Pizza Delivery

Generated by Doxygen 1.8.8

Sat Oct 21 2017 12:51:37

Contents

1 Specification

This program uses FLTK as a GUI. The main purpose of the program is to use queue for a pizza delivery. This function gets the drivers' names from the user using FLTK and stores the drivers in the driver ring buffer Queue. This function also gets the pizza input in the FLTK by the user and stores it in "order Queue" which is Linked list Queue and then few seconds later puts the pizza in the "cooked Queue" after it is cooked by calling the cooked_cb function. It then calls back the delivery_cb which removes the driver from the drivers RBQueue and the driver gets put into the temporary RBQ for delivery, and the pizza is also removed from the cooked LLQueue. If there is no driver in the queue the function then checks again after 5 seconds if there are any drivers. After 100 seconds the drivers who were on delivery are removed from the temporary RBQ and inserted back into the drivers Q.

2 Analysis

- The input will be: The Pizza and the delivery address will be input by the user and the user will be able to click the order button to place the order. The program then calls back appropriate functions associated to the order button. The user is also required to input the drivers manually for the first time and click the add button to add the driver and then they are added automatically after they are done delivering.

- The output will be: The alerts that FLTK shows after the order is placed have the input values from the user. Once the order is cooked there is another alert that shows that order is cooked. Then the driver's name, the pizza, and the delivery address shows up when it is ready to deliver. There is another alert stating the driver is back from delivery once he or she is back from delivery. Along with the alerts, there is also an output of 3 Queues which are updated everytime something is removed or inserted. The program uses a function from the Queue class to show all the data in the FLTK Queue boxes. There is also a small clock which manages the time.

- The overall Algorithim is: The program uses call back functions to run. Ather the user inputs the drivers name, the user is required to press the add button which class the driver_cb function and adds the driver in the RBQueue. After the pizza and the address input the user is required to click the order button, which calls back the order_cb and puts the order into the LLQueue. After few seconds the cook_cb function is called which removes the order from the OrderQ and places it in the cookedQ. It then calls the delivery_cb which removes the driver and the cooked order and send it delivery. If there are no drivers the programs keeps checking for the drivers in the queue and sends the order once it is ready. the driver is added to a temp RBQ when he or she is out for delivery and then gets removed from the temp and added to the drivers Q when he or she comes back from delivery.

3 Design

- addBackDr_cb.cpp: adds the drivers back after the delivery
- cook_cb.cpp: cooks the order
- delivery_cb.cpp: sends the driver for delivery with the pizza. Both are removed from the Qs
- driver_cb: adds driver to the Queue
- getQcontent.cpp: gets a string will all the data in the Q
- getRQcontent.cpp: gets a string will all the data in the Q
- insert.cpp: inserts the data into the Q
- main.cpp: takes care of the FLTK design and call backs from the buttons
- remove.cpp: removes data from the Queue
- timer.cpp: manages the clock with time in FLTK window

4 Class Index

4.1 Class List

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

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 Class Documentation

6.1 LLQUEUE Class Reference

`#include <lab.h>`

Public Member Functions

- LLQUEUE ()

  constructor of the queue class which sets the value of the private functions to 0
- ~LLQUEUE ()

  destructor of the queue class which removes all the data from the queue
- bool Insert (ORDER &info)

  insert adds data into the queue
- bool Remove (ORDER &info)

  removed data from the queue
- bool isEmpty ()

  checks if the queue is empty
- std::string getQcontent ()

  gives the user all the data that is present in the queue

6.1.1 Constructor & Destructor Documentation

6.1.1.1 LLQUEUE::LLQUEUE ( ) [inline]

constructor of the queue class which sets the value of the private functions to 0

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

6.1.1.2 LLQUEUE::∼LLQUEUE ( )

destructor of the queue class which removes all the data from the queue

```
3 {
4     NODE *next;
5     while (front)
6     {
7         next= front->next;
8         delete front; //deletes the front
9         front = next; //sets the front to the next node
10         }
11     }
```

6.1.2 Member Function Documentation

6.1.2.1 std::string LLQUEUE::getQcontent ( )

gives the user all the data that is present in the queue

Returns

string containing all the data that is stored in the queue

```
4 {
5     std::string str;
6     NODE *node;
7         for(node = front; node; node = node->next)// sets node to the front, if node is not null, make
```

```
        node equal the the next node
8          {
9               str += node->info.info + "\n"; //stors the data in the list into str string one by
10
11               }
12      return str; //returns the whole str with all the data
13      }
```

6.1.2.2 bool LLQUEUE::Insert (ORDER & *info*)

insert adds data into the queue

Parameters

in	*info*	is the data that will be added into the queue

Returns

boolean indication if the data was inserted successfully

```
3 {
4      NODE *newnode = new NODE; //allocates a new node
5      if(!newnode) return false;
6      newnode->info = info;
7      newnode->next = 0;
8
9      if(rear == 0)//if rear is null
10      {
11          front = rear= newnode; //set front and rear to the new node
12          }
13      else
14      {
```

```
15          rear->next = newnode;
16          rear= newnode;
17          }
18
19 return true;
20 }
```

6.1.2.3 bool LLQUEUE::isEmpty ( ) [inline]

checks if the queue is empty

Returns

boolean indication if the queue is empty or not

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

6.1.2.4 bool LLQUEUE::Remove (ORDER & *info*)

removed data from the queue

Parameters

out	*info*	is the data that will be removed from the queue

Returns

boolean indication if the data was removed successfully

```
3 {
```

```
4      if (front == 0)
5      return false;
6
7      info = front->info; //sets info to the info being removed
8
9      NODE *next;
10      next = front->next;
11      delete front; //delets front
12      front = next; //front equals the next node
13      if (front ==0)
14      {
15          rear =0;
16          }
17      return true;
18      }
```

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

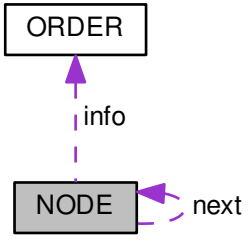
- lab.h
- getQcontent.cpp
- Insert.cpp
- llqueue.cpp
- Remove.cpp

6.2 NODE Struct Reference

struct NODE that will be used to make linked list

`#include <lab.h>`

Collaboration diagram for NODE:



Public Attributes

- ORDER info
- NODE * next

6.2.1 Detailed Description

struct NODE that will be used to make linked list

6.2.2 Member Data Documentation

6.2.2.1 ORDER NODE::info

6.2.2.2 NODE* NODE::next

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

- lab.h

6.3 ORDER Struct Reference

struct named ORDER which will have the name of the pizza and the address for delivery

`#include <lab.h>`

Public Attributes

- std::string info
- std::string addr

6.3.1 Detailed Description

struct named ORDER which will have the name of the pizza and the address for delivery

6.3.2 Member Data Documentation

6.3.2.1 std::string ORDER::addr

6.3.2.2 std::string ORDER::info

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

- lab.h

6.4 RBQUEUE Class Reference

```
#include <lab.h>
```

Public Member Functions

- RBQUEUE ()

  constructor of the ring buffer queue which sets the private data members to 0
- ~RBQUEUE ()

  destructor of the ring buffer queue which deletes all the data in the queue
- bool Insert (std::string s)

  insert adds data into the queue
- bool Remove (std::string &s)

  removed data from the queue
- bool isEmpty ()

  checks if the queue is empty
- bool isFull ()

  checks if the queue is full
- std::string getRQcontents ()

gives the user all the data that is present in the queue

6.4.1 Constructor & Destructor Documentation

6.4.1.1 RBQUEUE::RBQUEUE ( ) [inline]

constructor of the ring buffer queue which sets the private data members to 0

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

6.4.1.2 RBQUEUE::~RBQUEUE ( ) [inline]

destructor of the ring buffer queue which deletes all the data in the queue

```
143 {}
```

6.4.2 Member Function Documentation

6.4.2.1 std::string RBQUEUE::getRQcontents ( )

gives the user all the data that is present in the queue

Returns

string containing all the data that is stored in the queue

```
4 {
5     int curr;
6     std::string drivers;
```

```
7      for(int curr=front; curr<rear; curr=nextIndex(curr))//sets current to the front, if curr is less th
        rear, move current to the next index after front
8      {
9          drivers += buf[curr] + "\n"; //stores all the data into the string
10
11          }
12      return drivers; //returns the string with all the data
13      }
```

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

insert adds data into the queue

Parameters

in	*s*	is the data that will be added into the queue

Returns

boolean indication if the data was inserted successfully

```
4  {
5      if(isFull())// if the Q is full
6      {
7          return false;
8          }
9      buf[rear]= s; // inserts the data
10      rear = nextIndex(rear); // rear is the next available index
11      return true;
12      }
```

6.4.2.3 bool RBQUEUE::isEmpty ( ) [inline]

checks if the queue is empty

Returns

boolean indication if the queue is empty or not

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

6.4.2.4 bool RBQUEUE::isFull ( ) [inline]

checks if the queue is full

Returns

boolean indication if the queue is full or not

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

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

removed data from the queue

Parameters

out	*s*	is the data that will be removed from the queue

Returns

boolean indication if the data was removed successfully

```
4 {
5     if(isEmpty())
6     {
7         return false;
8         }
9    s = buf[front]; //sets s to the value being removed
10     front = nextIndex(front); // moves front to the next index
11     return true;
12     }
```

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

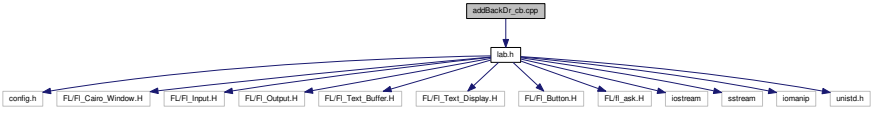
- lab.h
- getRQcontents.cpp
- insert.cpp
- remove.cpp

7 File Documentation

7.1 addBackDr_cb.cpp File Reference

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

Include dependency graph for addBackDr_cb.cpp:



Functions

- void addBackDr_cb (void *)

  This is the callback function called fron delivery which adds the driver back in the queue after they deliver the pizza void pointers not used return void.

7.1.1 Function Documentation

7.1.1.1 void addBackDr_cb (void *)

This is the callback function called fron delivery which adds the driver back in the queue after they deliver the pizza void pointers not used return void.

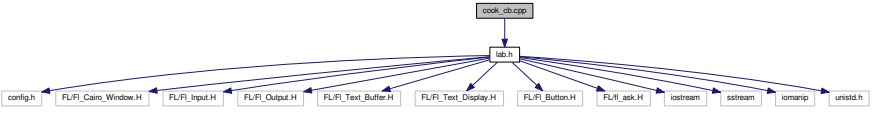
```
4 {
```

```
5     std::string Dback; //will be used to display alert
6     currD.Remove(Dback);//removes the driver from the queue of drivers who are currently
       delivering
7     d.Insert(Dback); //inserts the driver to the original driver queue who is ready to deliver
8     dbuff->text((d.getRQcontents().c_str())); //updates the text in FLTK
9     std::string alrt; //string that will help display alert
10     alrt = Dback + " is back from delivery";
11     fl_alert(alrt.c_str()); //shows alert when the driver is back
12     }
```

7.2 cook_cb.cpp File Reference

`#include "lab.h"`

Include dependency graph for cook_cb.cpp:



Functions

- void cook_cb (void *)

  This is the callback function called from the order_cb function which cooks the pizza void pointer not used return void.

7.2.1 Function Documentation

7.2.1.1 void cook_cb (void *)

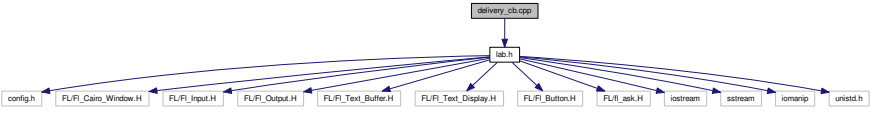
This is the callback function called from the order_cb function which cooks the pizza void pointer not used return void.

```
3 {
4     o.Remove(ord); //removes the data from the order queue
5     buff->text((o.getQcontent().c_str()));//updates the data for FLTK in the order Q
6     c.Insert(ord); //inserts the data into the cooked Q
7
8     cbuff->text((c.getQcontent().c_str())); //Updates the FLTK text box for cooked Q
9
10     std::string alrt = ord.info + " is cooked";
11     fl_alert(alrt.c_str()); //shows alert that the pizza is cooked
12
13     Fl::add_timeout(3,delivery_cb); //calls back the delivery_cb function with 3 sec delay for
       us to actually see whats happening
14
15 }
```

7.3 delivery_cb.cpp File Reference

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

Include dependency graph for delivery_cb.cpp:



Functions

- void delivery_cb (void *)

  This is the callback function called from the cook_cb function which assigns and send drivers out for delivery after the pizza is cooked void pointers not used return void.

7.3.1 Function Documentation

7.3.1.1 void delivery_cb (void *)

This is the callback function called from the cook_cb function which assigns and send drivers out for delivery after the pizza is cooked void pointers not used return void.

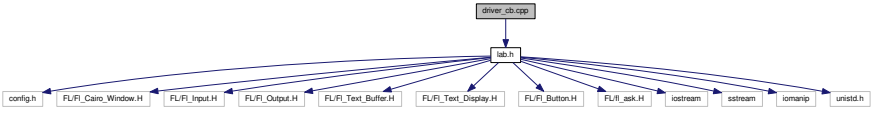
```
4 {
5
6       if(d.isEmpty()) //checks if the drivers Q is empty
```

```
7     {
8        fl_alert("Not enough drivers"); //shows alerts that there are no more drivers
9        Fl::add_timeout(5,delivery_cb); //checks again if there are any drivers in the Q after 5
      sec
10        }
11     else
12     {
13        std::string dname;
14        d.Remove(dname);//removes the driver who is ready for delivery
15        currD.Insert(dname); //puts the same driver in a different RBQ which is for the driver
      who are out for delivery
16        c.Remove(ord);//removes the pizza from the cooked Q
17        std::string temp = dname + " is delivering the "+ ord.info + " to " +
     ord.addr;
18        fl_alert(temp.c_str());//alert that shows who is delivering the pizaa and where
19        cbuff->text((c.getQcontent().c_str())); // updates FLTK text in cooked Q box
20        dbuff->text((d.getRQcontents().c_str())); // updates FLTK text in drivers Q box
21        Fl::add_timeout(100,addBackDr_cb); //cb to the function which adds the drivers back
      after they do delivery
22        }
23
24     }
```

7.4 driver_cb.cpp File Reference

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

Include dependency graph for driver_cb.cpp:



Functions

- void driver_cb (void *)

  This is the callback function to add drivers manually void pointers not used return void.

7.4.1 Function Documentation

7.4.1.1 void driver_cb (void *)

This is the callback function to add drivers manually void pointers not used return void.

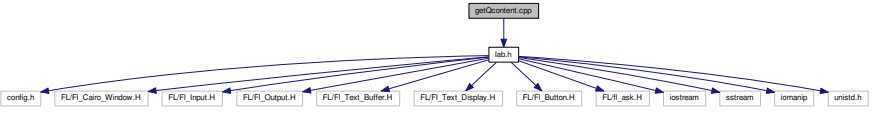
```
4 {
5    fl_alert(driver->value()); //shows alert for the driver who is being added by the user
6    std::string name;
7    name = driver->value();
8      d.Insert(name); //inserts the driver in the driversQ
```

```
9       dbuff->text((d.getRQcontents().c_str())); // updates the FLTK driversQ box
10       }
```

7.5 getQcontent.cpp File Reference

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

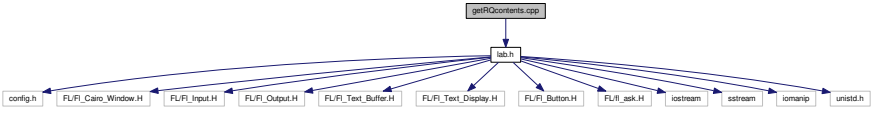
Include dependency graph for getQcontent.cpp:



7.6 getRQcontents.cpp File Reference

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

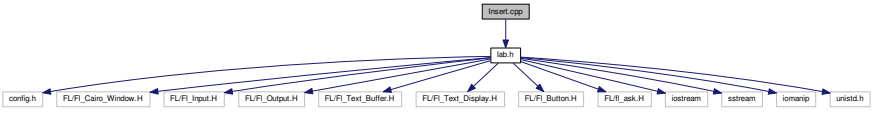
Include dependency graph for getRQcontents.cpp:



7.7 Insert.cpp File Reference

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

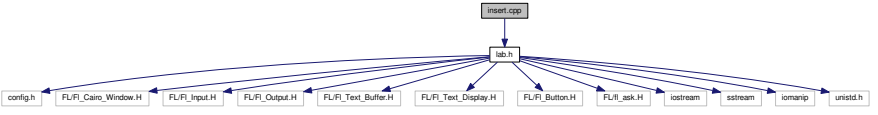
Include dependency graph for Insert.cpp:



7.8 insert.cpp File Reference

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

Include dependency graph for insert.cpp:



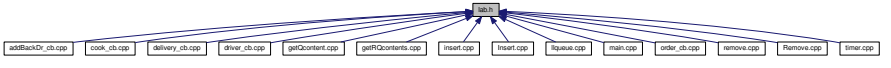
7.9 lab.dox File Reference

7.10 lab.h File Reference

```
#include "config.h"
#include <FL/Fl_Cairo_Window.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_Text_Buffer.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Button.H>
#include <FL/fl_ask.H>
#include <iostream>
#include <sstream>
#include <iomanip>
#include <unistd.h>
```

Include dependency graph for lab.h:

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



Classes

- struct ORDER

  struct named ORDER which will have the name of the pizza and the address for delivery
- struct NODE

  struct NODE that will be used to make linked list
- class LLQUEUE
- class RBQUEUE

Functions

- void order_cb (void *, void *)

  This is the callback function to order a pizza void pointers not used return void.
- void driver_cb (void *)

  This is the callback function to add drivers manually void pointers not used return void.
- void cook_cb (void *)

  This is the callback function called from the order_cb function which cooks the pizza void pointer not used return void.
- void delivery_cb (void *)

  This is the callback function called from the cook_cb function which assigns and send drivers out for delivery after the pizza is cooked void pointers not used return void.
- void addBackDr_cb (void *)

  This is the callback function called fron delivery which adds the driver back in the queue after they deliver the pizza void pointers not used return void.
- void timer (void *)

  This is a time function which keeps track of the time void pointers not used return void.

Variables

- Fl_Input * pizza
- Fl_Input * address
- Fl_Input * driver
- Fl_Output * watch
- Fl_Text_Buffer * buff
- Fl_Text_Buffer * cbuff
- Fl_Text_Buffer * dbuff
- Fl_Text_Display * driverQ
- Fl_Text_Display * orderQ
- Fl_Text_Display * cookedQ
- const int BUFSIZE = 256
- ORDER ord
- LLQUEUE o
- LLQUEUE c
- RBQUEUE d
- RBQUEUE currD

7.10.1 Function Documentation

7.10.1.1 void addBackDr_cb (void *)

This is the callback function called fron delivery which adds the driver back in the queue after they deliver the pizza void pointers not used return void.

```
4 {
5     std::string Dback; //will be used to display alert
```

```
6      currD.Remove(Dback);//removes the driver from the queue of drivers who are currently
        delivering
7      d.Insert(Dback); //inserts the driver to the original driver queue who is ready to deliver
8      dbuff->text((d.getRQcontents().c_str())); //updates the text in FLTK
9      std::string alrt; //string that will help display alert
10      alrt = Dback + " is back from delivery";
11      fl_alert(alrt.c_str()); //shows alert when the driver is back
12      }
```

7.10.1.2 void cook_cb (void *)

This is the callback function called from the order_cb function which cooks the pizza void pointer not used return void.

```
3 {
4      o.Remove(ord); //removes the data from the order queue
5      buff->text((o.getQcontent().c_str()));//updates the data for FLTK in the order Q
6      c.Insert(ord); //inserts the data into the cooked Q
7
8      cbuff->text((c.getQcontent().c_str())); //Updates the FLTK text box for cooked Q
9
10      std::string alrt = ord.info + " is cooked";
11      fl_alert(alrt.c_str()); //shows alert that the pizza is cooked
12
13      Fl::add_timeout(3,delivery_cb); //calls back the delivery_cb function with 3 sec delay for
        us to actually see whats happening
14
15 }
```

7.10.1.3 void delivery_cb (void *)

This is the callback function called from the cook_cb function which assigns and send drivers out for delivery after the pizza is cooked void pointers not used return void.

```
4 {
5
6      if(d.isEmpty()) //checks if the drivers Q is empty
7     {
8        fl_alert("Not enough drivers"); //shows alerts that there are no more drivers
9        Fl::add_timeout(5,delivery_cb); //checks again if there are any drivers in the Q after 5
       sec
10         }
11     else
12     {
13         std::string dname;
14         d.Remove(dname);//removes the driver who is ready for delivery
15         currD.Insert(dname); //puts the same driver in a different RBQ which is for the driver
       who are out for delivery
16         c.Remove(ord);//removes the pizza from the cooked Q
17         std::string temp = dname + " is delivering the "+ ord.info + " to " +
      ord.addr;
18         fl_alert(temp.c_str());//alert that shows who is delivering the pizaa and where
19         cbuff->text((c.getQcontent().c_str())); // updates FLTK text in cooked Q box
20         dbuff->text((d.getRQcontents().c_str())); // updates FLTK text in drivers Q box
21         Fl::add_timeout(100,addBackDr_cb); //cb to the function which adds the drivers back
       after they do delivery
22         }
23
24     }
```

7.10.1.4 void driver_cb (void *)

This is the callback function to add drivers manually void pointers not used return void.

```
4 {
5    fl_alert(driver->value()); //shows alert for the driver who is being added by the user
6    std::string name;
7    name = driver->value();
8      d.Insert(name); //inserts the driver in the driversQ
9     dbuff->text((d.getRQcontents().c_str())); // updates the FLTK driversQ box
10     }
```

7.10.1.5 void order_cb (void *, void *)

This is the callback function to order a pizza void pointers not used return void.

```
9 {
10     fl_alert(pizza->value()); //shows alert with input values
11     fl_alert(address->value());
12
13     ord.info = pizza->value();
14     ord.addr = address->value();
15
16     o.Insert(ord);//inserts the data into order Q
17
18     buff->text((o.getQcontent().c_str())); // updates th order Q in FLTK
19
20     Fl::add_timeout(10,cook_cb); //calls back the cook function
21     }
```

7.10.1.6 void timer (void *)

This is a time function which keeps track of the time void pointers not used return void.

```
4 {
5     //std::cout << "1 sec" << std::endl;
6     static int s = 0; static int m = 0;
7     std::ostringstream oss;
8
9     s++; if (s == 59) {s=0; m++;}
10     oss << std::setfill('0');
11     oss << std::setw(2) << m << ":" << std::setw(2) << s;
12     watch->value(oss.str().c_str());
13
14     Fl::repeat_timeout(1,timer); //repeats after every second
15
16 }
```

7.10.2 Variable Documentation

7.10.2.1 Fl_Input* address

7.10.2.2 Fl_Text_Buffer* buff

7.10.2.3 const int BUFSIZE = 256

7.10.2.4 LLQUEUE c

7.10.2.5 Fl_Text_Buffer* cbuff

7.10.2.6 Fl_Text_Display* cookedQ

7.10.2.7 RBQUEUE currD

7.10.2.8 RBQUEUE d

7.10.2.9 Fl_Text_Buffer* dbuff

7.10.2.10 Fl_Input* driver

7.10.2.11 Fl_Text_Display* driverQ

7.10.2.12 LLQUEUE o

7.10.2.13 ORDER ord

7.10.2.14 Fl_Text_Display* orderQ

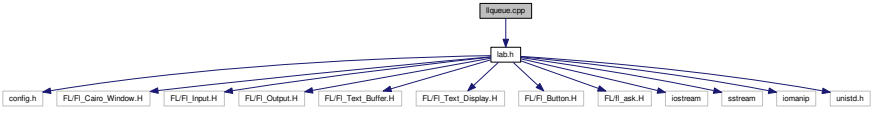
7.10.2.15 Fl_Input* pizza

7.10.2.16 Fl_Output* watch

7.11 llqueue.cpp File Reference

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

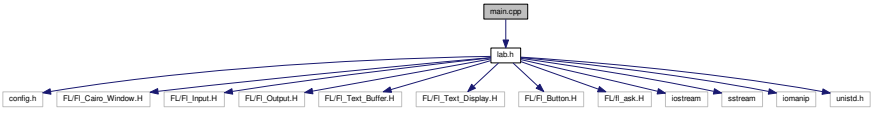
Include dependency graph for llqueue.cpp:



7.12 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 * driver
- Fl_Output * watch
- Fl_Text_Buffer * buff
- Fl_Text_Buffer * cbuff
- Fl_Text_Buffer * dbuff
- Fl_Text_Display * orderQ
- Fl_Text_Display * cookedQ
- Fl_Text_Display * driverQ

7.12.1 Function Documentation

7.12.1.1 int main ( )

```
18 {
19     Fl_Cairo_Window cw(400,300); //width and height of the window
20     cw.label("Pizza Deliveries");//title of your window
21     //cw.color(FL_BLUE);
22 
23     pizza = new Fl_Input(180,20,100,20, "Pizza:");
24     pizza->labelcolor(FL_BLUE);
25 
26     address = new Fl_Input(180,40,100,20, "Address:");
27     address->labelcolor(FL_BLUE);
28 
29     driver = new Fl_Input(180,60,100,20, "Driver:");
30     driver->labelcolor(FL_BLUE);
```

```
31
32      watch = new Fl_Output(70,20,60,20, "seconds:");
33
34      buff = new Fl_Text_Buffer();
35      orderQ = new Fl_Text_Display(50,100,100,100,"Order Q");
36      orderQ->buffer(buff);
37
38      cbuff = new Fl_Text_Buffer();
39      cookedQ = new Fl_Text_Display(150,100,100,100,"Cooked Q");
40      cookedQ->buffer(cbuff);
41
42      dbuff = new Fl_Text_Buffer();
43      driverQ = new Fl_Text_Display(250,100,100,100,"Driver Q");
44      driverQ->buffer(dbuff);
45
46      Fl_Button b(280,35,50,20, "Order");
47      b.callback((Fl_Callback*)order_cb);
48
49      Fl_Button a(280,60,50,20, "Add");
50      a.callback((Fl_Callback*)driver_cb);
51      cw.show();
52      Fl::add_timeout(1,timer);
53      return Fl::run();
54      }
```

7.12.2 Variable Documentation

7.12.2.1 Fl_Input* address

7.12.2.2 Fl_Text_Buffer* buff

7.12.2.3 Fl_Text_Buffer* cbuff

7.12.2.4 Fl_Text_Display* cookedQ

7.12.2.5 Fl_Text_Buffer* dbuff

7.12.2.6 Fl_Input* driver

7.12.2.7 Fl_Text_Display* driverQ

7.12.2.8 Fl_Text_Display* orderQ

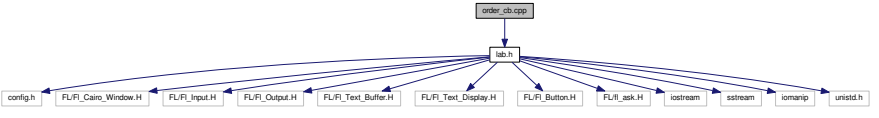
7.12.2.9 Fl_Input* pizza

7.12.2.10 Fl_Output* watch

7.13 order_cb.cpp File Reference

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

Include dependency graph for order_cb.cpp:



Functions

- void order_cb (void *, void *)

  This is the callback function to order a pizza void pointers not used return void.

Variables

- ORDER ord
- LLQUEUE o
- LLQUEUE c
- RBQUEUE d
- RBQUEUE currD

7.13.1 Function Documentation

7.13.1.1 void order_cb (void * , void *)

This is the callback function to order a pizza void pointers not used return void.

```
9 {
10      fl_alert(pizza->value()); //shows alert with input values
11      fl_alert(address->value());
12
13      ord.info = pizza->value();
14      ord.addr = address->value();
15
16      o.Insert(ord);//inserts the data into order Q
17
18      buff->text((o.getQcontent().c_str())); // updates th order Q in FLTK
```

```
19
20      Fl::add_timeout(10,cook_cb); //calls back the cook function
21      }
```

7.13.2 Variable Documentation

7.13.2.1 LLQUEUE c

7.13.2.2 RBQUEUE currD

7.13.2.3 RBQUEUE d

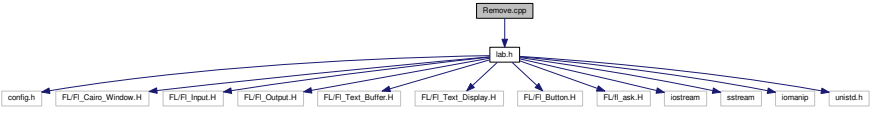
7.13.2.4 LLQUEUE o

7.13.2.5 ORDER ord

7.14 Remove.cpp File Reference

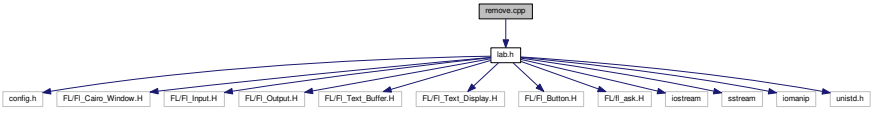
```
#include "lab.h"
```

Include dependency graph for Remove.cpp:



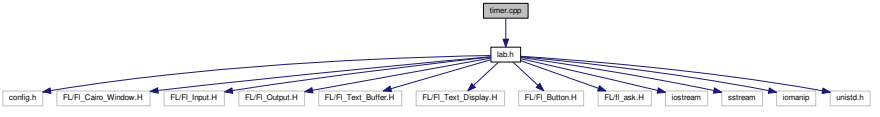
7.15 remove.cpp File Reference

`#include "lab.h"`
Include dependency graph for remove.cpp:



7.16 timer.cpp File Reference

`#include "lab.h"`
Include dependency graph for timer.cpp:



Functions

- void timer (void ∗)

  This is a time function which keeps track of the time void pointers not used return void.

7.16.1 Function Documentation

7.16.1.1 void timer (void *)

This is a time function which keeps track of the time void pointers not used return void.

```
4 {
5     //std::cout << "1 sec" << std::endl;
6     static int s = 0; static int m = 0;
7     std::ostringstream oss;
8
9     s++; if (s == 59) {s=0; m++;}
10     oss << std::setfill('0');
11     oss << std::setw(2) << m << ":" << std::setw(2) << s;
12     watch->value(oss.str().c_str());
13
14     Fl::repeat_timeout(1,timer); //repeats after every second
15
16 }
```