Lab 4 Pizza Program

Generated by Doxygen 1.8.8

Tue Oct 17 2017 15:29:18

Contents

1 Testcases

page Specification

This program is a pizza program that allows users to have a GUI interface that uses Queues that have been implemented by the use of Lists. We create 3 different queues which are implemented from the uses of queues, which allows the user to choose their choice of pizza and having it implemented into the program. After the user has entered their choice of pizza, it is then passed into the 3 different queues: Uncooked Q, Cooked Q and Driver Q. page Analysis

- The input will be the choice of pizza the user has chosen and entering it into the program. Also, the user may need to hire a driver to deliver their pizza and enter an address for a location for them to deliver.

- The output will be the 3 lists of Queues: Uncooked Q, Cooked Q, and Driver Q. These list hold the name of the dirver and the uncooked and cooked pizza which will be ready to deliver. The output will have an alert message which will tell the user the pizza have been delivered by the driver with the correct pizza to the correct address. It will also tell the user if there are no drivers to take the order or no orders for drivers to take.

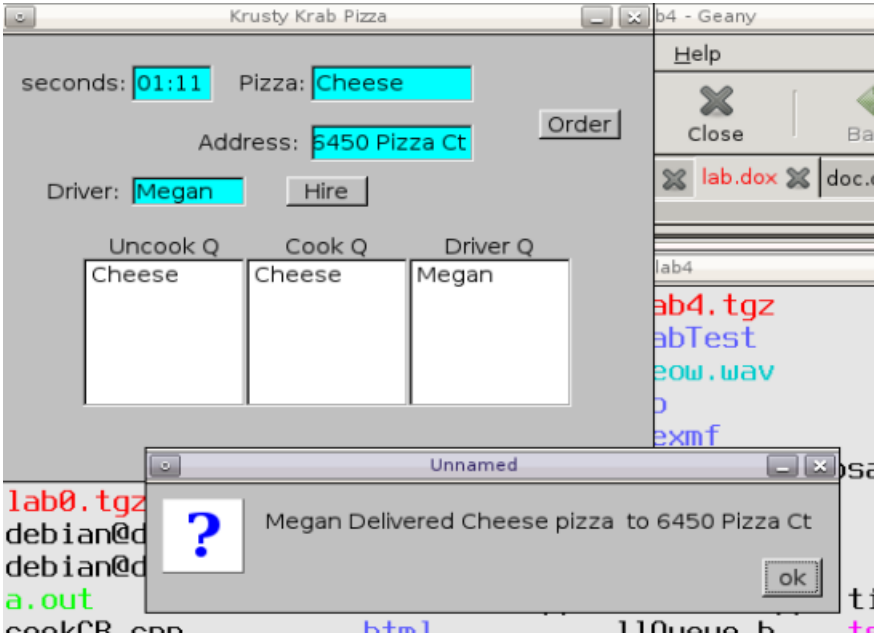
- The overall algorithm is to input pizza types and customers addresses and have it enter a queue of pizza from uncooked to cooked pizza then the driver that the user hires will take the pizza to the location which the user added also which is an address for the driver to get to, and as the driver arrives to their destination, the user will get a pop up notification which the pizza has been delivered.page Design

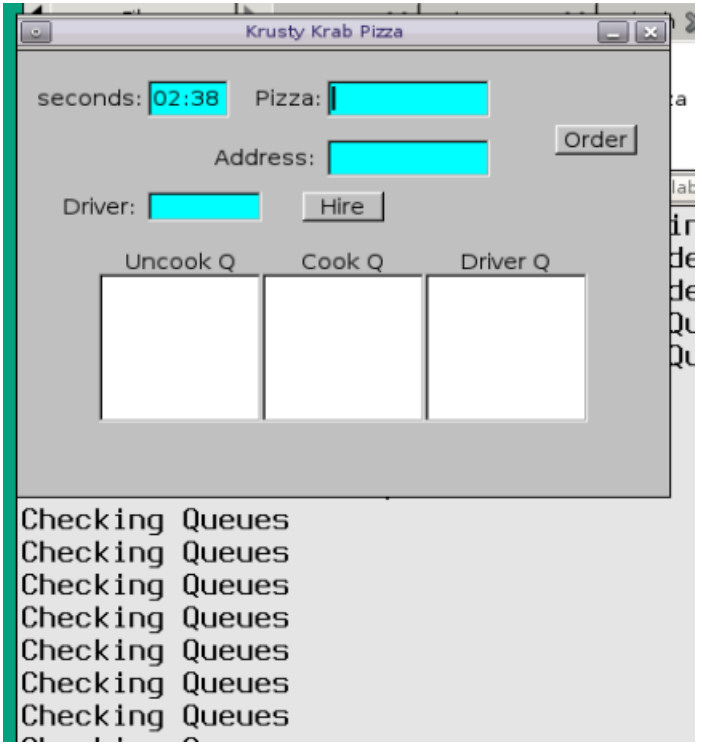
The Design of the program was to have a simple GUI for a owner of a pizza place. Having the owner of the pizza place manage the orders and the drivers. The owner, which is the user, enters their choice of pizza into the program. After they've chosen their pizza, they also enter the address of the place of delivery as well as entering a name of the delivery driver. The program is worked around by a clock which is implemented by the use of seconds. After an amount of seconds have been passed, the choice of pizza will first pass through the Uncooked

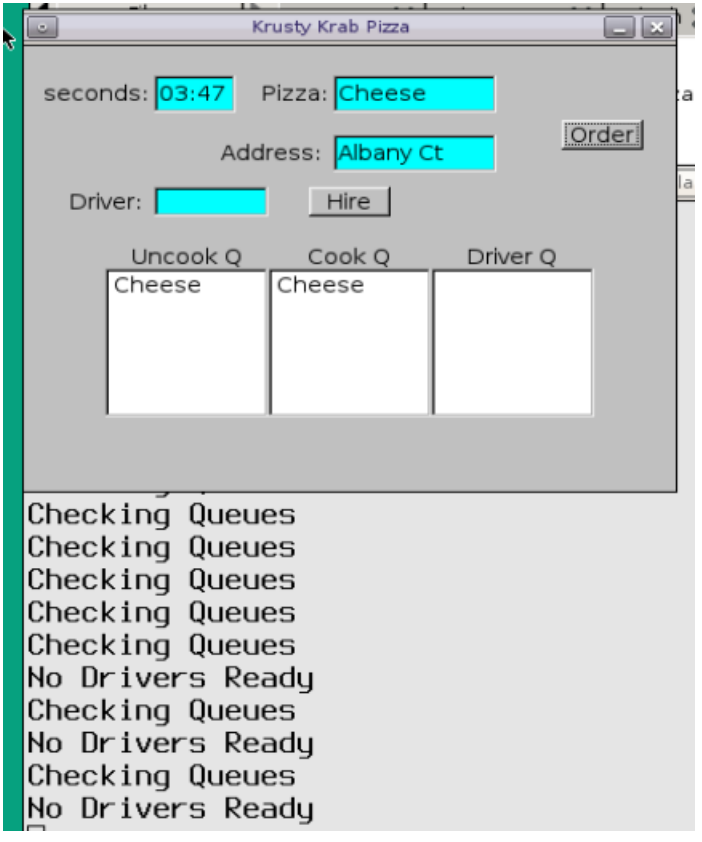
Queue, then passing into the Cooked Queue. Then the chosen delivery driver is passed into the Driver Queue, waiting to notify the owner of the pizza place that the chosen delivery driver has delivered the chosen pizza at the entered address. All test case videos are within the tgz file

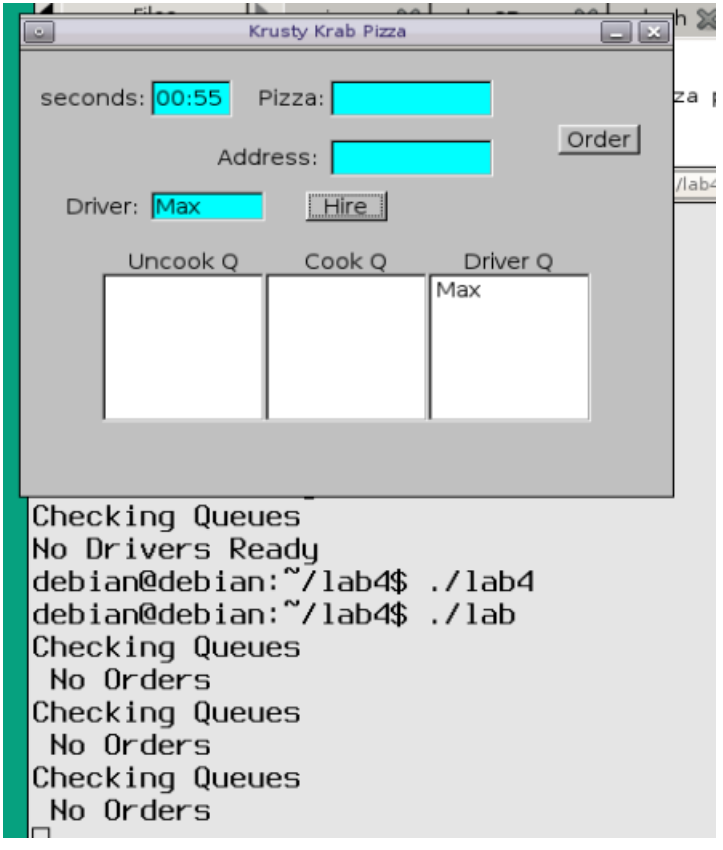
- tst1.mp4 has the the test case in which the user inputs a pizza, address, and driver and then gets delivered in a certain time.
- tst2.mp4 demonstrates the program uses queues as it shows the queue is being checked for orders and drivers.
- tst3.mp4 demonstrates queues as the user enters an order and it enters the queue, but notifies the user that there are no drivers
- tst4.mpt demonstrates queues as the user hires a driver and enters the driver queue,but has no orders in the other queue, so the user is notified it has no orders

2 picture 1


Krusty Krab Pizza
seconds: 01:11
Pizza: Cheese
Order
Address: 6450 Pizza Ct
Driver: Megan
Hire
Uncook Q
Cook Q
Driver Q
Cheese
Cheese
Megan
Unnamed
Megan Delivered Cheese pizza to 6450 Pizza Ct
ok
b4 - Geany
Help
Close
lab.dox
lab4
lab0.tgz
a.out


Krusty Krab Pizza
seconds: 02:38
Pizza:
Order
Address:
Driver:
Hire
Uncook Q
Cook Q
Driver Q
Checking Queues
Checking Queues
Checking Queues
Checking Queues
Checking Queues
Checking Queues
Checking Queues


Krusty Krab Pizza
seconds: 03:47
Pizza: Cheese
Order
Address: Albany Ct
Driver:
Hire
Uncook Q
Cook Q
Driver Q
Cheese
Cheese
Checking Queues
Checking Queues
Checking Queues
Checking Queues
Checking Queues
No Drivers Ready
Checking Queues
No Drivers Ready
Checking Queues
No Drivers Ready


Krusty Krab Pizza
seconds: 00:55
Pizza:
Order
Address:
Driver: Max
Hire
Uncook Q
Cook Q
Driver Q
Max
Checking Queues
No Drivers Ready
debian@debian:~/lab4$ ./lab4
debian@debian:~/lab4$ ./lab
Checking Queues
No Orders
Checking Queues
No Orders
Checking Queues
No Orders

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Class Documentation

5.1 LLQUEUE Class Reference

`#include <llQueue.h>`

Public Member Functions

- LLQUEUE ()
- ~LLQUEUE ()
- bool Insert (Order &info)
- bool Remove (Order &info)
- bool isEmpty ()

5.1.1 Constructor & Destructor Documentation

5.1.1.1 LLQUEUE::LLQUEUE ( ) [inline]

```
11 {front = rear = 0;}
```

5.1.1.2 LLQUEUE::~LLQUEUE ( ) [inline]

Destructor that delets all elements reaming in the queue

```
16         {
17             NODE *next;
18             while (front)
```

```
19                  {
20                      next = front->next;
21                      delete front;
22                      front = next;
23                  }
24          }//end of Destructor
```

5.1.2 Member Function Documentation

5.1.2.1 bool LLQUEUE::Insert (Order & *info*) [inline]

Inserts a new element at the rear of the queue Returns true if successful, false if out of memory

```
30          {
31              NODE *newnode = new NODE;
32              if(!newnode)
33                  return false;
34              newnode->info = info;
35              newnode->next = 0;
36              if(rear == 0)
37                  front = rear = newnode;
38              else {
39                  rear->next = newnode;
40                  rear = newnode;
41                  }
42              return true;
43          }
```

5.1.2.2 bool LLQUEUE::isEmpty ( ) [inline]

```
61 {return(front == 0);}
```

5.1.2.3 bool LLQUEUE::Remove (Order & *info*) [inline]

Retrieves and removes the first element from the queue. Returns true if successful, false if queue is empty

```
49          {
50
51              if (front == 0)
52                  return false;
53              info = front->info;
54              NODE *next = front->next;
55              delete front;
56              front = next;
57              if (front == 0)
58                  rear = 0;
59              return true;
60          }
```

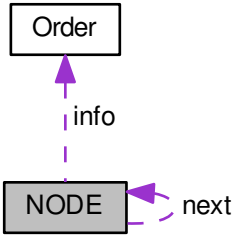
The documentation for this class was generated from the following file:

- llQueue.h

5.2 NODE Struct Reference

```
#include <llQueue.h>
```

Collaboration diagram for NODE:



Public Attributes

- Order info
- NODE * next

5.2.1 Member Data Documentation

5.2.1.1 Order NODE::info

5.2.1.2 NODE* NODE::next

The documentation for this struct was generated from the following file:

- llQueue.h

5.3 Order Class Reference

`#include <order.h>`

Public Member Functions

- void setOrder (std::string s)
- std::string getOrder ()
- void setAddress (std::string a)
- std::string getAddress ()

5.3.1 Member Function Documentation

5.3.1.1 std::string Order::getAddress ( ) [inline]

```
9 {return address;}
```

5.3.1.2 std::string Order::getOrder ( ) [inline]

```
7 {return order;}
```

5.3.1.3 void Order::setAddress (std::string *a*) [inline]

```
8 {address = a;}
```

5.3.1.4 void Order::setOrder (std::string *s*) [inline]

```
6 {order = s;}
```

The documentation for this class was generated from the following file:

- order.h

5.4 RBQUEUE Class Reference

```
#include <rbQueue.h>
```

Public Member Functions

- RBQUEUE ()
- ~RBQUEUE ()
- bool Insert (std::string s)
- bool Remove (std::string &s)
- bool isEmpty ()
- bool isFull ()
- void setName (std::string s)
- std::string getName ()
- std::string getFirst ()

5.4.1 Constructor & Destructor Documentation

5.4.1.1 RBQUEUE::RBQUEUE ( ) [inline]

```
6 {front = rear = 0;}
```

5.4.1.2 RBQUEUE::~RBQUEUE ( ) [inline]

```
7 {}
```

5.4.2 Member Function Documentation

5.4.2.1 std::string RBQUEUE::getFirst ( ) [inline]

```
14 {return buf[front];}
```

5.4.2.2 std::string RBQUEUE::getName ( ) [inline]

```
13 {return name;}
```

5.4.2.3 bool RBQUEUE::Insert (std::string *s*)

Appends s at the end of the buffer Returns true if successful, false is the queue is full

```
8 {
9     if(isFull())
10         return false;
11     buf[rear] = s;
12     rear = nextIndex(rear);
```

```
13      return true;
14
15  }
```

5.4.2.4 bool RBQUEUE::isEmpty ( ) [inline]

```
10 {return (front == rear);}
```

5.4.2.5 bool RBQUEUE::isFull ( ) [inline]

```
11 {return(nextIndex(rear) == front);}
```

5.4.2.6 bool RBQUEUE::Remove (std::string & *s*)

Retrieces and removes the element from the front of the buffer. Returns true is successful, false if the queue is empty.

```
22 {
23      if(isEmpty())
24          return false;
25      s = buf[front];
26      front = nextIndex(front);
27      return true;
28
29  }
```

5.4.2.7 void RBQUEUE::setName (std::string *s*) [inline]

```
12 {name = s;}
```

The documentation for this class was generated from the following files:

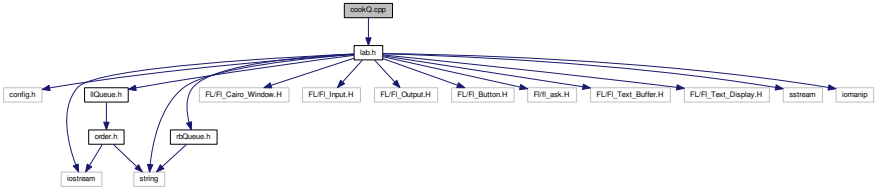
- rbQueue.h
- rbQueue.cpp

6 File Documentation

6.1 cookQ.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for cookQ.cpp:



Functions

- void cookList (void ∗)

  This is the callback function for the cook time of the pizza.

6.1.1 Function Documentation

6.1.1.1 void cookList (void ∗)

This is the callback function for the cook time of the pizza.

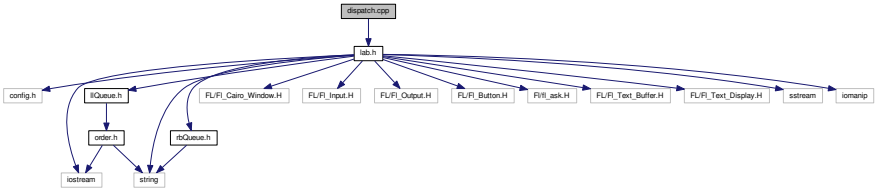
void pointer are not used returns void

```
4 {
5     std::string s = o.getOrder() + "\n";
6     buff1->text(s.c_str());
7
8 }
```

6.2 dispatch.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for dispatch.cpp:



Functions

- void dispatch (void *)

  This is the callback function to deliver pizza checks in queues if anything needs to be delivered void pointers are not used returns void.

Variables

- LLQUEUE orderQueue
- RBQUEUE driverQueue
- Order o

6.2.1 Function Documentation

6.2.1.1 void dispatch (void *)

This is the callback function to deliver pizza checks in queues if anything needs to be delivered void pointers are not used returns void.

```
6 {
7
8     Fl::repeat_timeout(15,dispatch);
9     std::string name;
10     std::cout << "Checking Queues\n";
11     name = driverQueue.getFirst();
12     if(orderQueue.isEmpty() && driverQueue.isEmpty());
13     else if (!orderQueue.isEmpty() && !driverQueue.
      isEmpty())
14     {
15         orderQueue.Remove(o);
16         driverQueue.Remove(name);
17         std::string msg;
18         msg = name + " Delivered " + o.getOrder() + " pizza ";
19         msg = msg + " to " + o.getAddress();
20         Fl::add_timeout(10,dispatch);
21         Fl::remove_timeout(dispatch);
```

```
22          switch(fl_choice(msg.c_str(),0,0,"ok")){
23      case 2:
24          Fl::add_timeout(5,dispatch);
25          break;
26              }
27              driverQueue.Insert(name);
28          }
29      else if (!orderQueue.isEmpty() && driverQueue.
        isEmpty())
30          { std::cout << "No Drivers Ready \n";}
31      else if (orderQueue.isEmpty() && !driverQueue.
        isEmpty())
32          { std::cout << " No Orders \n";}
33  }
```

6.2.2 Variable Documentation

6.2.2.1 RBQUEUE driverQueue

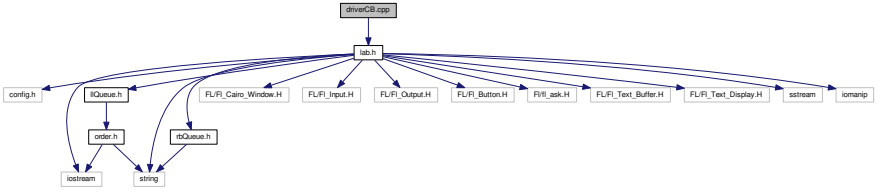
6.2.2.2 Order o

6.2.2.3 LLQUEUE orderQueue

6.3 driverCB.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for driverCB.cpp:



Functions

- void driver_cb (Fl_Button *, void *)

  This is the callback function for drivers information.

6.3.1 Function Documentation

6.3.1.1 void driver_cb (Fl_Button * , void *)

This is the callback function for drivers information.

void pointers are not used returns void

```
4 {
```

```
5       std::string s;
6       if(driverQueue.isFull())
7       {
8           s = "The driver queue is full";
9           fl_alert(s.c_str());
10       }
11       else if(!driverQueue.isFull())
12       {
13           driverQueue.setName(driverName->value());
14           driverQueue.Insert(driverName->value());
15           s = driverQueue.getName() + "\n";
16           buff->text(s.c_str());
17       }
18  }
```

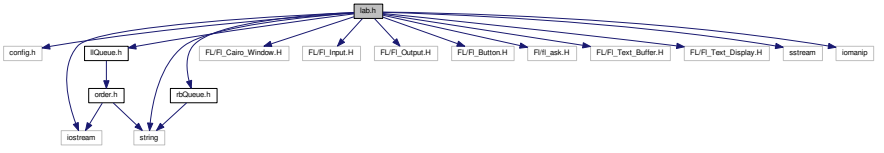
6.4 lab.dox File Reference

6.5 lab.h File Reference

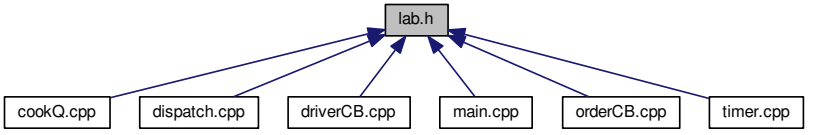
```
#include "config.h"
```

```
#include "llQueue.h"
#include "rbQueue.h"
#include <FL/Fl_Cairo_Window.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_Button.H>
#include <Fl/fl_ask.H>
#include <FL/Fl_Text_Buffer.H>
#include <FL/Fl_Text_Display.H>
#include <iostream>
#include <sstream>
#include <iomanip>
#include <string>
```

Include dependency graph for lab.h:



This graph shows which files directly or indirectly include this file:



Functions

- void dispatch (void ∗)

  This is the callback function to deliver pizza checks in queues if anything needs to be delivered void pointers are not used returns void.

- void driver_cb (Fl_Button ∗, void ∗)

  This is the callback function for drivers information.

- void order_cb (Fl_Button ∗, void ∗)

  This is the callback function to order a pizza and enter an address. void pointers are not used returns void.

- void cookList (void ∗)

  This is the callback function for the cook time of the pizza.

- void timer (void ∗)

  This is a callback function for the timer of the pizza.

Variables

- Fl_Input * pizza
- Fl_Input * address
- Fl_Input * driverName
- Fl_Output * watch
- Fl_Text_Buffer * buff
- Fl_Text_Buffer * buff1
- Fl_Text_Buffer * buff2
- LLQUEUE orderQueue
- RBQUEUE driverQueue
- Order o

6.5.1 Function Documentation

6.5.1.1 void cookList (void *)

This is the callback function for the cook time of the pizza.

void pointer are not used returns void

```
4 {
5     std::string s = o.getOrder() + "\n";
6     buff1->text(s.c_str());
7
8 }
```

6.5.1.2 void dispatch (void *)

This is the callback function to deliver pizza checks in queues if anything needs to be delivered void pointers are not used returns void.

```
6 {
7
8     Fl::repeat_timeout(15,dispatch);
9     std::string name;
10     std::cout << "Checking Queues\n";
11     name = driverQueue.getFirst();
12     if(orderQueue.isEmpty() && driverQueue.isEmpty());
13     else if (!orderQueue.isEmpty() && !driverQueue.
      isEmpty())
14     {
15         orderQueue.Remove(o);
16         driverQueue.Remove(name);
17         std::string msg;
18         msg = name + " Delivered " + o.getOrder() + " pizza ";
19         msg = msg + " to " + o.getAddress();
20         Fl::add_timeout(10,dispatch);
21         Fl::remove_timeout(dispatch);
22         switch(fl_choice(msg.c_str(),0,0,"ok")){
23     case 2:
24         Fl::add_timeout(5,dispatch);
25         break;
26             }
27             driverQueue.Insert(name);
28         }
29     else if (!orderQueue.isEmpty() && driverQueue.
      isEmpty())
30         { std::cout << "No Drivers Ready \n";}
```

```
31      else if (orderQueue.isEmpty() && !driverQueue.
        isEmpty())
32           { std::cout << " No Orders \n";}
33  }
```

6.5.1.3 void driver_cb (Fl_Button * , void *)

This is the callback function for drivers information.

void pointers are not used returns void

```
4  {
5      std::string s;
6      if(driverQueue.isFull())
7      {
8          s = "The driver queue is full";
9          fl_alert(s.c_str());
10      }
11      else if(!driverQueue.isFull())
12      {
13          driverQueue.setName(driverName->value());
14          driverQueue.Insert(driverName->value());
15          s = driverQueue.getName() + "\n";
16          buff->text(s.c_str());
17      }
18  }
```

6.5.1.4 void order_cb (Fl_Button * , void *)

This is the callback function to order a pizza and enter an address. void pointers are not used returns void.

```
4 {
5
6     o.setOrder(pizza->value());
7     o.setAddress(address->value());
8     orderQueue.Insert(o);
9     std::string s;
10     //std::string top[o.getOrder()];
11     //top[] = {o.getOrder()};
12     //s += top[o.getOrder()] + ;
13     s = o.getOrder() +  '\n';
14     buff2->text(s.c_str());
15     Fl::add_timeout(5,cookList);
16 }
```

6.5.1.5 void timer (void *)

This is a callback function for the timer of the pizza.

void pointer are not used returns void

```
3 {
4
5     static int s = 0;
6     static int m = 0;
7     std::ostringstream oss;
8     s++;    if(s == 59){s = 0; m++;}
9     oss << std::setfill('0');
10     oss << std::setw(2) << m << ":" << std::setw(2) << s;
11     //fl_alert("1 sec");
12     watch->value(oss.str().c_str());
13     //Fl::repeat_timeout(1,timer);
14
```

```
15      //static std:: string str;
16      //std::string pizzas[] = {"veggie","pepperoni","Hawaiian","Cheese"};
17      //if (s%10 ==0)
18      //{
19        //  str += pizzas[s%4] + "\n";
20          //buff->text(str.c_str());
21      //}
22          Fl::repeat_timeout(1,timer);
23
24  }
```

6.5.2 Variable Documentation

6.5.2.1 Fl_Input* address

6.5.2.2 Fl_Text_Buffer* buff

6.5.2.3 Fl_Text_Buffer* buff1

6.5.2.4 Fl_Text_Buffer* buff2

6.5.2.5 Fl_Input* driverName

6.5.2.6 RBQUEUE driverQueue

6.5.2.7 Order o

6.5.2.8 LLQUEUE orderQueue

6.5.2.9 Fl_Input* pizza

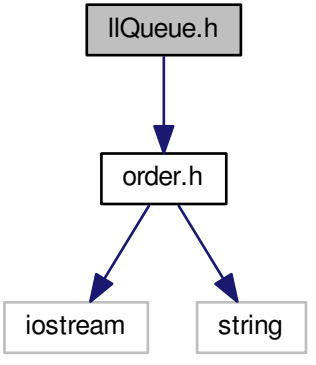
6.5.2.10 Fl_Output* watch

6.6 llQueue.cpp File Reference

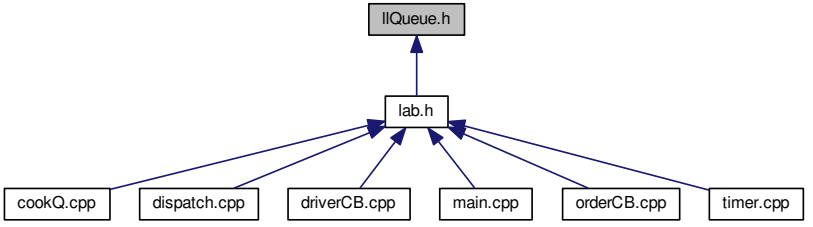
6.7 llQueue.h File Reference

```
#include "order.h"
```

Include dependency graph for llQueue.h:



This graph shows which files directly or indirectly include this file:



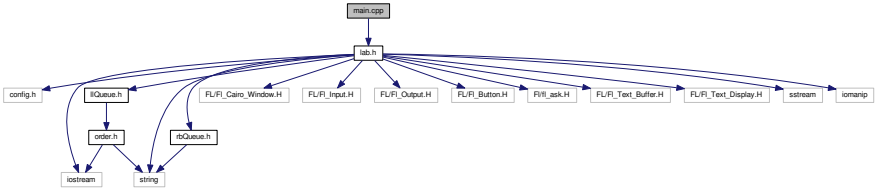
Classes

- struct NODE
- class LLQUEUE

6.8 main.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

Variables

- Fl_Input * pizza
- Fl_Input * address
- Fl_Input * driverName
- Fl_Output * watch
- Fl_Text_Buffer * buff
- Fl_Text_Buffer * buff1
- Fl_Text_Buffer * buff2
- Fl_Text_Display * cookQ
- Fl_Text_Display * uncookQ
- Fl_Text_Display * DriverQ

6.8.1 Function Documentation

6.8.1.1 int main ( )

```
13          {
14
15    Fl_Cairo_Window cw(400,300); // 400 = width, 300 height of window
16    cw.label("Krusty Krab Pizza"); //title of the window
17    cw.color(FL_GRAY);
18
19    pizza = new Fl_Input(190,20,100,25,"Pizza:");
20    pizza->color(FL_CYAN);
21    address = new Fl_Input(190,60,100,25,"Address: ");
22    address->color(FL_CYAN);
23
24    buff2 = new Fl_Text_Buffer();
25    uncookQ = new Fl_Text_Display(50,150,100,100,"Uncook Q");
26    uncookQ->buffer(buff2);
27
28    buff1 = new Fl_Text_Buffer();
29    cookQ = new Fl_Text_Display(150,150,100,100,"Cook Q");
30    cookQ->buffer(buff1);
31
32    watch = new Fl_Output(80,20,50,25,"seconds:");
33    watch->color(FL_CYAN);
34
35    Fl_Button b(330,50,50,20,"Order");
36    b.callback((Fl_Callback*)order_cb);
37
38    Fl_Button d(175,95,50,20,"Hire");
39    d.callback((Fl_Callback*)driver_cb);
40    driverName = new Fl_Input(80,95,70,20,"Driver: ");
```

```
41      driverName->color(FL_CYAN);
42
43      buff = new Fl_Text_Buffer();
44      DriverQ = new Fl_Text_Display(250,150,100,100,"Driver Q");
45      DriverQ->buffer(buff);
46
47      cw.show();
48      Fl::add_timeout(1,timer);
49      Fl::repeat_timeout(15,dispatch);
50      return Fl::run();
51
52  }
```

6.8.2 Variable Documentation

6.8.2.1 Fl_Input* address

6.8.2.2 Fl_Text_Buffer* buff

6.8.2.3 Fl_Text_Buffer* buff1

6.8.2.4 Fl_Text_Buffer* buff2

6.8.2.5 Fl_Text_Display* cookQ

6.8.2.6 Fl_Input* driverName

6.8.2.7 Fl_Text_Display* DriverQ

6.8.2.8 Fl_Input* pizza

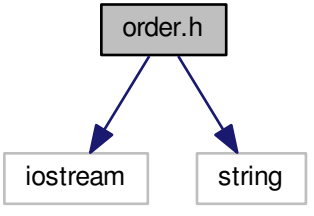
6.8.2.9 Fl_Text_Display* uncookQ

6.8.2.10 Fl_Output* watch

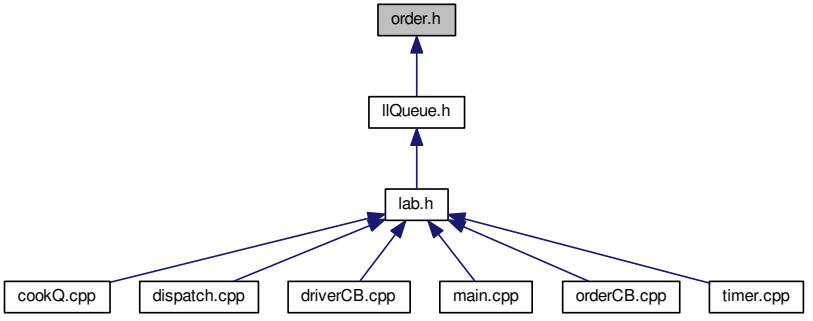
6.9 order.h File Reference

```
#include <iostream>
#include <string>
```

Include dependency graph for order.h:



This graph shows which files directly or indirectly include this file:



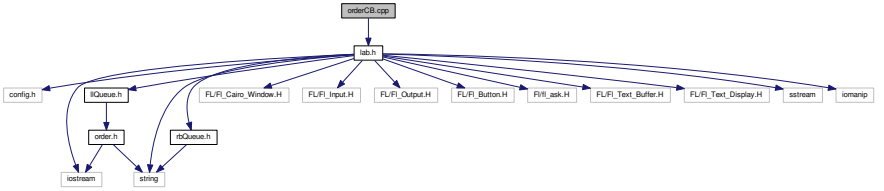
Classes

- class Order

6.10 orderCB.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for orderCB.cpp:



Functions

- void order_cb (Fl_Button *, void *)

  This is the callback function to order a pizza and enter an address. void pointers are not used returns void.

6.10.1 Function Documentation

6.10.1.1 void order_cb (Fl_Button * , void *)

This is the callback function to order a pizza and enter an address. void pointers are not used returns void.

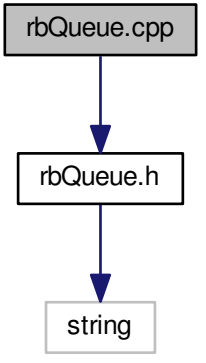
```
4 {
5
6     o.setOrder(pizza->value());
7     o.setAddress(address->value());
8     orderQueue.Insert(o);
9     std::string s;
```

```
10      //std::string top[o.getOrder()];
11      //top[] = {o.getOrder()};
12      //s += top[o.getOrder()] + ;
13      s = o.getOrder() +  '\n';
14      buff2->text(s.c_str());
15      Fl::add_timeout(5,cookList);
16  }
```

6.11 rbQueue.cpp File Reference

```
#include "rbQueue.h"
```

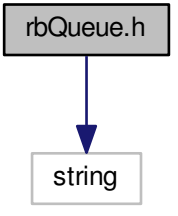
Include dependency graph for rbQueue.cpp:



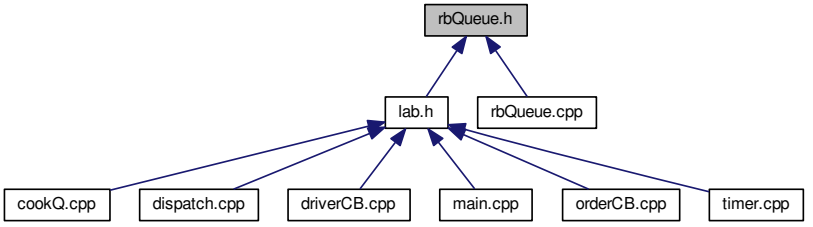
6.12 rbQueue.h File Reference

```
#include <string>
```

Include dependency graph for rbQueue.h:



This graph shows which files directly or indirectly include this file:



Classes

- class RBQUEUE

Variables

- const int BUFSIZE = 8

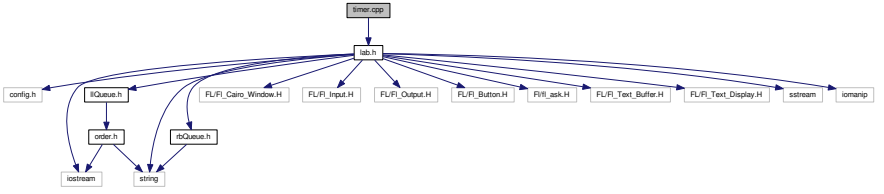
6.12.1 Variable Documentation

6.12.1.1 const int BUFSIZE = 8

6.13 timer.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for timer.cpp:



Functions

- void timer (void ∗)

  This is a callback function for the timer of the pizza.

6.13.1 Function Documentation

6.13.1.1 void timer (void ∗)

This is a callback function for the timer of the pizza.

void pointer are not used returns void

```
3 {
```

```
4
5     static int s = 0;
6     static int m = 0;
7     std::ostringstream oss;
8     s++;    if(s == 59){s = 0; m++;}
9     oss << std::setfill('0');
10     oss << std::setw(2) << m << ":" << std::setw(2) << s;
11     //fl_alert("1 sec");
12     watch->value(oss.str().c_str());
13     //Fl::repeat_timeout(1,timer);
14
15     //static std:: string str;
16     //std::string pizzas[] = {"veggie","pepperoni","Hawaiian","Cheese"};
17     //if (s%10 ==0)
18     //{
19       //  str += pizzas[s%4] + "\n";
20         //buff->text(str.c_str());
21     //}
22         Fl::repeat_timeout(1,timer);
23
24 }
```

Index