Lab4

Generated by Doxygen 1.8.8

Sat Oct 21 2017 12:31:38

Contents

1 Specification

This is the Pizza Shop Queue Program. It uses queues which are a data stucture for storing certain elements. It uses a last in first out method like how a normal queue operates. Elements are added at the end and removed at the front. This is usefull for a variety of reason namely restaurants; where orders are important.

Features:

1) A user can enter pizzas and have it show up in the queue

2) The queue shows everything in order along with the address.

3) Done witha a GUI so the user has ease of navigation

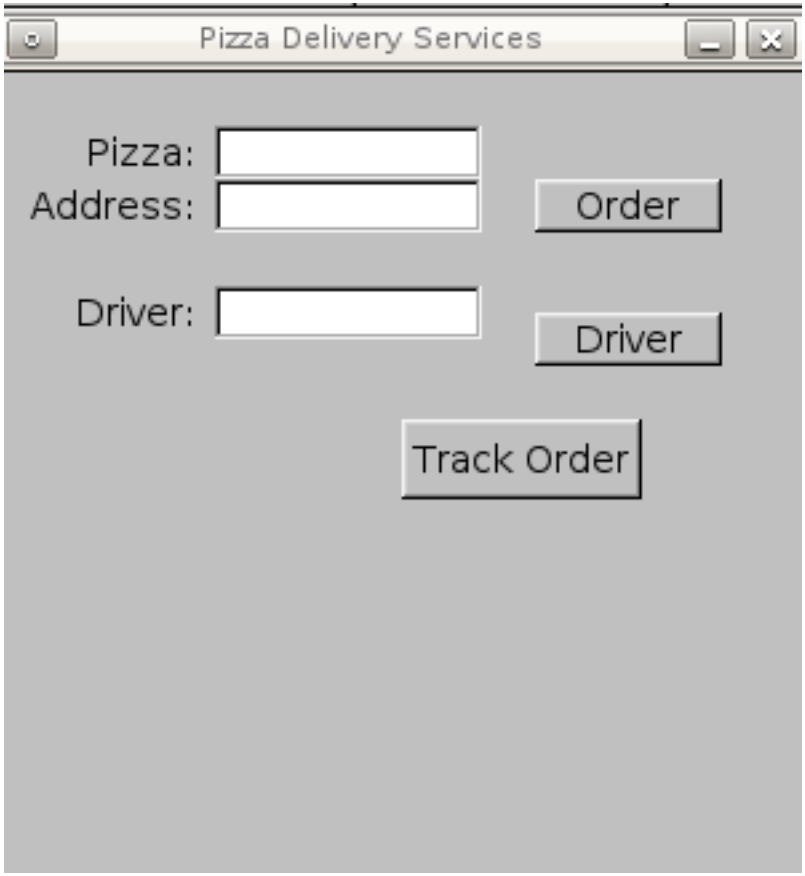
2 Analysis

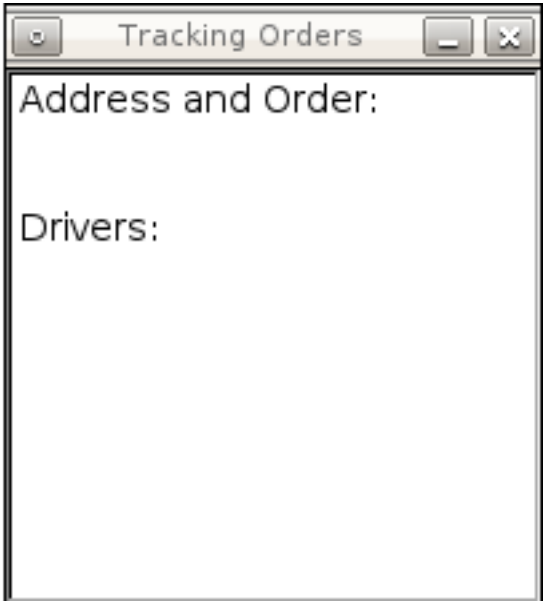
When the program runs a window will pop up. This is a good sign that the program is working. Then 3 selection criteria will open up. First enter the pizza and then enter the address. Then click order wich will ad that pizza into the queue. Then drivers can be entered. Once they are entered they begin to fill up the queue. They will then take the pizza and deliver it which will remove them from the queue.

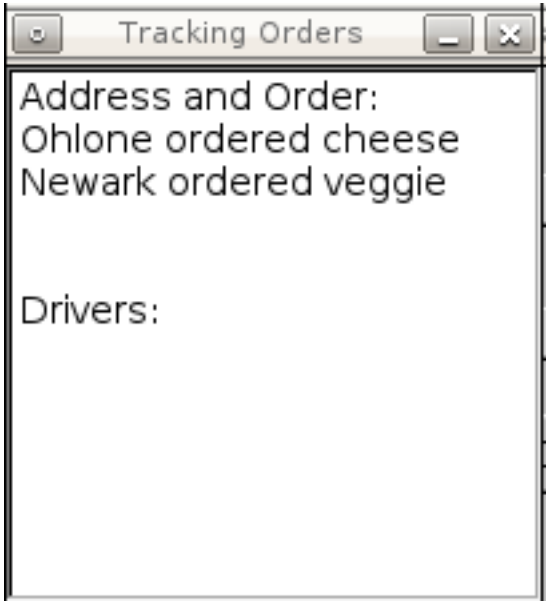
3 Design

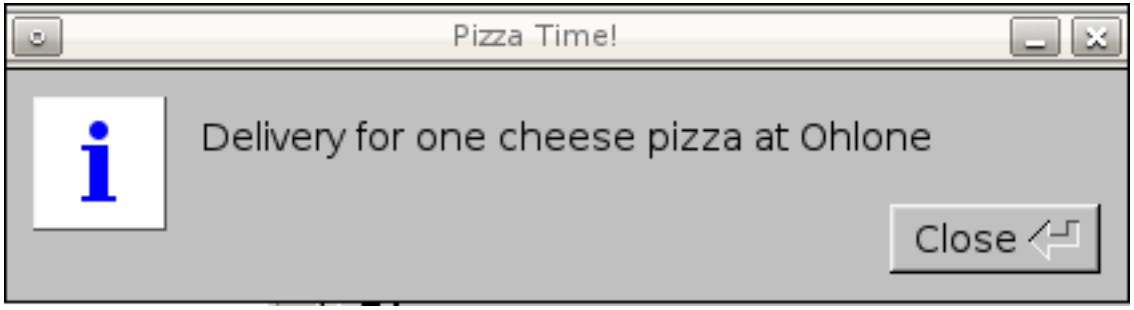
The Lab is designed using Fltk. This is what makes the GUI. The intention of showing the que on a seperate window was to allow the user to easily see the ques and the drivers and what actions were being taken on the pizzas Each file has a specific purpose see the individual slides to understand more. One of the design changes was the timer. There was a timer initially added into the program to show the time since launch. However since none of the queue information could be live updated (only updates when something was added or manually changed) I scrapped it since it would cause confusion. For example a pizza takes 10 seconds to cook. The pizza would do that but would not update on the GUI until the user entered something new. So it was confusing to exepect the GUI to update after 10 seconds. Now after the timer is removed the user can instead click a button which will show them the queue. The timer was largely irelevant to the program since it just counted how long it was running and not how long a pizza was cooking or being delivered. Instead a standard 10 seconds would be allotted to each pizza before the alert to have it deliver was alotted.

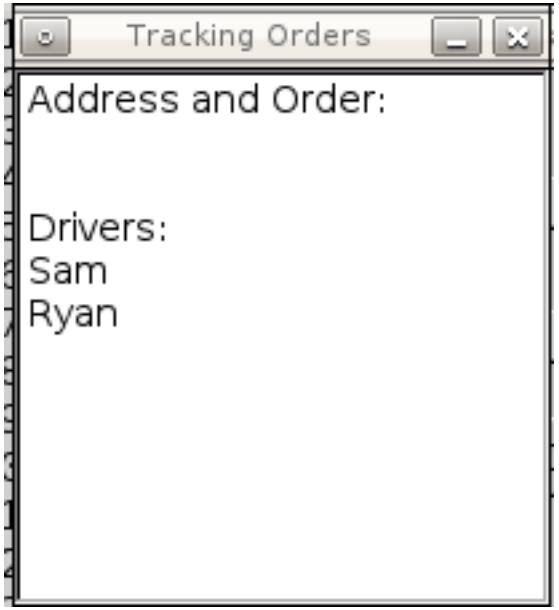
4 Test

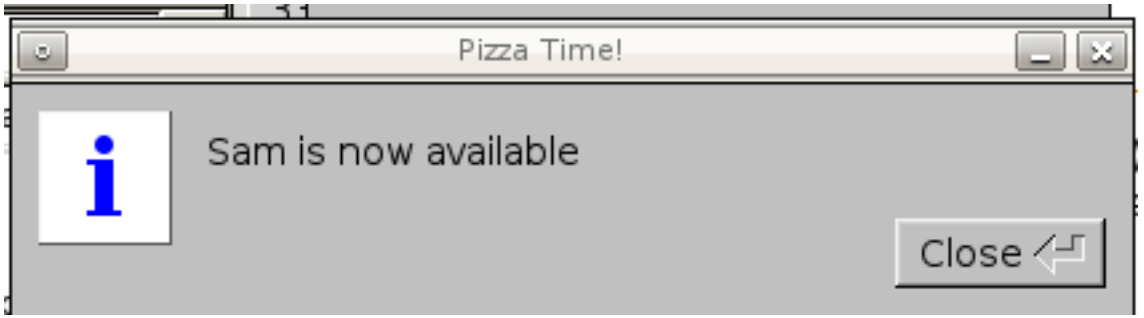

Pizza Delivery Services
Pizza:
Address:
Order
Driver:
Driver
Track Order

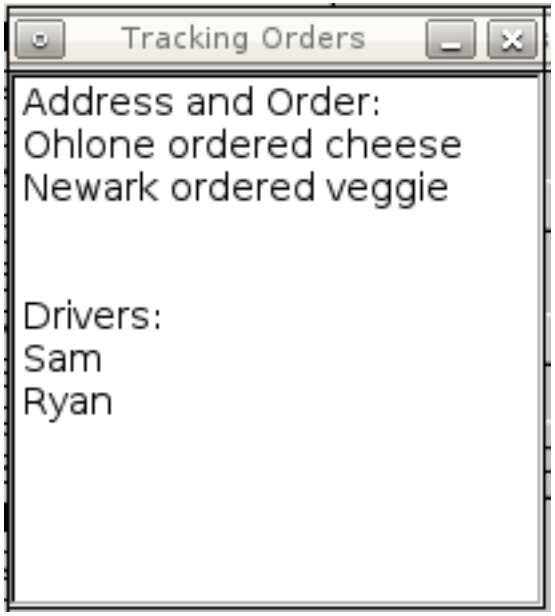

Tracking Orders
Address and Order:
Drivers:

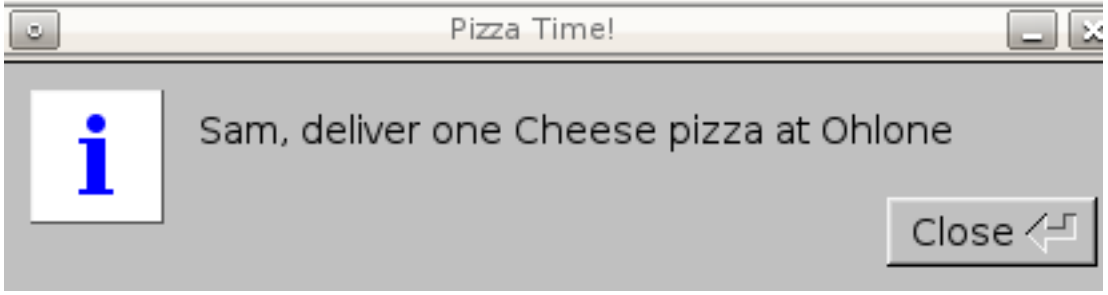

Tracking Orders
Address and Order:
Ohlone ordered cheese
Newark ordered veggie
Drivers:


Pizza Time!
Delivery for one cheese pizza at Ohlone
Close


Tracking Orders
Address and Order:
Drivers:
Sam
Ryan


Pizza Time!
Sam is now available
Close


Tracking Orders
Address and Order:
Ohlone ordered cheese
Newark ordered veggie
Drivers:
Sam
Ryan


Pizza Time!
Sam, deliver one Cheese pizza at Ohlone
Close

5 Class Index

5.1 Class List

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

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Class Documentation

7.1 LLQUEUE Class Reference

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

Public Member Functions

- LLQUEUE ()
- ~LLQUEUE ()
- bool Insert (ORDER &info)
- bool Remove (ORDER &info)
- bool isEmpty ()
- string traverse (ORDER &info)

7.1.1 Constructor & Destructor Documentation

7.1.1.1 LLQUEUE::LLQUEUE ( ) [inline]

```
47 {front = rear = 0;} //Initialize the pointers to null
```

7.1.1.2 LLQUEUE::~LLQUEUE ( ) [inline]

```
48                    { //destructor (default)
49             NODE *next;
50
51             while (front) {
```

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

7.1.2 Member Function Documentation

7.1.2.1 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;
     }
     cout << "insert " << order.items;
     return true;
}
```

7.1.2.2 bool LLQUEUE::isEmpty ( ) [inline]

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

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

```
{
    if (front == 0)
        return false;
    //Get the first element out of the que
    info = front -> info;

     //Remove the node from the front of the queue
     NODE *next = front -> next;

     delete front;
     front = next;
     if (front == 0) //if the last element was removed
         rear = 0;

     cout << "remove " << order.items;
     return true;
}
```

7.1.2.4 string LLQUEUE::traverse (ORDER & *info*)

This is the code to traverse the queues an build a list that can be put into the GUI. This function returns the list. A key note is that it is type string so that we can call it and use the return value.

```
{
    string list = "Address and Order: \n";
```

```
    for(NODE *p = front; p; p = p->next)
    {
        list+=p->info.address; //
        list+=" ordered ";
        list+= p->info.items;
        list+= "\n";

    }
    return list;

}
```

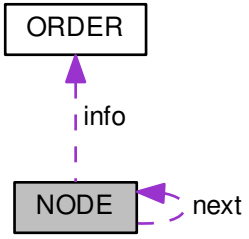
The documentation for this class was generated from the following files:

- lab.h
- insert.cpp
- remove.cpp
- traverse.cpp

7.2 NODE Struct Reference

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

Collaboration diagram for NODE:



Public Attributes

- ORDER info
- NODE * next

7.2.1 Member Data Documentation

7.2.1.1 ORDER NODE::info

7.2.1.2 NODE* NODE::next

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

- lab.h

7.3 ORDER Struct Reference

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

Public Attributes

- string address
- string items

7.3.1 Member Data Documentation

7.3.1.1 string ORDER::address

7.3.1.2 string ORDER::items

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

- lab.h

7.4 RBQUEUE Class Reference

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

Public Member Functions

- RBQUEUE ()
- ~RBQUEUE ()
- bool Insert (string s)
- bool Remove (string &s)
- bool isEmpty ()
- bool isFull ()
- string traverse ()

7.4.1 Constructor & Destructor Documentation

7.4.1.1 RBQUEUE::RBQUEUE ( ) [inline]

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

7.4.1.2 RBQUEUE::~RBQUEUE ( ) [inline]

```
{}
```

7.4.2 Member Function Documentation

7.4.2.1 bool RBQUEUE::Insert (string *s*)

```
{
    if (isFull()) return false;
    buf[rear] = s;
    rear = nextIndex(rear);
```

```
    return true;
     cout << "insert " << s;
}
```

7.4.2.2 bool RBQUEUE::isEmpty ( ) [inline]

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

7.4.2.3 bool RBQUEUE::isFull ( ) [inline]

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

7.4.2.4 bool RBQUEUE::Remove (string & *s*)

```
{
    if (isEmpty()) return false;
    s = buf[front];
    front = nextIndex(front);
    cout << "remove " << s;
    return true;

}
```

7.4.2.5 string RBQUEUE::traverse ( )

```
{
    string list = "\n\nDrivers: \n";
    for (int i = front; i!=rear; i++)
    {
        list += buf[i];
```

```
        list += "\n";
    }
    return list;
}
```

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

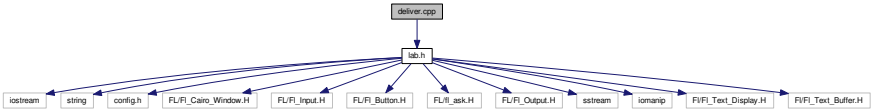
- lab.h
- RBinsert.cpp
- RBremove.cpp
- RBtraverse.cpp

8 File Documentation

8.1 deliver.cpp File Reference

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

Include dependency graph for deliver.cpp:



Functions

- void deliver (void *)

8.1.1 Function Documentation

8.1.1.1 void deliver (void *)

This function will put out alerts when drivers or Orders are ready See the comments for more details. The message displayed is based on which of the queues are empty. This wee then cause the message to be displayed.

```
10 {
11
```

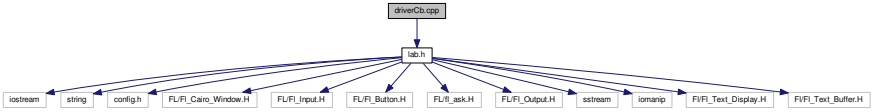
```
    string driverName;


    if(!pendingOrder.isEmpty() && !drivers.isEmpty())
    {
        drivers.Remove(driverName);
        pendingOrder.Remove(order);

        string alert = driverName + ", deliver one " + order.items
        + " pizza at " + order.address; // create the string for the alert

        cout << alert << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert.c_str()); //add in the message
        Fl::repeat_timeout(5.0,deliver); //display it

    }

    else if (!pendingOrder.isEmpty() && drivers.
isEmpty())
    {

        string alert1 ="Delivery for one " + order.items
        + " pizza at " + order.address; //Create the string for the message

        cout << alert1 << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert1.c_str()); // Add the message into the alert
        Fl::repeat_timeout(5.0,deliver); //display it

```

```
    }

     else if (pendingOrder.isEmpty() && !drivers.isEmpty())
    {

        string alert2 = driverName + " is now available";

        cout << alert2 << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert2.c_str());
        Fl::repeat_timeout(5.0,deliver);



    }


}
```

8.2 driverCb.cpp File Reference

`#include "lab.h"`

Include dependency graph for driverCb.cpp:



Functions

- void driverCb (Fl_Callback ∗, void ∗)

8.2.1 Function Documentation

8.2.1.1 void driverCb (Fl_Callback ∗ , void ∗)

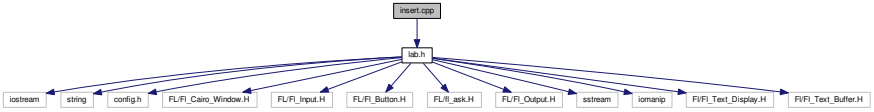
This is the driver callback function which is similar to the order one This inserts the drivers into the que and allows it to be displayed.

```
{
    drivers.Insert(Driver->value());
}
```

8.3 insert.cpp File Reference

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

Include dependency graph for insert.cpp:

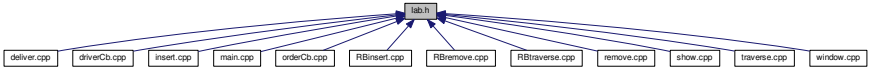


8.4 lab.h File Reference

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

Include dependency graph for lab.h:

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



Classes

- struct ORDER
- struct NODE
- class LLQUEUE
- class RBQUEUE

Functions

- void orderCb (Fl_Callback, void ∗)
- void driverCb (Fl_Callback, void ∗)
- void showQ (Fl_Callback ∗, void ∗)

- void deliver (void ∗)
- Fl_Cairo_Window ∗ window ()

Variables

- const int w = 300
- const int h = 300
- const int BUFSIZE = 10
- Fl_Input ∗ pizza
- Fl_Input ∗ address
- Fl_Input ∗ Driver
- Fl_Output ∗ watch
- Fl_Text_Buffer ∗ buff
- Fl_Text_Display ∗ orderQ
- ORDER order
- LLQUEUE pendingOrder
- RBQUEUE drivers

8.4.1 Function Documentation

8.4.1.1 void deliver (void ∗)

This function will put out alerts when drivers or Orders are ready See the comments for more details. The message displayed is based on which of the queues are empty. This wee then cause the message to be displayed.

```
{

    string driverName;
```

```


    if(!pendingOrder.isEmpty() && !drivers.isEmpty())
    {
        drivers.Remove(driverName);
        pendingOrder.Remove(order);

        string alert = driverName + ", deliver one " + order.items
        + " pizza at " + order.address; // create the string for the alert

        cout << alert << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert.c_str()); //add in the message
        Fl::repeat_timeout(5.0,deliver); //display it

    }

    else if (!pendingOrder.isEmpty() && drivers.
   isEmpty())
    {

        string alert1 ="Delivery for one " + order.items
        + " pizza at " + order.address; //Create the string for the message

        cout << alert1 << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert1.c_str()); // Add the message into the alert
        Fl::repeat_timeout(5.0,deliver); //display it


```

```

    }

     else if (pendingOrder.isEmpty() && !drivers.isEmpty())
    {

        string alert2 = driverName + " is now available";

        cout << alert2 << endl;

        fl_message_title("Pizza Time!");
        fl_message(alert2.c_str());
        Fl::repeat_timeout(5.0,deliver);



    }


}
```

8.4.1.2 void driverCb (Fl_Callback , void *)

8.4.1.3 void orderCb (Fl_Callback , void *)

8.4.1.4 void showQ (Fl_Callback * , void *)

This function shows the the queues when the user presses Track order in the GUI. This function shows the addresses and the pizza as well as the drivers that are available.

```
{
```

```
    string orderlist;
     string driverlist;

     static Fl_Cairo_Window trackWindow(200, 200); //Build the window
     trackWindow.label("Tracking Orders");
     static Fl_Text_Buffer buff;
     static Fl_Text_Display OrderQ(0,0, 200,200, "Track Order:");
     OrderQ.buffer(&buff);

     string o = pendingOrder.traverse(order); //using the traverse fucntion to creat the list
     string d = drivers.traverse();
     o+=d;//This creates the list of Orders and Drivers

     buff.text(o.c_str());
     trackWindow.add(OrderQ);
     trackWindow.show(); //Displays the window
}
```

8.4.1.5 Fl_Cairo_Window* window ( )

```
{
     cw = new Fl_Cairo_Window(w,h);

     cw->label("Pizza Delivery Services");

     cw->color(FL_GRAY);

     Order = new Fl_Button(200,40,70,20,"Order");
     Order->callback((Fl_Callback*)orderCb);

     driver = new Fl_Button(200,90,70,20,"Driver");
```

```
    driver->callback((Fl_Callback*)driverCb);

    tracker = new Fl_Button (150,130,90,30,"Track Order");
    tracker->callback((Fl_Callback*)showQ);

    pizza = new Fl_Input(80,20,100,20,"Pizza: ");
    pizza -> color(FL_WHITE);

    address = new Fl_Input(80,40,100,20, "Address: ");
    address-> color(FL_WHITE);

    Driver = new Fl_Input(80,80,100,20, "Driver: ");
    Driver-> color(FL_WHITE);

    return cw;


}
```

8.4.2 Variable Documentation

8.4.2.1 Fl_Input* address

8.4.2.2 Fl_Text_Buffer* buff

8.4.2.3 const int BUFSIZE = 10

8.4.2.4 Fl_Input* Driver

8.4.2.5 RBQUEUE drivers

8.4.2.6 const int h = 300

8.4.2.7 ORDER order

This is the main function which creates the objects for the lists and structs It also passes control to Fltk and runs the Fltk GUI. The code of the GUI can be seen in window.cpp

8.4.2.8 Fl_Text_Display∗ orderQ

8.4.2.9 LLQUEUE pendingOrder

8.4.2.10 Fl_Input∗ pizza

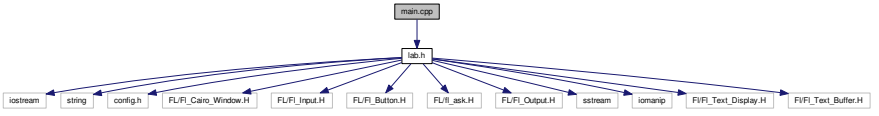
8.4.2.11 const int w = 300

8.4.2.12 Fl_Output∗ watch

8.5 main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

- int main ()

Variables

- ORDER order
- LLQUEUE pendingOrder
- RBQUEUE drivers

8.5.1 Function Documentation

8.5.1.1 int main ( )

```
{
    window()->show();
    Fl::add_timeout(20, deliver);
    return Fl::run();
}
```

8.5.2 Variable Documentation

8.5.2.1 RBQUEUE drivers

8.5.2.2 ORDER order

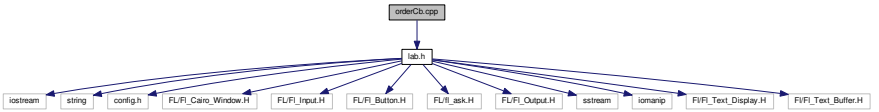
This is the main function which creates the objects for the lists and structs It also passes control to Fltk and runs the Fltk GUI. The code of the GUI can be seen in window.cpp

8.5.2.3 LLQUEUE pendingOrder

8.6 orderCb.cpp File Reference

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

Include dependency graph for orderCb.cpp:



Functions

- void orderCb (Fl_Callback ∗, void ∗)

8.6.1 Function Documentation

8.6.1.1 void orderCb (Fl_Callback ∗ , void ∗)

This the the order fucntion. This adds the ordered elements into the list.

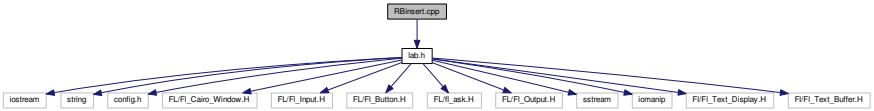
```
{
    order.address = address->value();
    order.items = pizza->value();
     pendingOrder.Insert(order); //Where the data is inserted
```

```
}
```

8.7 RBinsert.cpp File Reference

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

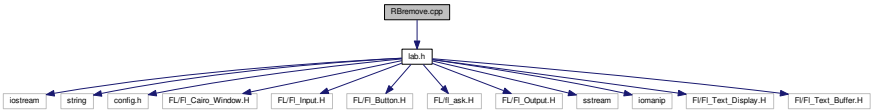
Include dependency graph for RBinsert.cpp:



8.8 RBremove.cpp File Reference

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

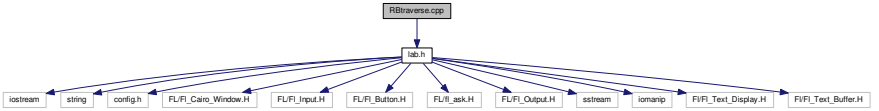
Include dependency graph for RBremove.cpp:



8.9 RBtraverse.cpp File Reference

`#include "lab.h"`

Include dependency graph for RBtraverse.cpp:



8.10 remove.cpp File Reference

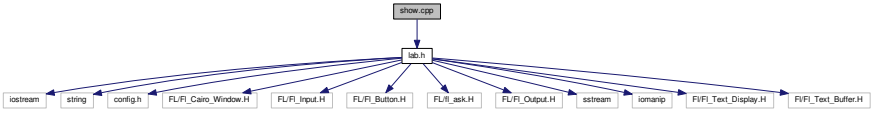
`#include "lab.h"`

Include dependency graph for remove.cpp:

8.11 show.cpp File Reference

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

Include dependency graph for show.cpp:



Functions

- void showQ (Fl_Callback *, void *)

8.11.1 Function Documentation

8.11.1.1 void showQ (Fl_Callback *, void *)

This function shows the the queues when the user presses Track order in the GUI. This function shows the addresses and the pizza as well as the drivers that are available.

```
{
    string orderlist;
     string driverlist;

     static Fl_Cairo_Window trackWindow(200, 200); //Build the window
     trackWindow.label("Tracking Orders");
```

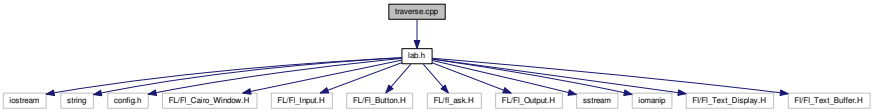
```
    static Fl_Text_Buffer buff;
    static Fl_Text_Display OrderQ(0,0, 200,200, "Track Order:");
    OrderQ.buffer(&buff);

    string o = pendingOrder.traverse(order); //using the traverse fucntion to creat the list
    string d = drivers.traverse();
    o+=d;//This creates the list of Orders and Drivers

    buff.text(o.c_str());
    trackWindow.add(OrderQ);
    trackWindow.show(); //Displays the window
}
```

8.12 specification.dox File Reference

8.13 traverse.cpp File Reference

`#include "lab.h"`

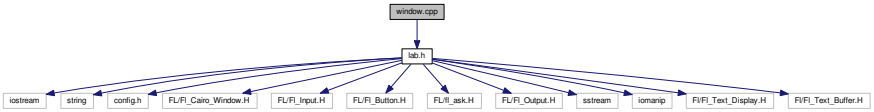
Include dependency graph for traverse.cpp:



8.14 window.cpp File Reference

`#include "lab.h"`

Include dependency graph for window.cpp:



Functions

- Fl_Cairo_Window ∗ window ()

Variables

- Fl_Cairo_Window ∗ cw
- Fl_Input ∗ pizza
- Fl_Input ∗ address
- Fl_Input ∗ Driver
- Fl_Button ∗ Order
- Fl_Button ∗ driver
- Fl_Button ∗ tracker

8.14.1 Function Documentation

8.14.1.1 Fl_Cairo_Window* window ( )

```
{
    cw = new Fl_Cairo_Window(w,h);

    cw->label("Pizza Delivery Services");

    cw->color(FL_GRAY);

    Order = new Fl_Button(200,40,70,20,"Order");
    Order->callback((Fl_Callback*)orderCb);

    driver = new Fl_Button(200,90,70,20,"Driver");
    driver->callback((Fl_Callback*)driverCb);

    tracker = new Fl_Button (150,130,90,30,"Track Order");
    tracker->callback((Fl_Callback*)showQ);

    pizza = new Fl_Input(80,20,100,20,"Pizza: ");
    pizza -> color(FL_WHITE);

    address = new Fl_Input(80,40,100,20, "Address: ");
    address-> color(FL_WHITE);

    Driver = new Fl_Input(80,80,100,20, "Driver: ");
    Driver-> color(FL_WHITE);

    return cw;


```

```
}
```

8.14.2 Variable Documentation

8.14.2.1 Fl_Input* address

8.14.2.2 Fl_Cairo_Window* cw

This function is special because it creates the GUI window. This was takign up too much space in the main function so I split most of into here. I create three buttons and three text inputs. This is to enter the pizza and the drivers as well as show the queues.

8.14.2.3 Fl_Input* Driver

8.14.2.4 Fl_Button* driver

8.14.2.5 Fl_Button* Order

8.14.2.6 Fl_Input* pizza

8.14.2.7 Fl_Button* tracker