动态分配存储空间

2022 年 8 月 16 日

1. 首先输入一个正整数 n，然后输入包含 n 个正整数的数列，要求先按输入的顺序输出该数列，然后把该数列按逆序输出（要求所用内存尽量少）。

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int len;
    printf("Input the len of an input integer sequence:\n");
    scanf("%d", &len);
    // malloc()在内存中分配连续的len个空间，每个空间的大小都是一个int型变量的大小，
    // 并返回这片空间的最低低字节的地址
    // 左边是int*型，右边也是int*型，左右同型，可以赋值
    int *p = (int*) malloc(len * sizeof(int));
    if(!p) // 分配失败
    {
        printf("Insufficient memory");
        exit(-1);
    }
    for(int i = 0; i < len; i++)
    {
        printf("Input the %d-th integer:\n", i + 1);
        scanf("%d", p + i); // 键盘输入，然后存入p+i指向的空间
    }
    printf("the input sequence is:\n");
    for(int i = 0; i < len; i++)
    {
        printf("%d\t", p[i]);
    }
```

```
    puts("\n");
    printf("the reversed sequence is:\n");
    for(int i = len - 1; i >= 0; i--)
    {
        printf("%d\t", p[i]);
    }
    puts("\n");
    free(p); // 回收空间
    return 0;
}
```

2. 输入两个英文单词，输出它们在字典中的先后。

```
#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);
   int i = 0, j = 0; // 用于定位需要进行比较的字符
   // 对两个单词进行扫描
   while(i < strlen(s1) - 1 && j < strlen(s2) - 1) // 当两个单词均为被扫描至末端
   {
      if(s1[i] < s2[j]) // 字符先则单词先
      {
         printf("%s comes before %s\n", s1, s2);
         return 0;
      }
      else if(s1[i] > s2[j]) // 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
      {
```

```
27          i++; // 准备好下一个字符的位置
28          j++; // 同上
29        }
30    }
31
32    // 至少有一个单词被扫描至末端
33
34    if(i != strlen(s1) - 1) // 前一个输入的单词未被扫描至末端
35    {
36      printf("%s comes before %s\n", s2, s1);
37      return 0;
38    }
39
40    if(j != strlen(s2) - 1) // 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  }
```

3. 输入两个正整数 m 和 n，然后输入一个 m 行 n 列的矩阵 A，程序输出矩阵 A 的转置(要求所用内存尽量少)。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      int m, n;
6      printf("Input the row and column number for a matrix:\n");
7      scanf("%d %d", &m, &n);
8
9      // malloc()在内存中分配连续的m个空间，每个空间的大小是int*型变量的大小，
10     // 并返回这片空间最低字节的地址
11     // 左边是int**型，右边也是int**型，左右同型，可以赋值
```

```
12      int **p = (int **) malloc(m * sizeof(int*));
13      if(!p) // 空间分配失败
14      {
15          printf("Insufficient memory");
16          exit(-1);
17      }
18      for(int i = 0; i < m; i++)
19      {
20          // malloc()在内存中分配连续的n个空间，每个空间的大小是int型变量的大小，
21          // 并返回这片空间最低字节的地址
22          // 注意到p是int**型，因此，p偏移i个元素所指向的对象是int*型，左右同型，可以赋值
23          p[i] = (int*) malloc(n * sizeof(int));
24          if(!p[i])
25          {
26              printf("Insufficient memory");
27              exit(-1);
28          }
29      }
30
31      for(int i = 0; i < m; i++)
32      {
33          printf("Input the %d-th row:\n", i + 1);
34          for(int j = 0; j < n; j++)
35          {
36              printf("Input the %d-th column:\n", j + 1);
37              scanf("%d", &p[i][j]);
38          }
39      }
40
41      printf("inputs:\n");
42      for(int i = 0; i < m; i++)
43      {
44          for(int j = 0; j < n; j++)
45          {
46              printf("%d\t", p[i][j]);
47          }
48          putchar('\n');
49      }
50
```

```
51      // 以下分配空间，用于存储n*m的矩阵
52      int **q = (int **) malloc(n * sizeof(int*));
53      if(!q)
54      {
55          printf("Insufficient memory");
56          exit(-1);
57      }
58      for(int i = 0; i < n; i++)
59      {
60          q[i] = (int*) malloc(m * sizeof(int));
61          if(!q[i])
62          {
63              printf("Insufficient memory");
64              exit(-1);
65          }
66      }
67
68      for(int i = 0; i < n; i++)
69      {
70          for(int j = 0; j < m; j++)
71          {
72              q[i][j] = p[j][i];
73          }
74      }
75
76      printf("transposed:\n");
77      for(int i = 0; i < n; i++)
78      {
79          for(int j = 0; j < m; j++)
80          {
81              printf("%d\t", q[i][j]);
82          }
83          putchar('\n');
84      }
85
86      for(int i = 0; i < m; i++)
87      {
88          free(p[i]);
89      }
```

```
90      free(p);
91      for(int i = 0; i < n; i++)
92      {
93          free(q[i]);
94      }
95      free(q);
96      return 0;
97  }
```