带参数的构造函数

范 懿

目录

```cpp
#include <iostream>
using namespace std;
class MyClass {
        int a, b;
    public:
        MyClass(int i, int j) {a = i; b = j;}
        void show() {cout << a << " " << b;}
};
int main(){
    MyClass ob(3, 5);
    ob.show();
    return 0;
}
```

- 传入参数 i 和 j 用来给参数 a 和 b 赋值。
- MyClass ob(3, 5); 创建一个对象 ob 并把 3 和 5 赋给 MyClass() 的参数 i 和 j
- 也可以这样传参数：MyClass ob = MyClass(3, 5);
- 两种写法有细微区别：详见拷贝构造函数

```
#include <iostream>
#include <cstring>
using namespace std;
const int IN = 1;
const int CHECKED_OUT = 0;
class Book {
        char author[40];
        char title[40];
        int status; // 是否在馆
    public:
        Book(char *n, char *t, int s);
        int get_status() {return status;}
        void set_status(int s) {status = s;}
        void show();
};


Book::Book(char *n, char *t, int s)
{
    strcpy(author, n);
    strcpy(title, t);
```

```
22      status = s;
23  }
24  void Book::show()
25  {
26      cout << title << " by " << author;
27      cout << " is ";
28      if(status==IN) cout << "in.\n";
29      else cout << "out.\n";
30  }
31  int main()
32  {
33      Book b1("Twain", "Tom Sawyer", IN);
34      Book b2("Melville", "Moby Dick", CHECKED_OUT);
35      b1.show();
36      b2.show();
37      return 0;
38  }
```

带参数的构造函数直接可以完成全部初始化

```
1  #ifndef MY_STRING_H
2  #define MY_STRING_H
3  class MyString{
4      private:
5          char *str; // 字符串首元素地址
6          int len; // 字符串长度
7          int capacity; // 字符串容量
8          // 获取字符串首元素地址
9          // 涉及内存，设为私有比较保险
10         char* get_str()
11         {
12             return str;
13         }
14     public:
15         MyString();
16         MyString(const char* s);
17         ~MyString();
18         // 把MyString类型对象s赋值给
19         // 当前MyString类型对象
20         void assign(MyString &s);
21         // 把对象s连接到当前的对象之后
```

```
22          void append(MyString &s);
23          void show();
24  };
25  #endif
```

```cpp
#include <iostream>
#include <cstring>
#include "MyString.h"

using namespace std;

// 无参构造函数
MyString::MyString()
{
    str = NULL;
    len = 0;
    capacity = 0;
}

//带参构造函数
MyString::MyString(const char* s)
{
    len = strlen(s); // 确定长度
    capacity = len; // 确定容量
    str = new char[capacity + 1]; // 分配空间
    strcpy(str, s); // 复制
```

```
22 }
23
24 // 析构函数
25 MyString::~MyString()
26 {
27     delete []str; // 释放内存
28 }
29
30 // 把MyString型对象s赋值给调用本成员函数的对象
31 void MyString::assign(MyString &s)
32 {
33     len = strlen(s.get_str());
34     if(capacity < len) // 容量不足
35     {
36         capacity = len;
37         delete[] str; // 清理旧空间
38         str = new char[capacity + 1]; // 空间扩充
39         if(!str) // 注意检查是否分配成功
40         {
41             cout << "insufficient memory" << endl;
42             exit(1);
```

```
43          }
44      }
45      strcpy(str, s.get_str()); // 复制
46  }
47
48  // 把MyString型对象添加到调用本成员函数的对象的后面
49  void MyString::append(MyString &s)
50  {
51      len += strlen(s.get_str()); // 扩充长度
52      if(capacity < len){
53          capacity = len; // 扩充容量
54          // 扩充空间
55          char* temp_str = new char[len + 1];
56          if(!temp_str) // 注意检查是否分配成功
57          {
58              cout << "insufficient memory" << endl;
59              exit(1);
60          }
61          // 把原字符串复制过去
62          strcpy(temp_str, str);
63          delete[] str; // 回收原字符串占据的空间
```

```
64              str = temp_str; // 指针重定向
65          }
66          strcat(str, s.get_str()); // 连接
67      }
68
69      void MyString::show()
70      {
71          cout << str;
72      }
```

```cpp
#include <iostream>
#include "MyString.h"

using namespace std;

int main()
{
    MyString str1("long"), str2("-term");
    str1.show();
    cout << endl;
    str2.show();
    cout << endl;
    str1.append(str2);
    str1.show();
    cout << endl;
    str1.assign(str2);
    str1.show();
    cout << endl;
    // 析构函数自动释放内存
    return 0;
}
```

```
1  #ifndef MY_VECTOR_H
2  #define MY_VECTOR_H
3  #include <iostream>
4
5  using namespace std;
6
7  const int OK = 1;
8  const int ERROR = 0;
9  const int INIT_CAPACITY = 4;
10
11 class MyVector{
12     private:
13         int *arr; // 数组首元素地址
14         int array_size; // 数组大小
15         int capacity; // 数组容量
16
17     public:
18         MyVector();
19         MyVector(int *a, int sz);
20         ~MyVector();
21
```

```
22          /*返回下标为i的元素（即第i+1个元素）的值*/
23          int get_element(int i, int &e);
24
25          /*返回最后一个值为e的元素的下标；若满足条件的元素不存在，则返回-1，*/
26          int locate_element(int e);
27
28          /*在下标为i的位置之前插入元素e，i的范围是[0, array_size]*/
29          int insert_element(int i, int e);
30
31          /*删除下标为i的元素*/
32          int delete_element(int i);
33
34          /*输出迭代器的内容*/
35          void show();
36  };
37
38  #endif
```

```
1  #include <iostream>
2  #include "MyVector.h"
3
4  // 无参构造函数
5  MyVector::MyVector()
6  {
7      capacity = INIT_CAPACITY;
8      arr = new int[capacity];
9      if(!arr)
10     {
11         cout << "insufficient memory" << endl;
12         exit(1);
13     }
14     array_size = 0;
15 }
16
17
18
19
20
21
```

```
22
23
24 // 带参构造函数
25 MyVector::MyVector(int *a, int sz)
26 {
27     // 确定容量
28     capacity = INIT_CAPACITY>sz?INIT_CAPACITY:sz;
29     // 分配空间
30     arr = new int[capacity];
31     if(!arr)
32     {
33         cout << "insufficient memory" << endl;
34         exit(1);
35     }
36     // 复制
37     for(int i = 0; i < sz; i++)
38     {
39         arr[i] = a[i];
40     }
41     array_size = sz; // 维护数组大小
42 }
```

```
43
44
45
46
47 // 析构函数
48 MyVector::~MyVector()
49 {
50     delete[] arr;
51 }
52
53 // 取值
54 int MyVector::get_element(int i, int &e)
55 /*返回下标为i的元素（即第i+1个元素）的值*/
56 {
57     if(i < 0 || i >= array_size) // 访问越界
58     {
59         return ERROR;
60     }
61     e = arr[i];
62     return OK;
63 }
```

```
64
65
66
67
68 // 定位
69 int MyVector::locate_element(int e)
70 /*返回最后一个值为e的元素的下标; 若满足条件的元素不
       存在, 则返回-1, */
71 {
72     int i;
73     // 从后往前搜查
74     for(i = array_size - 1; i >= 0; i--)
75     {
76         if(arr[i] == e) // 找到
77         {
78             break;
79         }
80     }
81     return i;
82 }
83
```

```
84
85
86
87
88
89
90 // 插入
91 int MyVector::insert_element(int i, int e)
92 /*在下标为i的位置之前插入元素e, i的范围是[0,
       array_size], 允许在末端插入元素*/
93 {
94     if(i < 0 || i > array_size)
95     {
96         return ERROR; // 不在合法范围内
97     }
98
99     if(capacity < array_size + 1) // 容量不足
100    {
101        capacity *= 2; // 容量增大为2倍
102        int* new_arr = new int[capacity]; // 分配空
       间
```

```
103             if(!new_arr)
104             {
105                 cout << "insufficient memory" << endl;
106                 exit(1);
107             }
108             int j;
109             // 把前面i个旧元素复制过去
110             for(j = 0; j < i; j++)
111             {
112                 new_arr[j] = arr[j];
113             }
114             new_arr[j++] = e; // 插入元素
115             for(; j < array_size + 1; j++)
116             {
117                 // 复制发生偏移
118                 new_arr[j] = arr[j - 1];
119             }
120             delete[] arr; // 销毁原数组
121             arr = new_arr; // 关联新数组
122         }
123         else
```

```
124     {
125         for(int j = array_size - 1; j >= i; --j)
126         {
127             arr[j + 1] = arr[j]; // 元素后移，包括下标为i的元素
128         }
129         arr[i] = e;
130     }
131     array_size++;
132     return OK;
133 }
134 
135 // 删除
136 int MyVector::delete_element(int i)
137 /*删除下标为i的元素*/
138 {
139     if(i < 0 || i > array_size - 1)
140     {
141         return ERROR;
142     }
143     // 下标从i+1开始是后面的元素，直到最后一个
```

```
144         for(int j = i + 1; j < array_size; j++)
145         {
146             arr[j - 1] = arr[j]; // 前移
147         }
148         array_size--;
149         return OK;
150 }
151
152 // 打印
153 void MyVector::show()
154 {
155         for(int i = 0; i < array_size; i++)
156         {
157             cout << arr[i] << '\t';
158         }
159 }
```

```cpp
#include <iostream>
#include "MyVector.h"

using namespace std;

const int FAILURE = 1;

int main()
{
    // 创建
    int a1[] = {1, 2, 3};
    int a2[] = {5, 4, 3, 2, 1};
    MyVector vec1(a1, 3), vec2(a2, 5);
    cout << "vec1:" << endl;
    vec1.show();
    cout << endl;
    cout << "vec2:" << endl;
    vec2.show();
    cout << endl;

    // 取值
```

```
22      int e;
23      if(!vec1.get_element(2, e))
24      {
25          return FAILURE;
26      }
27      cout << "The 3rd element in vec1 is " << e <<
        endl;
28      if(!vec2.get_element(3, e))
29      {
30          return FAILURE;
31      }
32      cout << "The 4th element in vec2 is " << e <<
        endl;
33
34      // 定位
35      int index = vec1.locate_element(4);
36      if(index == -1)
37      {
38          cout << "4 is not found in vec1" << endl;
39      }
40      else
```

```
41      {
42          cout << "the index of 4 in vec1 is " <<
        index << endl;
43      }
44
45      index = vec2.locate_element(4);
46      if(index == -1)
47      {
48          cout << "4 is not found in vec2" << endl;
49      }
50      else
51      {
52          cout << "the index of 4 in vec2 is " <<
        index << endl;
53      }
54
55      // 插入
56      cout << "try to insert 8 into vec1 before
        location 4" << endl;
57      if(!vec1.insert_element(4, 8))
58      {
```

```
59          cout << "inserting 8 into vec1 before
    location 4 causes failures" << endl;
60      }
61      cout << "vec1:" << endl;
62      vec1.show();
63      cout << endl;
64
65      cout << "try to insert 8 into vec1 before
    location 2" << endl;
66      if(!vec1.insert_element(2, 8))
67      {
68          cout << "inserting 8 into vec1 before
    location 2 causes failures" << endl;
69      }
70      cout << "vec1:" << endl;
71      vec1.show();
72      cout << endl;
73
74      cout << "try to insert 7 into vec2 before
    location 5" << endl;
75      if(!vec2.insert_element(5, 7))
```

```
76      {
77          cout << "inserting 7 into vec2 before
        location 5 causes failures" << endl;
78      }
79      cout << "vec2:" << endl;
80      vec2.show();
81      cout << endl;
82
83      cout << "try to delete the element at location
        1 in vec1" << endl;
84      if(!vec1.delete_element(1))
85      {
86          cout << "deleting the element at location 1
         in vec1 failed" << endl;
87      }
88      cout << "vec1:" << endl;
89      vec1.show();
90      cout << endl;
91
92      cout << "try to delete the element at location
        5 in vec2" << endl;
```

```
93      if(!vec2.delete_element(5))
94      {
95          cout << "deleting the element at location 5 in vec2 failed" << endl;
96      }
97      cout << "vec1:" << endl;
98      vec2.show();
99      cout << endl;
100
101     // 无需手工调用析构函数
102     return 0;
103 } /*g++ .\testMyVector.cpp .\MyVector.cpp -o .\test
      */
```

```
1  #ifndef MY_QUEUE_H
2  #define MY_QUEUE_H
3
4  const int INIT_SIZE = 4;
5  const int OK = 1;
6  const int ERROR = 0;
7
```

```
8  class MyQueue{
9  /*用长度为n+1的数组来实现长度不超过n的动态队列，
10 当空间不足时，动态扩充*/
11     private:
12         int *arr; // 数组首元素地址
13         int array_capacity; // 数组容量
14         int head; // 队首下标
15         int tail; // 队尾的下一个元素的下标
16         /*当head==tail时，队列为空；当tail+1==head
       时，队列为满。*/
17
18     public:
19         MyQueue();
20
21         // 根据数组构造队列
22         MyQueue(int *a, int n);
23         ~MyQueue();
24
25         // 入队，空间不足时扩充
26         void enQueue(int x);
27         // 出队，返回出队的元素的值
```

```
28          int deQueue();
29          void show();
30  };
31
32  #endif
```

```cpp
#include <iostream>
#include <stdio.h>
#include "MyQueue.h"

using namespace std;

// 无参构造函数
MyQueue::MyQueue()
{
    array_capacity = INIT_SIZE; // 确定容量
    arr = new int[array_capacity + 1]; // 分配空间
    if(!arr)
    {
        cout << "insufficient memory" << endl;
        exit(1);
    }
    head = tail = 0; // 空队列的头尾指针相等
}


```

```
22
23
24 // 带参构造函数
25 MyQueue::MyQueue(int* a, int n)
26 {
27     // 确定容量
28     array_capacity = INIT_SIZE > n ? INIT_SIZE : n;
29     // 分配空间
30     arr = new int[array_capacity + 1];
31     if(!arr)
32     {
33         cout << "insufficient memory" << endl;
34         exit(1);
35     }
36     // 空队列的头尾指针相等
37     head = 0;
38     for(int i = 0; i < n; i++)
39     {
40         arr[i] = a[i]; // 填入
41     }
42     tail = n; // 最后一个元素的下标加1
```

```
43 }
44
45
46
47 // 析构函数
48 MyQueue::~MyQueue()
49 {
50     delete[] arr;
51 }
52
53 // 打印
54 void MyQueue::show()
55 {
56     for(int i = head; i != tail;)
57     {
58         cout << arr[i] << '\t';
59         // 注意数组长度为capacity+1
60         if(i == array_capacity)
61         {
62             i = 0; // 自增后回到数组首元素位置
63         }
```

```
64              else
65              {
66                  i++;
67              }
68          }
69  }
70
71  // 入队
72  void MyQueue::enQueue(int x)
73  {
74      if(head == (tail + 1) % (array_capacity + 1))
        //满队
75      {
76          // 多一倍空间，肯定不需要折回，可按通常数组
        处理
77          int* new_arr = new int[array_capacity * 2 +
         1];
78          int j = head;
79          /*i用于扫描原数组，j用于扫描新数组*/
80          for(int i = head; i != tail; j++)
81          /*head下标保持不变，因此i和j起点相同*/
```

```
82          {
83              new_arr[j] = arr[i];
84              if(i == array_capacity) // 原数组已扫到尽头
85              {
86                  i = 0; // 折回
87              }
88              else
89              {
90                  i++;
91              }
92          } // 循环结束时，j为有效元素的下一个位置的下标
93          delete[] arr; // 销毁原数组
94          arr = new_arr; // 关联新数组
95          tail = j; // 维护队尾下一个位置的下标
96          array_capacity *= 2;
97      } // 循环结束，不满队
98      arr[tail] = x;
99      if(tail == array_capacity)
100     {
```

```
101             tail = 0;
102         }
103         else
104         {
105             tail++;
106         }
107 }
108
109
110
111
112
113
114
115
116 //  出队
117 int MyQueue::deQueue()
118 {
119     if(tail == head)
120     {
121         cout << "underflow" << endl;
```

```
122             return ERROR;
123         }
124         int x = arr[head]; // 获取队首元素
125         if(head == array_capacity)
126         {
127             head = 0; // 折回
128         }
129         else
130         {
131             head++;
132         }
133         return x;
134     }
```

```
#include <iostream>

#include "MyQueue.h"

using namespace std;

int main()
{
    int a1[] = {2, 3, 5, 7};
    int a2[] = {11, 13, 17, 19};
    MyQueue que1(a1, 4), que2(a2, 4);

    cout << "que1:" << endl;
    que1.show();
    cout << endl;

    cout << "que2:" << endl;
    que2.show();
    cout << endl;

    int x = 8;
```

```
22      que1.enQueue(x); // que1.enQueue(4);
23      cout << "appended " << x << " into que1" <<
    endl;
24      cout << "que1:" << endl;
25      que1.show();
26      cout << endl;
27
28      x = 6;
29      que2.enQueue(x);
30      cout << "appended " << x << " into que2" <<
    endl;
31      x = 4;
32      que2.enQueue(x);
33      cout << "appended " << x << " into que2" <<
    endl;
34      cout << "que2:" << endl;
35      que2.show();
36      cout << endl;
37
38      x = que1.deQueue();
39      cout << "obtained " << x << " from que1" <<
```

```
    endl;
40   cout << "que1:" << endl;
41   que1.show();
42   cout << endl;
43
44   x = que2.deQueue();
45   cout << "obtained " << x << " from que2" <<
    endl;
46   x = que2.deQueue();
47   cout << "obtained " << x << " from que2" <<
    endl;
48   cout << "que2:" << endl;
49   que2.show();
50   cout << endl;
51
52   x = 3;
53   que1.enQueue(x);
54   cout << "appended " << x << " into que1" <<
    endl;
55   cout << "que1:" << endl;
56   que1.show();
```

```
57      cout << endl;
58
59      x = 4;
60      que2.enQueue(x);
61      cout << "appended " << x << " into que2" <<
        endl;
62      cout << "que2:" << endl;
63      que2.show();
64      cout << endl;
65
66      return 0;
67  }
```