题目描述

在本题中，你需要实现一个**智能消息队列系统**，该系统用于管理多条消息，每条消息包含内容和优先级。消息的优先级会影响系统处理的顺序，因此需要按优先级顺序插入新消息。同时，系统会频繁地在不同队列管理器之间移动消息，因此**移动构造函数**的使用非常重要。

你的任务是实现一个消息队列管理器 `MessageQueueManager`，它可以动态添加并管理 `Message` 对象，并能够高效地转移消息资源。你需要将框架代码中的 `MessageQueueManager.h` 和 `MessageQueueManager.cpp` 复制到CPPOJ中，`Main.cpp` 将在自测和提交测试时自动替换成框架代码。

你需要完成 `MessageQueueManager.cpp` 中所有标注 `TODO` 的部分，本题不需要你来实现输入输出功能。下面将具体讲述你要完成的任务：

Task 1 实现Message类（50%）

`Message` 类用来表示消息，每个消息包含一个指针 `char* data`（表示消息内容）和一个 `int priority`（表示消息优先级，范围 1 到 100）。

- 你需要实现 `Message` 类的**构造函数**，注意在构造 `Message` 对象时，内容 `data` 需要动态分配内存。（**10分**）
- 你需要实现 `Message` 的**移动构造函数**和**移动赋值运算符**，在不拷贝 `data` 数据的情况下高效转移 `Message` 对象的所有权。（**40分**）
- 析构函数已为你实现好，在对象销毁时会释放 `data` 指针的内存。

> 提示：在Task1和Task3中，请考虑使用初始化成员列表

Task 2 MessageQueueManager类中添加消息 （20%）

`MessageQueueManager` 使用一个 `std::vector` 来存储消息并提供一个 `addMessage` 方法，用于按优先级顺序将消息插入到适当位置，确保消息队列始终保持优先级排序（按 `priority` 从小到大排序，题目保证所有消息的优先级各不相同）。

- 你需要实现 `addMessage` 方法，往 `MessageQueueManager` 添加一个消息，并保证消息队列始终保持优先级排序。每次添加使用O(n)开销的时间即可。（**20分**）

  提示：可以使用 `std::vector` 的 `insert` 方法，为此你需要获得一个位置的迭代器（二分或顺序查找）。

  注意：在本题中，`Message` 的**拷贝构造函数被禁用**，你需要考虑使用 `std::move()` 来实现移动。

Task 3 MessageQueueManager类的移动（20%）

- 实现 `MessageQueueManager` 的**移动构造函数**和**移动赋值运算符**，高效地转移其他 `MessageQueueManager` 对象的消息，避免不必要的深拷贝。（**20分**）

本题还有一个隐藏测试用例，正确完成Task1~Task3并通过该用例，你可以获得剩余的**10分**。

注意：不要删除框架代码中的 `std::cout` 语句或移动它们的顺序，会影响程序正确性的判断。测试代码中不会出现修改消息队列中消息的优先级。

框架代码

MessageQueueManager.h

```
#ifndef MESSAGE_QUEUE_MANAGER_H
#define MESSAGE_QUEUE_MANAGER_H

#include <vector>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>

// Message 类的声明
class Message {
public:
      char* data;
      int priority;

      Message(const char* d, int p);
      Message(const Message& other) = delete;
      Message& operator=(const Message& other) = delete;
      Message(Message&& other) noexcept;
      Message& operator=(Message&& other) noexcept;
      ~Message();
};

// MessageQueueManager 类的声明
class MessageQueueManager {
public:
      std::vector<Message> messages;

      // 添加消息并按优先级顺序插入
      void addMessage(Message m);
      MessageQueueManager();
      MessageQueueManager(MessageQueueManager& other) = delete;
      MessageQueueManager& operator=(MessageQueueManager& other) = delete;
      MessageQueueManager(MessageQueueManager&& other) noexcept;
      MessageQueueManager& operator=(MessageQueueManager&& other) noexcept;

      // 打印消息队列
      void printMessages() const;

      ~MessageQueueManager();
};

#endif // MESSAGE_QUEUE_MANAGER_H
```

MessageQueueManager.cpp

```
#include "MessageQueueManager.h"

// Message 类的实现

Message::Message(const char *d, int p) {
   // TODO: Task 1-1
}

Message::Message(Message &&other) noexcept {
   // TODO: Task 1-2
}
```

```
Message &Message::operator=(Message &&other) noexcept {
    // TODO: Task 1-2
    return *this;
}

Message::~Message() {
    delete[] data;
}

// MessageQueueManager 类的实现
MessageQueueManager::MessageQueueManager() {}

void MessageQueueManager::addMessage(Message m) {
    // TODO: Task 2
}

MessageQueueManager::MessageQueueManager(MessageQueueManager &&other) noexcept {
    // TODO: Task 3-1
}

MessageQueueManager &
MessageQueueManager::operator=(MessageQueueManager &&other) noexcept {
    // TODO: Task 3-2
    return *this;
}

void MessageQueueManager::printMessages() const {
    for (const auto &msg : messages) {
        std::cout << "Message: " << msg.data << ", Priority: " << msg.priority
                  << std::endl;
    }
}

MessageQueueManager::~MessageQueueManager() {
    std::cout << "Destructing MessageQueueManager" << std::endl;
}
```

测试代码（仅含9个公开用例）

Main.cpp

```
#include "MessageQueueManager.h"
#include <cassert>
#include <cstdlib>
#include <functional>

// === TEST_CASES ===
void TEST_1();
void TEST_2();
void TEST_3();
void TEST_4();
void TEST_5();
void TEST_6();
void TEST_7();
void TEST_8();
void TEST_9();
```

```
#define REGISTER_TEST_CASE(name) {#name, name}

int main() {
   std::unordered_map<std::string, std::function<void()>>
         test_functions_by_name = {
               REGISTER_TEST_CASE(TEST_1), REGISTER_TEST_CASE(TEST_2),
               REGISTER_TEST_CASE(TEST_3), REGISTER_TEST_CASE(TEST_4),
               REGISTER_TEST_CASE(TEST_5), REGISTER_TEST_CASE(TEST_6),
               REGISTER_TEST_CASE(TEST_7), REGISTER_TEST_CASE(TEST_8),
               REGISTER_TEST_CASE(TEST_9),
         };

   std::string test_case_name;
   std::cin >> test_case_name;
   auto it = test_functions_by_name.find(test_case_name);
   assert(it != test_functions_by_name.end());
   auto fn = it->second;
   fn();
   return 0;
}

void TEST_1() {
   Message m1 = {"hello, world!", 10};
   Message m2("I love NJU", 1);
   Message *m3 = new Message("I", 100);
   std::cout << m1.data << " " << m1.priority << std::endl;
   std::cout << m2.data << " " << m2.priority << std::endl;
   std::cout << m3->data << " " << m3->priority << std::endl;
   delete m3;
   m1.data[0] = 'd';
   m1.data[1] = 'e';
   std::cout << m1.data << " " << m1.priority << std::endl;
   const char *SCHOOL = "NJU";
   Message m4{SCHOOL, 50};
   std::cout << m4.data << " " << m4.priority << std::endl;
   m4.data[0] = 'Z';
   m4.priority = 60;
   std::cout << m4.data << " " << m4.priority << std::endl;
}

void TEST_2() {
   Message m1 = {"hello, world!", 10};
   char *pre_ptr = m1.data;
   Message m11 = std::move(m1);
   std::cout << (m11.data == pre_ptr) << std::endl;
   std::cout << m11.data << " " << m11.priority << std::endl;
   Message m2(std::move(m11));
   std::cout << (m2.data == pre_ptr) << std::endl;
   std::cout << m2.data << " " << m2.priority << std::endl;
}

void TEST_3() {
   Message m1 = {"hello, world!", 10};
   char *pre_ptr = m1.data;
   Message m11 = std::move(m1);
   m11.data[0] = 'H';
   std::cout << (m11.data == pre_ptr) << std::endl;
```

```
    std::cout << m11.data << " " << m11.priority << std::endl;
    Message m2 = {"nihao", 20};
    pre_ptr = m2.data;
    Message m21(std::move(m2));
    m21.data[0] = 'N';
    std::cout << (m21.data == pre_ptr) << std::endl;
    std::cout << m21.data << " " << m21.priority << std::endl;
}

void TEST_4() {
    Message m1 = {"hello, world!", 10};
    Message m2 = {"nihao, world", 20};
    std::cout << m1.data << " " << m1.priority << std::endl;
    char *pre_ptr_1 = m1.data;
    char *pre_ptr_2 = m2.data;
    m1 = std::move(m2);
    std::cout << m1.data << " " << m1.priority << std::endl;
    std::cout << (m1.data == pre_ptr_1) << std::endl;
    std::cout << (m1.data == pre_ptr_2) << std::endl;
    Message m3 = std::move(m1);
    std::cout << m3.data << " " << m3.priority << std::endl;
    std::cout << (m3.data == pre_ptr_1) << std::endl;
    std::cout << (m3.data == pre_ptr_2) << std::endl;
}

void TEST_5() {
    Message m1 = {"hello, world!", 10};
    char *pre_ptr = m1.data;
    Message m11 = std::move(m1);
    m11.data[0] = 'H';
    m11.priority = 15;
    std::cout << (m11.data == pre_ptr) << std::endl;
    std::cout << m11.data << " " << m11.priority << std::endl;
    Message m2 = {"nihao", 20};
    m2 = std::move(m11);
    m2.data[1] = 'E';
    ++m2.priority;
    std::cout << m2.data << " " << m2.priority << std::endl;
    m2 = std::move(m2);
    std::cout << m2.data << " " << m2.priority << std::endl;
}

void TEST_6() {
    MessageQueueManager manager;
    int priorities[] = {0, 45, 57, 34, 100, 78};
    for (int i = 1; i <= 5; ++i) {
        int priority = priorities[i];
        manager.addMessage(
              Message(("Message " + std::to_string(i)).c_str(), priority));
   }
    manager.printMessages();
}

void TEST_7() {
    MessageQueueManager manager;
    int priorities[] = {0, 45, 57, 34, 100, 78, 12, 35, 56, 99, 54, 1};
    for (int i = 1; i <= 11; ++i) {
        int priority = priorities[i];
```

```
        manager.addMessage(
                Message(("Message " + std::to_string(i)).c_str(), priority));
    }
    manager.printMessages();
    manager.addMessage({"hello, world", 15});
    manager.addMessage({"I love NJU", 80});
    manager.printMessages();
    MessageQueueManager manager2;
    int priorities2[] = {0, 5, 4, 3, 2, 1};
    for (int i = 1; i <= 5; ++i) {
        int priority = priorities[i];
        manager2.addMessage(
                Message(("Message " + std::to_string(i)).c_str(), priority));
    }
    manager2.printMessages();
}

void TEST_8() {
    MessageQueueManager manager;
    int priorities[] = {0, 45, 57, 34, 100, 78};
    for (int i = 1; i <= 5; ++i) {
        int priority = priorities[i];
        manager.addMessage(
                Message(("Message " + std::to_string(i)).c_str(), priority));
    }

    std::cout << "\nOriginal MessageQueueManager:" << std::endl;
    manager.printMessages();

    std::cout << "\nMoving manager to anotherManager:\n";
    MessageQueueManager anotherManager = std::move(manager);

    std::cout << "\nMessages in anotherManager:" << std::endl;
    anotherManager.printMessages();

    std::cout << "\nExiting Test:\n";
}

void TEST_9() {
    MessageQueueManager manager;
    int priorities[] = {0, 58, 57, 34, 98, 28};
    for (int i = 1; i <= 5; ++i) {
        int priority = priorities[i];
        manager.addMessage(
                Message(("Message " + std::to_string(i)).c_str(), priority));
    }

    std::cout << "\nOriginal MessageQueueManager:" << std::endl;
    manager.printMessages();

    std::cout << "\nMoving manager to anotherManager:\n";
    MessageQueueManager anotherManager;
    anotherManager.addMessage({"I love NJU!", 20});
    anotherManager = std::move(manager);

    std::cout << "\nMessages in anotherManager:" << std::endl;
    anotherManager.printMessages();
    anotherManager = std::move(anotherManager);
```

```
    anotherManager.printMessages();
    std::cout << "\nExiting Test:\n";
}
```

注：你可以在标准输入中输入 `TEST_1` 到 `TEST_9` 进行自测。