

《数据结构》实验报告

姓名： 高心阳 学号： 084623237
班级： 计算机 232 日期： 2024/9/18
程序名： 栈和队列应用实验

一、上机实验的问题和要求：

1、编写一个程序，模拟病人（基本信息有就诊号和姓名，其中就诊号每天从1开始累计）到医院看病，排队看医生的情形。在病人排队过程中，主要发生两件事。

- 1)、病人挂号，到诊室门口候诊。
- 2)、轮到某个号，该号对应的病人进入诊室就诊。

要求程序采用菜单方式，其选项及功能说明如下：

- 1) 挂号候诊——更新待诊病人信息，新挂号的出现在最后。
- 2) 叫号就诊——输出即将就诊的病人（队列中排最前面的病人）信息，该病人进入诊室就诊，更新待诊病人信息，已就诊的病人不再出现。
- 3) 查询——输出前面需要等的人数。
- 4) 退出——退出运行。

2、编写一个程序实现十进制整数转换成其他进制的数。

- 1) 提示用户输入待转换十进制整数和待转换的进制数（2-16）。
- 2) 输出转换后的数。

3、（选做题）在题1中如果每个病人增加一个紧急程度（特别急、一般急、不急）的属性，需要根据病人紧急程度和挂号顺序进行安排，请编程实现。

4、将实验报告以自己的学号和姓名命名，提交到教学平台上。

二、算法设计和基本思想描述：

（分析问题中的数据及数据元素之间的关系，确定选择何种逻辑结构；再根据操作特点分析选择何种存储结构，最后再具体分析每个功能，看需要用到哪些基本操作，如何用基本操作组合出该功能。）

LinkQueue.h

1. **LinkQueue()**: 构造函数，初始化链式队列。
2. **~LinkQueue()**: 析构函数，释放链式队列的内存。
3. **EnQueue(DataType x)**: 向队列末尾添加数据元素。
4. **DeQueue()**: 从队列头部删除并返回数据元素。
5. **GetQueue()**: 获取队列头部的元素但不删除。
6. **Empty()**: 检查队列是否为空。

SeqStack.h

1. **SeqStack()**: 构造函数，初始化顺序栈。
2. **~SeqStack()**: 析构函数，销毁栈（这里未做具体实现）。
3. **Push(DataType x)**: 向栈中添加数据元素。
4. **Pop()**: 弹出并返回栈顶元素。
5. **GetTop()**: 获取栈顶元素但不弹出。
6. **Empty()**: 检查栈是否为空。

LinkPriorityQueue.h

1. **LinkPriorityQueue()**: 构造函数，初始化优先级队列。
2. **LinkPriorityQueue(const LinkPriorityQueue& other)**: 拷贝构造函数，用于深拷贝另一

个优先级队列。

3. **~LinkPriorityQueue()**: 析构函数，释放优先级队列的内存。
4. **Push(DataType x, int priority)**: 向优先级队列中插入带有优先级的数据元素。
5. **Pop()**: 弹出并返回优先级最高的元素。
6. **Get()**: 获取优先级最高的元素，但不弹出。
7. **Priority_Get(int priority)**: 获取指定优先级的元素。
8. **ResetPriority(const DataType x, int priority)**: 重设某元素的优先级。
9. **Empty()**: 检查队列是否为空。

三、源程序及注释（代码直接复制到下面，标好文件名）：

```
LinkQueue.h

#pragma once

template<class DataType>
struct Node {
    DataType data;
    Node<DataType>* next;
};

template<class DataType>
class LinkQueue {
public:
    LinkQueue();
    ~LinkQueue();
    void EnQueue(DataType x);
    DataType DeQueue();
    DataType GetQueue();
    int Empty();
private:
    Node<DataType>* front, * rear;
};

template<class DataType>
LinkQueue<DataType>::LinkQueue() {
    Node<DataType>* s = NULL;
    s = new Node<DataType>;
    s->next = NULL;
    front = rear = s;
}

template<class DataType>
LinkQueue<DataType>::~~LinkQueue() {
    Node<DataType>* p = NULL;
    while (front != NULL) {
```

```

        p = front->next;
        delete front;
        front = p;
    }
}

template<class DataType>
void LinkQueue<DataType>::EnQueue(DataType x) {
    Node<DataType>* p = NULL;
    p = new Node<DataType>;
    p->data = x;
    p->next = NULL;
    rear->next = p;
    rear = p;
}

template<class DataType>
DataType LinkQueue<DataType>::DeQueue() {
    Node<DataType>* p = NULL;
    p = new Node<DataType>;
    DataType x;
    if (rear == front) throw "下溢";
    p = front->next;
    x = p->data;
    front->next = p->next;
    if (p->next == NULL)
        rear = front;
    delete p;
    return x;
}

template<class DataType>
DataType LinkQueue<DataType>::GetQueue() {
    if (front == rear) throw "下溢";
    return front->next->data;
}

template<class DataType>
int LinkQueue<DataType>::Empty() {
    if (front == rear) return 1;
    return 0;
}

```

Sicker.cpp

```
#include <iostream>
using namespace std;
#include "LinkQueue.h"

class sicker {
public:
    sicker();
    sicker(int num, char* name);
    sicker(const sicker& other);
    int GetNumber() { return number; }
    char* GetName() { return name; }
private:
    int number;//就诊号
    char* name;//姓名
};

sicker::sicker()
{
    number = -1;
    char Name[5] = "none";
    name = Name;
}

sicker::sicker(int num, char* name)
{
    number = num;
    this->name = name;
}

sicker::sicker(const sicker& other)
{
    this->name = other.name;
    this->number = other.number;
}

int num = 1;//当前挂号量+1
LinkQueue<sicker> SL;

void Register(char* name) {
    sicker* s = new sicker(num, name);
    SL.Enqueue(*s);
    cout << "挂号成功, 您的就诊号为: " << num << endl;
    num++;
}

void call() {
```

```

    sicker s(SL.GetQueue());
    cout << "请" << s.GetNumber() << "号：" << s.GetName() << "就诊。" << endl;
    SL.DeQueue();
}

int wait_number(int x) {
    sicker shead(SL.GetQueue());
    int count = x - shead.GetNumber();
    return count;
}

void guahao() {
    system("cls");
    char a[10];
    cout << "----- 挂 号 -----" << endl;
    cout << "请输入姓名：" << endl;
    cin >> a;
    Register(a);
    system("pause");
}

void Call() {
    call();
    system("pause");
}

void search() {
    system("cls");
    cout << "----- 挂 号 -----" << endl;
    cout << "请输入就诊号：" << endl;
    int num;
    cin >> num;
    cout << "等待人数：" << wait_number(num) << endl;
    system("pause");
}

void test() {
    char name1[10] = "张三";
    Register(name1);
    char name2[10] = "李四";
    Register(name2);
    char name3[10] = "王五";
    Register(name3);
    char name4[10] = "赵六";

```

```

    Register(name4);
    char name5[10] = "钱七";
    Register(name5);
    cout << "4号前还有：" << wait_number(4) << "人" << endl;
    call();
    call();
    cout << "4号前还有：" << wait_number(4) << "人" << endl;
    system("pause");
}

void menu(bool &IsRun) {
    int input;
    system("cls");
    cout << "----- 菜 单 -----" << endl;
    cout << "1. 患者挂号" << endl;
    cout << "2. 医生叫号" << endl;
    cout << "3. 查询等待人数" << endl;
    cout << "4. 运行测试" << endl;
    cout << "0. 退出系统" << endl;
    cin >> input;
    switch (input)
    {
        case 1:
            guahao();
            break;
        case 2:
            Call();
            break;
        case 3:
            search();
            break;
        case 4:
            test();
            break;
        default:
            IsRun = false;
            break;
    }
}

int main() {
    bool IsRun = true;
    while (IsRun) {
        menu(IsRun);
    }
}

```

```

    }

    return 0;
}

```

SeqStack.h

```

#pragma once

const int StackSize = 20;

template<class DataType>
class SeqStack {
public:
    SeqStack();
    ~SeqStack() {}
    void Push(DataType x);
    DataType Pop();
    DataType GetTop();
    int Empty();
private:
    DataType data[StackSize];
    int top;
};

template<class DataType>
SeqStack<DataType>::SeqStack() {
    top = -1;
}

template<class DataType>
void SeqStack<DataType>::Push(DataType x) {
    if (top == StackSize - 1) throw "下溢";
    top++;
    data[top] = x;
}

template<class DataType>
DataType SeqStack<DataType>::Pop() {
    if (top == -1) throw "下溢";
    DataType x = data[top];
    top--;
    return x;
}

template<class DataType>
DataType SeqStack<DataType>::GetTop() {

```



```

        if (top != -1) {
            return data[top];
        }
        throw "栈为空";
    }

```

```

template<class DataType>
int SeqStack<DataType>::Empty() {
    if (top == -1) return 1;
    return 0;
}

```

DecimalConversion.cpp

```

#include <iostream>
using namespace std;
#include "SeqStack.h"

void Decimal_Conversion(int number, int decimal) {
    char chars[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
    SeqStack<char> S;
    while (number != 0) {
        int c = number % decimal;
        S.Push(chars[c]);
        number = number / decimal;
    }
    while (!S.Empty())
        cout << S.Pop();
    cout << endl;
}

int main() {
    int number, decimal;
    cin >> number >> decimal;
    Decimal_Conversion(number, decimal);
    return 0;
}

```

LinkPriorityQueue.h

```

#pragma once

template<class DataType>
struct Node {
    DataType data;
    int priority;
}

```

```

        Node<DataType>* next;
};

template<class DataType>
class LinkPriorityQueue {
public:
    LinkPriorityQueue();
    LinkPriorityQueue(const LinkPriorityQueue& other);
    ~LinkPriorityQueue();
    void Push(DataType x, int priority);
    DataType Pop();
    DataType Get();
    DataType Priority_Get(int priority);
    void ResetPriority(const DataType x, int old_priority, int new_priority);
    int Empty();
    int Length();
    int Priority_Length(int priority);
private:
    Node<DataType>* front, * rear;
};

template<class DataType>
LinkPriorityQueue<DataType>::LinkPriorityQueue()
{
    Node<DataType>* p = new Node<DataType>;
    p->next = NULL;
    p->priority = -1;
    front = rear = p;
}

template<class DataType>
LinkPriorityQueue<DataType>::LinkPriorityQueue(const LinkPriorityQueue& other)
{
    this->front = other.front;
    this->rear = other.rear;
}

template<class DataType>
LinkPriorityQueue<DataType>::~~LinkPriorityQueue()
{
    Node<DataType>* p = new Node<DataType>;
    while (front != NULL) {
        p = front->next;
        delete front;
    }
}

```

```

        front = p;
    }
}

template<class DataType>
void LinkPriorityQueue<DataType>::Push(DataType x, int priority)
{
    Node<DataType>* q = new Node<DataType>{ x, priority, NULL };
    Node<DataType>* p = front;
    if (front->next == NULL || priority < front->next->priority) {
        q->next = front->next;
        front->next = q;
        if (rear == front) {
            rear = q;
        }
    }
    else {
        while (p->next != NULL && p->next->priority <= priority) {
            p = p->next;
        }
        q->next = p->next;
        p->next = q;
        if (p->next == NULL)
            rear = q;
    }
}

template<class DataType>
DataType LinkPriorityQueue<DataType>::Pop()
{
    DataType x;
    Node<DataType>* p = NULL;
    p = new Node<DataType>;
    if (rear == front) throw "下溢";
    p = front->next;
    x = p->data;
    front->next = p->next;
    if (p->next == NULL)
        rear = front;
    delete p;
    return x;
}

template<class DataType>

```

```

DataType LinkPriorityQueue<DataType>::Get()
{
    if (front == rear) throw "下溢";
    DataType x = front->next->data;
    return x;
}

template<class DataType>
DataType LinkPriorityQueue<DataType>::Priority_Get(int priority)
{
    if (front == rear) throw "下溢";
    Node<DataType>* p = front->next;
    while (p->priority < priority && p->next!=NULL)
        p = p->next;
    if (p->priority > priority || p->next==NULL) throw "不存在该优先级";
    DataType x = p->data;
    return x;
}

template<class DataType>
void LinkPriorityQueue<DataType>::ResetPriority(const DataType x, int old_priority, int
new_priority)
{
    Node<DataType>* p = front->next, * q = NULL;
    while (p->priority < old_priority && p != NULL)
        p = p->next;
    while (p->data != x && p->priority == old_priority && p != NULL) {
        if (p->next->data == x)
            q = p;
        p = p->next;
    }
    if (p == NULL) throw "下溢";
    if (p->priority != old_priority) throw "下溢";
    q->next = p->next;
    delete p;
    Push(x, new_priority);
}

template<class DataType>
int LinkPriorityQueue<DataType>::Empty()
{
    if (front == rear) return 1;
    return 0;
}

```

```

template<class DataType>
int LinkPriorityQueue<DataType>::Length()
{
    int count = 0;
    Node<DataType>* p = front->next;
    while (p != NULL) {
        count++;
        p = p->next;
    }
    return count;
}

template<class DataType>
int LinkPriorityQueue<DataType>::Priority_Length(int priority)
{
    int count = 0;
    Node<DataType>* p = front->next;
    while (p->priority != priority)
        p = p->next;
    while (p->priority == priority) {
        count++;
        p = p->next;
    }
    return count;
}

```

Sicker_Ordered.cpp

```

#include <iostream>
#include <string.h>
using namespace std;
#include "LinkPriorityQueue.h"

/*
编写一个程序，模拟病人（基本信息有就诊号和姓名，其中就诊号每天从 1 开始累计）到医院看病，
排队看医生的情形。在病人排队过程中，主要发生两件事。
1）、病人挂号，到诊室门口候诊。
2）、轮到某个号，该号对应的病人进入诊室就诊。
要求程序采用菜单方式，其选项及功能说明如下：
1）挂号候诊——更新待诊病人信息，新挂号的出现在最后。
2）叫号就诊——输出即将就诊的病人（队列中排最前面的病人）信息，该病人进入诊室就诊，更新
待诊病人信息，已就诊的病人不再出现。
3）查询——输出前面需要等的人数。
4）退出——退出运行。

```

```

*/

class sicker {
public:
    sicker();
    sicker(string num, string name);
    sicker(const sicker& other);
    string GetNumber() { return number; }
    string GetName() { return name; }
private:
    string number;//就诊号
    string name;//姓名
};

sicker::sicker()
{
    number = "None";
    name = "None";
}

sicker::sicker(string num, string name)
{
    this->number = num;
    this->name = name;
}

sicker::sicker(const sicker& other)
{
    this->name = other.name;
    this->number = other.number;
}

const string priority_char[4] = { "", "危", "急", "普" };
int num[4] = { 0,1,1,1 };
LinkPriorityQueue<sicker> SL;

void Register(string name, int priority) {
    string number = priority_char[priority];
    number += (char)(num[priority] + (int)('0'));
    num[priority]++;
    sicker s(number, name);
    SL.Push(s, priority);
    cout << "挂号成功, 您的就诊号为: " << number << endl;
}

void call() {

```

```

sicker s(SL.Get());
cout << "请" << s.GetNumber() << "号：" << s.GetName() << "就诊。" << endl;
SL.Pop();
}

int wait_number(string x) {
    int priority;
    if (x.substr(0, 2) == "危")priority = 1;
    else if (x.substr(0, 2) == "急")priority = 2;
    else if (x.substr(0, 2) == "普")priority = 3;
    else throw "就诊号不合法";
    int np = 1, count = 0;
    while (np < priority) {
        count = SL.Priority_Length(np);
        np++;
    }
    int now_num = 0, num = 0;
    for (int i = 2; x[i] != '\0'; i++)
        num = num * 10 + (int)(x[i] - (int)('0'));
    string nx = SL.Priority_Get(priority).GetNumber();
    for (int i = 2; nx[i] != '\0'; i++)
        now_num = now_num * 10 + (int)(nx[i] - (int)('0'));
    count += num - now_num;
    return count;
}

void guahao() {
    system("cls");
    string name;
    int priority;
    cout << "----- 挂 号 -----" << endl;
    cout << "请输入姓名：";
    cin >> name;
    cout << "请输入紧急程度（1 为危重，2 为紧急，3 为普通）：";
    cin >> priority;
    Register(name, priority);
    system("pause");
}

void Call() {
    call();
    system("pause");
}

```

```

void search() {
    system("cls");
    cout << "----- 挂 号 -----" << endl;
    cout << "请输入就诊号： ";
    string num;
    cin >> num;
    cout << "等待人数： " << wait_number(num) << endl;
    system("pause");
}

void test() {
    Register("张三", 1);
    Register("李四", 2);
    Register("王五", 2);
    Register("李六", 3);
    call();
    call();
    Register("赵七", 3);
    Register("钱八", 2);
    Register("孙九", 2);
    cout << "插入危 2 前，急 3 前面还有： " << wait_number("急 3") << "人" << endl;
    Register("李十", 1);
    cout << "插入危 2 后，急 3 前面还有： " << wait_number("急 3") << "人" << endl;
    Register("高玖", 2);
    call();
    call();
    Register("顾十", 2);
    Register("夏菲", 3);
    system("pause");
}

void menu(bool& IsRun, bool clear = true) {
    int input;
    if (clear)system("cls");
    cout << "----- 菜 单 -----" << endl;
    cout << "1. 患者挂号" << endl;
    cout << "2. 医生叫号" << endl;
    cout << "3. 查询等待人数" << endl;
    cout << "4. 运行测试" << endl;
    cout << "0. 退出系统" << endl;
    cin >> input;
    switch (input)
    {
    case 0:

```



```

        IsRun = false;
        break;
    case 1:
        guahao();
        break;
    case 2:
        Call();
        break;
    case 3:
        search();
        break;
    case 4:
        test();
        break;
    default:
        system("cls");
        cout << "输入非法, 请重新输入: " << endl;
        menu(IsRun, false);
        break;
    }
}

int main() {
    bool IsRun = true;
    while (IsRun)
        menu(IsRun);
    return 0;
}

```

四、运行输出结果：

（可以将运行结果抓图贴至此处）

```
F:\Code\NJUCM_DataStructur X + -
----- 菜 单 -----
1.患者挂号
2.医生叫号
3.查询等待人数
4.运行测试
0.退出系统
4
挂号成功, 您的就诊号为: 1
挂号成功, 您的就诊号为: 2
挂号成功, 您的就诊号为: 3
挂号成功, 您的就诊号为: 4
挂号成功, 您的就诊号为: 5
4号前还有: 3人
请1号: 张三就诊。
请2号: 李四就诊。
4号前还有: 1人
请按任意键继续... |
```

图 1 Sicker.cpp 测试运行结果

```
Microsoft Visual Studio 调试 X + -
12 8
14
F:\Code\NJUCM_DataStructure\230918_VS\x64\Debug\230918_VS.exe (进程 580696)已退出, 代码为 0。
请按任意键关闭此窗口... |
```

图 2 DecimalConversion.cpp 运行结果

```
F:\Code\NJUCM_DataStructur X + -
----- 菜 单 -----
1.患者挂号
2.医生叫号
3.查询等待人数
4.运行测试
0.退出系统
4
挂号成功, 您的就诊号为: 危3
挂号成功, 您的就诊号为: 急7
挂号成功, 您的就诊号为: 急8
挂号成功, 您的就诊号为: 普4
请危3号: 张三就诊。
请急3号: 钱八就诊。
挂号成功, 您的就诊号为: 普5
挂号成功, 您的就诊号为: 急9
挂号成功, 您的就诊号为: 急:
插入危2前, 急3前面还有: ~1人
挂号成功, 您的就诊号为: 危4
插入危2后, 急3前面还有: 0人
挂号成功, 您的就诊号为: 急;
请危4号: 李十就诊。
请急4号: 孙九就诊。
挂号成功, 您的就诊号为: 急<
挂号成功, 您的就诊号为: 普6
请按任意键继续... |
```

图 3 Sicker_Ordered.cpp 测试运行结果

五、调试和运行程序过程中产生的问题及采取的措施:

无

六、分析主要算法的时间复杂度，如果可以优化，提出优化方案：

LinkQueue.h

1. **LinkQueue():** $O(1)$
2. **~LinkQueue():** $O(n)$
3. **EnQueue(DataType x):** $O(1)$
4. **DeQueue():** $O(1)$
5. **GetQueue():** $O(1)$
6. **Empty():** $O(1)$

SeqStack.h

1. **SeqStack():** $O(1)$
2. **~SeqStack():** $O(1)$
3. **Push(DataType x):** $O(1)$
4. **Pop():** $O(1)$
5. **GetTop():** $O(1)$
6. **Empty():** $O(1)$

LinkPriorityQueue.h

1. **LinkPriorityQueue():** $O(1)$
2. **LinkPriorityQueue(const LinkPriorityQueue& other):** $O(n)$
3. **~LinkPriorityQueue():** $O(n)$
4. **Push(DataType x, int priority):** $O(n)$
5. **Pop():** $O(1)$
6. **Get():** $O(1)$
7. **Priority_Get(int priority):** $O(n)$
8. **ResetPriority(const DataType x, int priority):** $O(n)$
9. **Empty():** $O(1)$

七、对数据结构教学的意见和建议：

无