

电子科技大学

作业报告

学生姓名：冯文森 学号：201522010534 指导教师：李林

学生 E-mail：1071881690@qq.com

一、作业名称

编写自定义消息队列

二、作业要求

- 1.该消息队列不能依赖于已有库的队列，必须是自己设计实现的；
2. 该消息队列是线程安全的；
3. 该消息队列能接收 CLMessage 继承体系的消息；
4. 该消息队列能融入到本课程所提供的程序库中，能支持线程之间的通信（即需要提供发送消息的通信类，以及消息循环的管理类）

三、设计与实现

由作业要求中条件 1，可实现自己的泛型链表队列，由于泛型的特性，故满足条件 3 和 4，因为原继承体系是线程安全的，故只需将队列融入原继承体系即可，满足了条件 2。

代码清单 3.1 队列的实现头文件 CLQueue.h

```
#ifndef _CLQUEUE_H
#define _CLQUEUE_H
// 链表节点
template<class T>
class CLNode {

public:
    T item;
    class CLNode* next;
};

template<class T>
```

```

class CLQueue {

public:
    CLQueue();
    ~CLQueue();
    void QPush(T item); //入队
    T QPop(); //出队
    bool isQEmpty(); //判断队列是否为空

private:
    class CLNode<T> *QHead; //头节点
    class CLNode<T> *QTail; //尾节点
};
#endif

```

代码清单 3.1 为队列实现的头文件

代码清单 3.2 CLQueue.cpp 队列实现

```

#include<iostream>
#include"CLQueue.h"
using namespace std;
// 构造函数
template<typename T>
CLQueue<T>::CLQueue() {
    QHead = NULL;
    QTail = NULL;
}

// 析构函数，将队列清空
template<typename T>
CLQueue<T>::~~CLQueue() {

    while(QHead) {
        class CLNode<T> *tmp = QHead;
        QHead = QHead->next;
        delete tmp;
    }
}

// 入队函数，如果队列为空，则创建头结点，同时将尾节点也指向头结点
// 如果队列不为空，则将新节点插入尾节点
template<typename T>
void CLQueue<T>::QPush(T item) {

    if (isQEmpty()) {

```

```

        QHead = new class CLNode<T>;
        QHead->item = item;
        QTail = QHead;
        return;
    }
    class CLNode<T> *nNode = new class CLNode<T>;
    nNode->item = item;
    nNode->next = NULL;
    QTail->next = nNode;
    QTail = nNode;

}
// 出队函数，取得头结点数据，同时释放头结点
template<typename T>
T CLQueue<T>::QPop() {

    if (isQEmpty()) {
        return NULL;
    }

    T item = QHead->item;
    class CLNode<T> *tmp = QHead;
    QHead = QHead->next;
    delete tmp;
    return item;
}

// 判断队列是否为空函数，如果 Qhead 等于 NULL，则返回 true，否则返回
false
template<typename T>
bool CLQueue<T>::isQEmpty() {

    return QHead ? false : true;
}

```

四、测试

代码清单 4.1 test.cpp

```

#include <iostream>
#include "CLThread.h"
#include "CLEXecutiveFunctionProvider.h"
#include "CLMessageQueueByMyqueue.h"

```

```

#include "CLMessage.h"
#include "CLMsgLoopManagerForMyqueue.h"

using namespace std;

#define ADD_MSG 0
#define QUIT_MSG 1

class CLMyMsgProcessor;
// 加法消息，继承至 CLMessage
class CLAddMessage : public CLMessage
{
public:
    friend class CLMyMsgProcessor;

    CLAddMessage(int Op1, int Op2):CLMessage(ADD_MSG)
    {
        m_Op1 = Op1;
        m_Op2 = Op2;
    }

    virtual ~CLAddMessage()
    {
    }

private:
    int m_Op1;
    int m_Op2;
};
// 退出消息
class CLQuitMessage : public CLMessage
{
public:
    CLQuitMessage() : CLMessage(QUIT_MSG)
    {
    }

    virtual ~CLQuitMessage()
    {
    }
};
// 线程执行体
class CLMyMsgProcessor : public CLMsgLoopManagerForMyqueue
{

```

```

    public:
        CLMyMsgProcessor(CLMessageQueueByMyqueue *pQ) :
        CLMsgLoopManagerForMyqueue(pQ)
        {
        }

        virtual ~CLMyMsgProcessor()
        {
        }
        // 分派消息
        virtual CLStatus DispatchMessage(CLMessage *pM)
        {
            CLAddMessage *pAddMsg;
            switch(pM->m_clMsgID)
            {
            case ADD_MSG:
                pAddMsg = (CLAddMessage *)pM;
                cout << pAddMsg->m_Op1 + pAddMsg->m_Op2 << endl;
                break;

            case QUIT_MSG:
                cout << "quit..." << endl;
                return CLStatus(QUIT_MESSAGE_LOOP, 0);

            default:
                break;
            }

            return CLStatus(0, 0);
        }
};

class CLAdder: public CLExecutiveFunctionProvider
{
    CLMessageLoopManager *m_pMsgLoopManager;

public:
    CLAdder(CLMessageLoopManager *pMsgLoopManager)
    {
        m_pMsgLoopManager = pMsgLoopManager;
    }

    virtual ~CLAdder()
    {

```

```

        if(m_pMsgLoopManager != 0)
            delete m_pMsgLoopManager;
    }
    // 执行函数，进入消息队列
    virtual CLStatus RunExecutiveFunction(void* pContext)
    {
        return m_pMsgLoopManager->EnterMessageLoop(pContext);
    }
};

int main()
{
    // 创建消息队列
    CLMessageQueueByMyqueue *pQ = new CLMessageQueueByMyqueue();
    // 创建线程
    CLThread *t = new CLThread(new CLAdder(new CLMyMsgProcessor(pQ)),
true);

    t->Run(0);
    // 发送 3 条加法消息
    CLAddMessage *paddmsg = new CLAddMessage(2, 4);
    pQ->PushMessage(paddmsg);

    CLAddMessage *paddmsg1 = new CLAddMessage(3, 6);
    pQ->PushMessage(paddmsg1);

    CLAddMessage *paddmsg2 = new CLAddMessage(5, 6);
    pQ->PushMessage(paddmsg2);

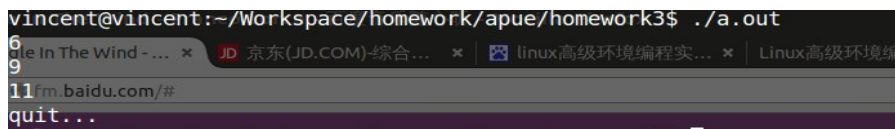
    CLQuitMessage *pquitmsg = new CLQuitMessage();
    pQ->PushMessage(pquitmsg);

    t->WaitForDeath();

    return 0;
}

```

代码清单 4.1 为测试代码，测试结果如下：



```

vincent@vincent:~/Workspace/homework/apue/homework3$ ./a.out
6
9
11
quit...

```

图 4.1 测试结果

五、对本课程或本作业的建议和意见

1. 队列实现中，可增加回收链表的方法，将删除的节点回收到一个链表中，这样可避免重复的内存分配回收。

六、附录

具体代码见 `src/homework3`