LAB 4: PIZZA QUEUES BY:AMULDEEP DHILLON

Queues

We will be using queues to organize both pizza orders and the available drivers to deliver them.
We will be using two different types of queues, one for each queue.

- We will use a Linked List Queue for the Orders which contain the pizza and the address.
- We will use a Ring Buffer Array Queue for the drivers

These could both be done using the other type of queue. We are using both just for educational purposes.

Pizza.h

```
#pragma once
#include <iostream>
#include "config.h"
#include <FL/Fl_Cairo_Window.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/fl_ask.H>
#include "LLQueue.h"
#include "RBQueue.h"
extern LLQUEUE orderQueue;
extern RBQUEUE driverQueue;
void order_cb(Fl_Button*,void*);
extern Fl_Input* pizza;
void driver_cb(Fl_Button*,void*);
extern Fl_Input* driverName;
void dispatch(void*);
extern Fl_Input* addressName;
extern bool status;
void deliver();
```

Contains all shared variables and functions

Order.h

```
#pragma once
class Order{
private:
    std::string order;
    std::string address;
public:
    std::string getOrder(){return order;}
    std::string getAddress(){return address;}
    void setOrder(std::string o) {order = o;}
    void setAddress(std::string o) {address = o;}
};
```

Contains the class Order

LLQueue.h

```
#pragma once
#include "pizza.h"
#include "order.h"


struct NODE {
    Order info;
    NODE *next;
};

class LLQUEUE
{
private:
    NODE *front;
    NODE *rear;
public:
    LLQUEUE(){front = rear = 0;}
    ~LLQUEUE();
    bool Insert(Order &info);
    bool Remove(Order &info);
    bool isEmpty() {return(front == 0);}
};
```

Contains class LLQueue and the NODE definition

LL Queue Destructor

```
#include "pizza.h"

LLQUEUE::~LLQUEUE (){
    NODE *next;
    while(front){
        next = front->next;
        delete front;
        front = next;
    }
}
```

Defines LLQueue's destructor

LL Queue Insert

```
#include "pizza.h"

bool LLQUEUE::Insert (Order &info){
    NODE *newnode = new NODE;
    if (!newnode) return false;
    newnode -> info = info;
    newnode -> next = 0;

    if(rear == 0)
        front = rear = newnode;
    else{
        rear->next = newnode;
        rear = newnode;
    }
    return true;
}
```

Contains LLQueue's Insert function

LL Queue Remove

```
#include "pizza.h"
bool LLQUEUE::Remove (Order &info){
    if(front == 0) return false;
    info = front -> info;

    NODE *next = front->next;
    delete front;
    front = next;
    if(front == 0)
        rear = 0;
    return true;
}
```

Contains LLQueue's Remove function

RB Queue.h

```
#pragma once
#include "pizza.h"
#include <string>
const int BUFSIZE = 256;
class RBQUEUE{
private:
    std::string name;
    std::string buf[BUFSIZE];
    int front;
    int rear;
    int nextIndex(int index){
        if(++index == BUFSIZE) index = 0;
        return index;
    }
public:
    void setName(std::string s) {name = s;}
    RBQUEUE () {front = rear = 0;}
    ~RBQUEUE () {/* */}
    bool Insert(std::string s);
    bool Remove (std::string &s);
    bool isEmpty() {return (front==rear);}
    bool isFull() {return (nextIndex(rear) == front);}
};
```

Contains the class RBQueue

RB Queue

```
#include "pizza.h"

bool RBQUEUE::Insert (std::string s){
    if (isFull()) return false;
    buf[rear] = s;
    rear = nextIndex(rear);
    return true;
}

bool RBQUEUE::Remove (std::string &s){
    if(isEmpty()) return false;
    s = buf[front];
    front = nextIndex(front);
    return true;
}
```

Contains the definitions of RBQueue's functions

Order cb

```
#include "pizza.h"
LLQUEUE orderQueue;
void order_cb(Fl_Button*,void*){

    Order o;
    std::cout << pizza->value() << std::endl;
    o.setOrder(pizza->value());


    std::cout << addressName->value() << std::endl;
    o.setAddress(addressName->value());

    status = orderQueue.Insert(o);

    if(status == true)
        std::cout << "Inserted" << std::endl;
    else std::cout << "Queue overflowed" << std::endl;



    std::string s = "Magnificent Choice";
    fl_alert(s.c_str());
}
```

- Gets the Pizza from the GUI
- Gets the Address from the GUI
- Places both in an Order type
- Inserts Order into the Queue

Driver cb

```
#include "pizza.h"
RBQUEUE driverQueue;
void driver_cb(Fl_Button*,void*){

    std::cout << driverName->value() << std::endl;
    status = driverQueue.Insert(driverName->value());
    if(status == true)
        std::cout << "Driver Inserted" << std::endl;
    else std::cout << "Queue Overflow" << std::endl;
    std::string s = "Driver is Ready";
    fl_alert(s.c_str());
}
```

- Gets driver from the GUI
- Places driver into the queue

Dispatch

```
#include "pizza.h"
void dispatch(void*){
    std::cout << "ok" << std::endl;
    deliver();
    Fl::repeat_timeout(10,dispatch);
}
```

Repeats the deliver function after a set amount of time

Deliver

```
#include "pizza.h"

void deliver(){
    while(status == true && !driverQueue.isEmpty() && !orderQueue.isEmpty()){
        Order myOrder;
        orderQueue.Remove(myOrder);
        std::string driverName ;
        driverQueue.Remove(driverName);
        std::string s = "Driver " + driverName + " is delivering the "
        + myOrder.getOrder() + " pizza to " + myOrder.getAddress();
        fl_alert(s.c_str());
    }
}
```

Sends the first driver and first pizza out of the queue, if they exist

Main

```
#include "pizza.h"
Fl_Input* pizza;Fl_Input* driverName;Fl_Input* addressName;
bool status;
int main(){
    Fl_Cairo_Window cw(300,300);
    cw.color(fl_rgb_color(76,201,140));
    Fl_Button order(150,150,50,30,"Order");
    order.callback((Fl_Callback*)order_cb);
    order.color(fl_rgb_color(100,188,231));
    pizza = new Fl_Input(100,100,180,20,"Pizza:");
    addressName = new Fl_Input(100,50,180,20,"Address:");
    Fl_Button driver(150,250,50,30,"Driver");
    driver.callback((Fl_Callback*)driver_cb);
    driver.color(fl_rgb_color(100,188,231));
    driverName = new Fl_Input(100,200,180,20,"Driver:");
    cw.show();
    std::string s = "Don't forget to submit an address";
    fl_alert(s.c_str());
    Fl::add_timeout(5,dispatch);
    Fl::run();
}
```

Creates the GUI and executes all the other function

Test

The test of the Program can be seen in the video file included