:\n", (p - (*a)) / 4 + 1);
16          }
17          scanf("%d", p);
18          p++;
19      }
20
21      printf("The input matrix is:\n");
22      p = &a[0][0];
```

```
    while(p < p_out) // 按次序枚举每一个元素
    {
        printf("%d\t", *p);
        p++;
        if((p - (*a)) % 4 == 0)
        {
           printf("\n", (p - (*a)) / 4 + 1);
        }
    }

    // 以下求每一行的最大值
    int (*q)[4] = &a[0];
    while(q < (&a[0]) + 5) // 枚举每一行
    {
        maxi_for_each_row[q - (&a[0])] = (*q)[0]; // 假定该行第一列的元素是最大的
        for(int j = 1; j < 4; j++) // 枚举每一列
        {
            if((*q)[j] > maxi_for_each_row[q - (&a[0])]) // 找到更大的
            {
                maxi_for_each_row[q - (&a[0])] = (*q)[j]; // 更新最大值
            }
        }
        q++;
    }

    // 先假定第一个元素最小
    int min_max = maxi_for_each_row[0];
    for(int i = 1; i < 5; i++)
    {
        if(min_max > maxi_for_each_row[i]) // 找到更小
        {
            min_max = maxi_for_each_row[i]; // 更新最小
        }
    }

    printf("The minimum value of the maximum in each row is: %d\n", min_max);
}
```

6. 输入两个英文单词，输出它们在字典中的先后。

```c
#include <stdio.h>
#include <string.h>
const int MAX_LEN = 1024; // 字符串空间的大小，须确保足够大
int main()
/*用户输入两个英文单词，程序返回它们在字典中的先后*/
{
   char s1[MAX_LEN], s2[MAX_LEN]; // 用于存储待比较的两个字符串
   printf("Input two string:\n");
   gets(s1);
   gets(s2);
   char *p1 = s1, *p2 = s2; // 用于定位需要进行比较的字符
   // 对两个单词进行扫描
   while(p1 && p2) // 当两个单词均为被扫描至末端
   {
      if((*p1) < (*p2)) // 字符先则单词先
      {
         printf("%s comes before %s\n", s1, s2);
         return 0;
      }
      else if((*p1) > (*p2)) // 字符后则单词后
      {
         printf("%s comes before %s\n", s2, s1);
         return 0;
      }
      else // 字符相同，因此去比较各自的下一个字母
      {
         p1++; // 准备好下一个字符的位置
         p2++; // 同上
      }
   }

   // 至少有一个单词被扫描至末端

   if(p1) // 前一个输入的单词未被扫描至末端
   {
      printf("%s comes before %s\n", s2, s1);
      return 0;
   }

```

```
40      if(p2) // 后一个输入的单词未被扫描至末端
41      {
42          printf("%s comes before %s\n", s1, s2);
43          return 0;
44      }
45
46      // 两单词都扫描至末端
47      printf("%s and %s are equal\n", s1, s2);
48
49      return 0;
50  }
```