Lab5

Sheharyar Alam Khan

April 13, 2019

Contents

1 main.h

```
#include <iostream>
```

Node

This Node structure is a common way to implement a node. it hold a pointer to the next node and the data , or in the case the key. I've called it "key" because it holds the key to its corresponding set of blocks stored in the map. This Node structure also has a constructor that sets the key equal to the string provided as a parameter.

```
struct Node
{
        Node* next = nullptr;
        std::string key = "";
        Node(std::string key);
};
```

Queue

The Queue structure holds the head and tail node pointer of the linked list queue. I've Implemented a linked list queue because I am quite comfortable with linked lists and I am certainly more comfortable dealing with linked lists than I am dealing with a circular array. The Queue has built in push and pop functions as well.

```
struct Queue
{
        Node* head = nullptr;
        Node* tail = nullptr;
        void push(std::string key);
        void pop();
};

Node::Node(std::string key)
{
        this->key = key;
}
```

push

The push function adds a node to back of the list. I decided to make a Queue that adds to the tail and pops from the head because in order to pop from the tail we would have to traverse through the linked list. this inefficient compared to the method I am using which does not require traversing through the entire list.

```
void Queue::push(std::string key)
{
        Node* newnode = new Node(key);
        if(head == nullptr){
                tail = newnode;
                head = newnode;
        }
        else
        {
                tail->next = newnode;
                tail = newnode;
        }
}
```

pop

The pop function pops from the head if the list is not empty. First we make a temporary node pointer which points to head so we don't lose it. Then we move the head to the next node and then delete the temporay node.

```
void Queue::pop()
{
        if(head == nullptr)
                return;
        Node* tmp = head;
        head = head->next;
        delete(tmp);
}
```

```
g++ -g -I/home/debian -I/home/debian/fltk-1.3.4-2 -o "lab" main.cpp `fltk-config --cxxflags
    --ldflags --use-images --use-cairo`
```

2 main.cpp

File Block HandlerSince files can be created and deleted, we use blocks of fixed sizeto hold the data in the file. We need to maintain a set of blocks thatare free to be used for a file, and when file is deleted, to add thoseblocks to a queue of files waiting to have their blocks freed.i.e. There are 2 sets: free blocks and used blocks. A file is a subsetof the used blocks. The queue has the files that are waiting to be freed.

Included Libraries

I have used many libraries in this program. many of which were required.
FL/Fl_Cairo_Window.H : required in order to use an fltk cairo window with cairo graphics and the cairo functions.
FL/Fl_Button.H : required in order to use fltk buttons
FL/Fl_Value_Input.H : required in order to use fltk input boxes
config.h : required in order to configure fltk
<iomanip> : required in order to use setw function
<cmath> : required in order to use sqrt function
<set> : required in order to use sets
<map> : required in order to use maps
<iterator> : required in order to use iterators
<sstream> : required in order to use osstream
<algorithm> : required in order to use set_union

```
#include "config.h"
#include "main.h"
#include <iomanip>
#include "FL/Fl_Cairo_Window.H"
#include "FL/Fl_Button.H"
#include "FL/Fl_Value_Input.H"
#include <iostream>
#include <sstream>
#include <cmath>
#include <set>
#include <map>
#include <iterator>
#include <algorithm>
```

Global Variables

Although the use of global variables is discouraged and may be considered bad practice, I had to use several in order to make this program work. They had to be made global because if I would have to pass all of them by reference into each of the functions being called. We can acheive the same goal by making the variables global and simplify our code.

Queue deleted : deleted is a Queue structure. it is the queue that will hold all the deleted files and delete them after a certain period.

Fl_Input*fileName : this is a pointer to the fltk input box we use to get the file name from the user.

Fl_Input*fileBlocks : this is a pointer to the fltk input box we use to get the number of memory blocks to allocate to the file from the user.

int WIDTH : this is the width of the window.

int HEIGHT : this is the height of the window.

Fl_Button*bc : this is pointer to an fltk button. we will later use this pointer to point to the create button in our program.

Fl_Button*bd : this is pointer to an fltk button. we will later use this pointer to point to the delete button in our program.

typedef unsigned int BLOCKS : this is a type definition. it allows us to substitute a word for a type. in this case we are substituting the word BLOCKS for the type unsigned int.

std::set<BLOCKS> allBlocks : this creates a set of blocks called allBlocks. This is an empty set for now.

std::set<BLOCKS> usedBlocks : this creates a set of blocks called usedBlocks. This is an empty set for now.

std::set<BLOCKS> freeBlocks : this creates a set of blocks called freeBlocks. This set has the number from 0 to 15.

std::map<std::string,std::set<BLOCKS>> files : this creates a map from a string to a set of blocks.

Fl_Cairo_Window cw(WIDTH,HEIGHT) : This make an fltk cairo window of dimensions WIDTH*HEIGHT by calling an overloaded constructor. this will allow us to call redraw in all of the below functions without passing it as a parameter. the redraw function

```
    is essential for this program because we need to remove text and in order to do that we
    need to redraw the window.
    const int N = 16 :  Right now we're assuming that the number of memory blocks on the
    disk is 16.
Queue deleted;
Fl_Input* fileName;
Fl_Input* fileBlocks;
const int WIDTH = 400;
const int HEIGHT = 400;
Fl_Button* bc;
Fl_Button* bd;
typedef unsigned int BLOCKS;
std::set<BLOCKS> allBlocks;  empty set
std::set<BLOCKS> usedBlocks;
std::set<BLOCKS> freeBlocks = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
std::map<std::string,std::set<BLOCKS>> files;
Fl_Cairo_Window cw(WIDTH,HEIGHT);
const int N = 16;      for now, assume 16 blocks on disk
```

`displayState()`

The display state function displays the state of the blocks and the queue on to the console.

we're using a range based loop. auto allows the compiler to selct whichever data type fits best instead of having to manually type it in. the ':' makes the a "for each" loop and this is a new kind of for loop that traverses through each of the elements without the progammer having to worry about the when the list ends.

```
std::string displayState(){
        std::cout<<"========================================="<<std::endl;
        std::cout<<"Current State"<<std::endl;
        std::cout<<"========================================="<<std::endl;
    std::cout << "All blocks:" << std::endl;
    for (auto e:allBlocks) std::cout << e <<" ";
    std::cout<<std::endl;
    std::cout<<"-----------------------------------------"<<std::endl;
    std::cout << "Free blocks:" << std::endl;
    for (auto e:freeBlocks) std::cout << e <<" ";
    std::cout<<std::endl;
    std::cout<<"-----------------------------------------"<<std::endl;
    std::cout << "Used blocks:" << std::endl;
    for (auto e:usedBlocks) std::cout << e <<" ";
    std::cout<<std::endl;
    std::cout<<"-----------------------------------------"<<std::endl;
    std:: ostringstream oss;
    std::cout << "file blocks:"<< std::endl;
    std::cout << "File Name"<< "\t" << "Memory Blocks" << std::endl;
    std::cout << "---------"<< "\t" << "-------------" << std::endl;
    for (auto f:files)
    {
                oss << f.first << std::setw(16);
        for (auto e:f.second)
            oss << e <<" ";
```

```
        oss << std::endl;
        }
    std::cout<<oss.str()<<std::endl;
    std::cout<<"========================================="<<std::endl;
        std::cout<<"Deletion Queue " << std::endl;
        std::cout<<"========================================="<<std::endl;
        Node*tmp = deleted.head;
    while(tmp != nullptr)
    {
           std::cout << tmp->key << std::endl;
           tmp = tmp->next;
        }
        std::cout<<std::endl;
        std::cout<<"========================================="<<std::endl;
    cw.redraw();
    return oss.str();
}
```

This is console output by the program but mostly the displayState function.

```
==========================================
Current State
==========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
------------------------------------------
Free blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
------------------------------------------
Used blocks:

------------------------------------------
file blocks:
File Name Memory Blocks
----------------------

==========================================
Deletion Queue
==========================================

==========================================
==========================================
Created file description
File Name: alpha
Memory Blocks: 3
==========================================
==========================================
Current State
==========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
-----------------------------------------
Free blocks:
0 1 2 4 5 7 8 10 11 12 13 14 15
-----------------------------------------
Used blocks:
3 6 9
-----------------------------------------
file blocks:
File Name Memory Blocks
----------------------
alpha               3 6 9

=========================================
Deletion Queue
=========================================

=========================================
=========================================
Created file description
File Name: beta
Memory Blocks: 5
=========================================
=========================================
Current State
=========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------
Free blocks:
0 1 2 4 5 7 8 14
-----------------------------------------
Used blocks:
```

```
3 6 9 10 11 12 13 15
-----------------------------------------
file blocks:
File Name Memory Blocks
----------------------
alpha                3 6 9
beta              10 11 12 13 15

=========================================
Deletion Queue
=========================================

=========================================
=========================================
Created file description
File Name: gamma
Memory Blocks: 7
=========================================
=========================================
Current State
=========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------
Free blocks:
0
-----------------------------------------
Used blocks:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------
file blocks:
File Name Memory Blocks
```

```
----------------------
alpha                3 6 9
beta              10 11 12 13 15
gamma               1 2 4 5 7 8 14

========================================
Deletion Queue
========================================

========================================
file added to deletion queue: beta
========================================
Current State
========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
----------------------------------------
Free blocks:
0
----------------------------------------
Used blocks:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
----------------------------------------
file blocks:
File Name Memory Blocks
----------------------
alpha                3 6 9
beta              10 11 12 13 15
gamma               1 2 4 5 7 8 14

========================================
Deletion Queue
```

```
==========================================
beta

==========================================
file added to deletion queue: alpha
==========================================
Current State
==========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
------------------------------------------
Free blocks:
0 10 11 12 13 15
------------------------------------------
Used blocks:
1 2 3 4 5 6 7 8 9 14
------------------------------------------
file blocks:
File Name Memory Blocks
----------------------
alpha               3 6 9
gamma               1 2 4 5 7 8 14

==========================================
Deletion Queue
==========================================
alpha

==========================================
file added to deletion queue: gamma
==========================================
Current State
```

```
==========================================
All blocks:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
------------------------------------------
Free blocks:
0 10 11 12 13 15
------------------------------------------
Used blocks:
1 2 3 4 5 6 7 8 9 14
------------------------------------------
file blocks:
File Name Memory Blocks
----------------------
alpha               3 6 9
gamma               1 2 4 5 7 8 14

==========================================
Deletion Queue
==========================================
alpha
gamma

==========================================
```

createCB

This is the callback for the create button. It selects random blocks of free memory. it adds thos blocks to a new set, adds them to the usedBlocks set, and removes the block from the freeBlocks set. it then maps the new set to the filename provided by the user. It then redraws the image.

```
void createCB(void*, void*){
        std::cout<<"========================================="<<std::endl;
    std::cout << "Created file description" << std::endl;
    std::cout << "File Name: " << fileName->value()<<std::endl;
    std::cout << "Memory Blocks: " << fileBlocks->value()<<std::endl;
    std::cout<<"========================================="<<std::endl;
     Select users number of blocks needed from free set
     move those from free set
    std::set<BLOCKS> f;     int x = 3;
    std::set<BLOCKS>::iterator i;
    int BLKS = atoi(fileBlocks->value());
    int found = 0;
    while(found < BLKS && !freeBlocks.empty())
    {
                i = freeBlocks.find(rand() % (N));// find block 3
                if(i != freeBlocks.end())
                {
                        f.insert(*i);
                        usedBlocks.insert(*i);
                        freeBlocks.erase(*i);
                        found++;
                }
        }
    std::string filename = fileName->value();
    files[filename] = f;
    displayState();
    cw.redraw();
```

```
}
```

callback

This is the callback function. it gets called every 10 seconds. id pops one element from the queue and adds the blocks associated to that file back to freeBlocks set and removes it from the used block set. It then redraws the window.

```
void callback(void*){
        if(deleted.head != nullptr){
                std::string key = deleted.head->key;
    //pop 1 file of blocks from 0 (if not empty)
    deleted.pop();
    //add to free set and delete from used set
                for(auto block:files[key])
                {
                        usedBlocks.erase(block); freeBlocks.insert(block);
                }
                files.erase(key);
        }
    Fl::repeat_timeout(10.0, callback);     //call every 10 seconds
    cw.redraw();
}
```

deleteCB

This function is a callback for the delete button it pushes the selected file to the queue for deletion. then it displays the state and redraws the window.

```
void deleteCB(void*, void*){
    //push this file of blocks on the queue
    std::cout<<"file added to deletion queue: ";
    std::string key = fileName->value();
        std::cout << key << std::endl;
    deleted.push(key);
    displayState();
    cw.redraw();
}
```

`drawCB`

This function draws the blocks, displays the file names, and displays the deletion queue.

```
void drawCB(Fl_Cairo_Window* win, cairo_t* cr){
    cairo_set_font_size(cr,20);
    const int s = 35; //scale: pixels per unit
    const int offset = 5; //moving text away from corner
    int COLS = std::sqrt(N);
    int ROWS = COLS;
    for(int i = 0; i < COLS; i++){
        for (int j=0; j < ROWS; j++){
            cairo_set_source_rgb(cr,0,1,0); //green
            cairo_rectangle(cr,i*s,j*s,s,s);
            cairo_stroke(cr);

            cairo_move_to(cr,i*s+offset,j*s+s-offset);
            int blockNumber = i+j*std::sqrt(N);
            std::string b = std::to_string(blockNumber);
            if(freeBlocks.find(blockNumber) != freeBlocks.end())
                    cairo_set_source_rgb(cr,1,0,0);//red
                else
                    cairo_set_source_rgb(cr,0,0,1);//blue
            cairo_show_text(cr,b.c_str());
                }
    }
    std::string str;
    int i = 0;
    for(auto file:files)
    {
                str = file.first;
                cairo_set_source_rgb(cr,0,0,0);
                cairo_move_to(cr,180,25+20*i);
```

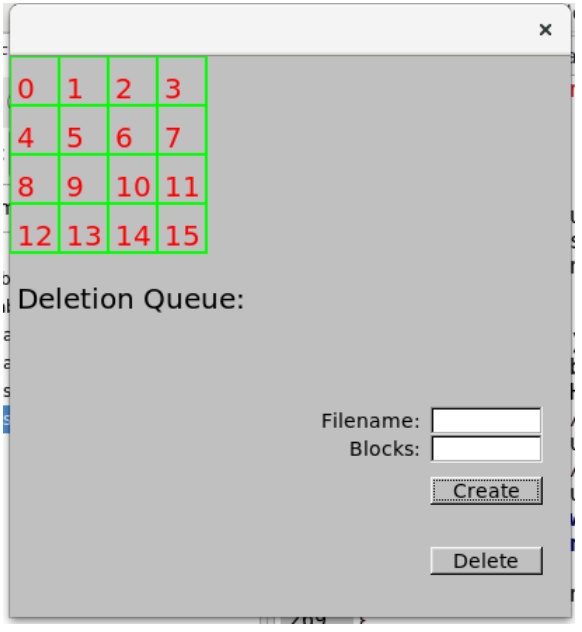
```
                cairo_show_text(cr,str.c_str());
                i++;
        }
        cairo_move_to(cr,5,180);
        cairo_set_source_rgb(cr,0,0,0);
        str = "Deletion Queue:";
        cairo_show_text(cr,str.c_str());
        cairo_move_to(cr,5,215);
        i = 1;
        Node* trav = deleted.head;
        while(trav != nullptr)
        {
                str = trav->key;
                cairo_show_text(cr,str.c_str());
                cairo_move_to(cr,5,215+25*i);
                trav = trav->next;
                i++;
        }
}
```

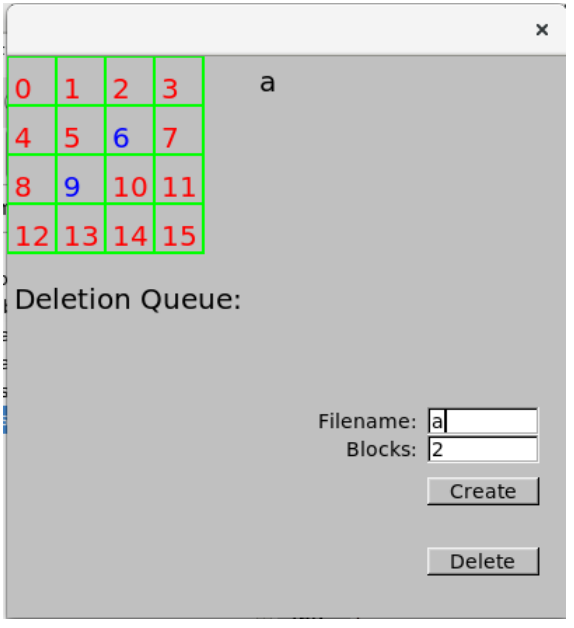
main

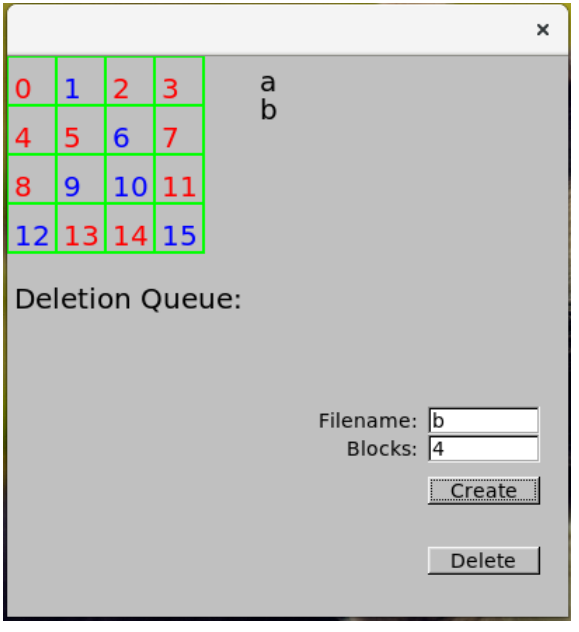
The main function runs the program. and places the button on their position on the window.

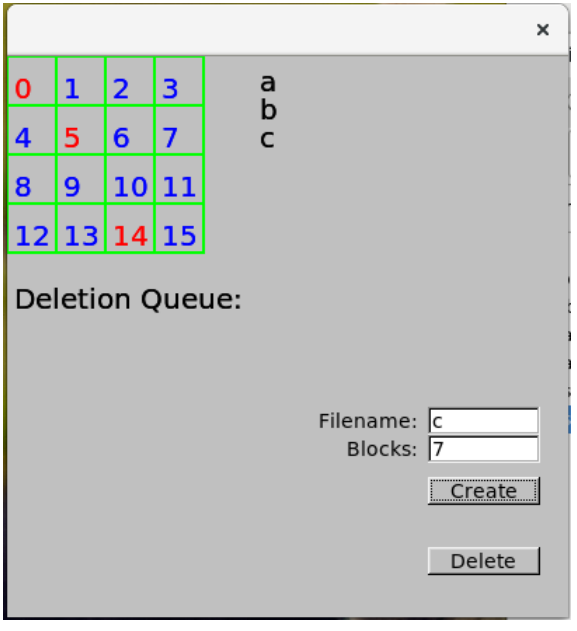
```
int main (void)
{
    Fl::add_timeout(1.0, callback);
    std::cout << std::endl;
    std::set_union(usedBlocks.begin(),usedBlocks.end(),
                   freeBlocks.begin(),freeBlocks.end(),
                   std::inserter(allBlocks,allBlocks.begin()));
    displayState();
    cw.set_draw_cb(drawCB);
    int x =3*WIDTH/4; int y = 3*HEIGHT/4; const char* tc = "Create";
    int w = WIDTH/5; int h = HEIGHT/20;
    bc = new Fl_Button(x,y,w,h,tc);    bc->callback((Fl_Callback*)createCB);
        x=3*WIDTH/4; y = 7*HEIGHT/8; const char* td = "Delete";
    bd = new Fl_Button(x,y,w,h,td);    bd->callback((Fl_Callback*)deleteCB);
    fileName = new Fl_Input(x,y-80-h,w,h,"Filename: ");
    fileBlocks = new Fl_Input(x,y-80,w,h,"Blocks: ");
    cw.show();
    return Fl::run();
}
```

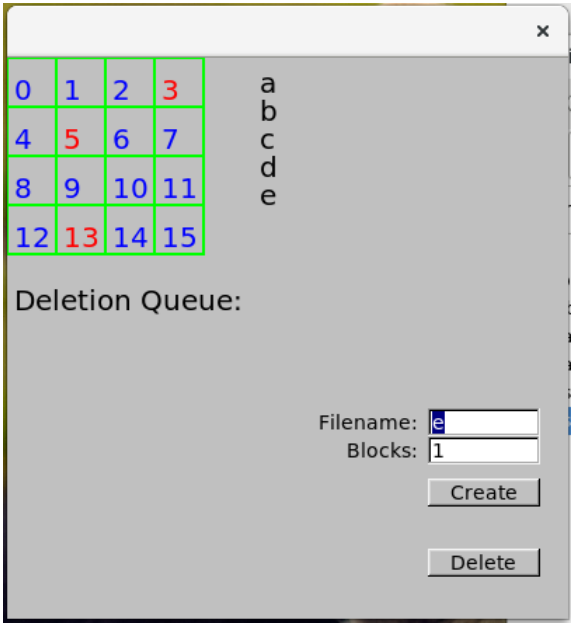
I've Include screenshots from the program.

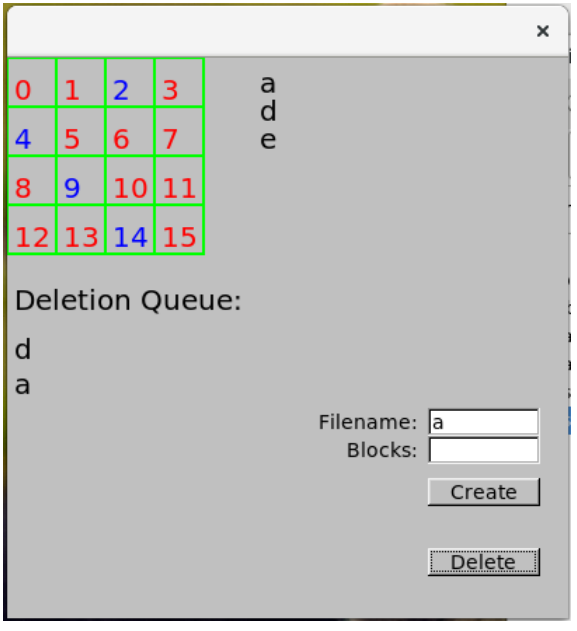


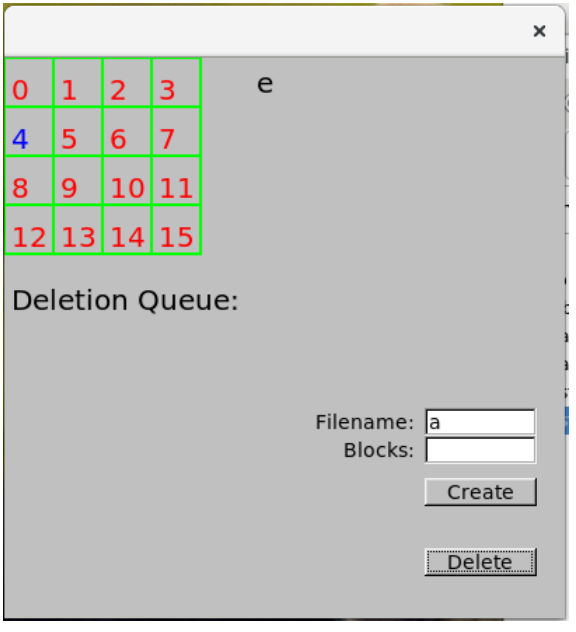

0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
a
Deletion Queue:
Filename: a
Blocks: 2
Create
Delete


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
a
b
Deletion Queue:
Filename: b
Blocks: 4
Create
Delete


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
a
b
c
Deletion Queue:
Filename: c
Blocks: 7
Create
Delete


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
a
b
c
d
e
Deletion Queue:
Filename:
Blocks:
1
Create
Delete


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
a
d
e
Deletion Queue:
d
a
Filename: a
Blocks:
Create
Delete


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
e
Deletion Queue:
Filename: a
Blocks:
Create
Delete

3 testCase.cpp

```
#include "main.h"
#include "catch.hpp"
```

Test Cases

These are the test cases I used to ensure my queue was working as intended. All of the test cases passed so I can conclude that everything is working as intended.
I have include a compact version of the unit testing below.

```
testCase.cpp:9: passed: queue != nullptr for: 0x000055a92b38bc00 != nullptr
testCase.cpp:10: passed: queue->head == nullptr for: nullptr == nullptr
testCase.cpp:11: passed: queue->tail == nullptr for: nullptr == nullptr
testCase.cpp:16: passed: queue->head == nullptr for: nullptr == nullptr
testCase.cpp:18: passed: queue->tail != nullptr for: 0x000055a92b38bb60 != nullptr
testCase.cpp:20: passed: queue->tail->key == "beta" for: "beta" == "beta"
testCase.cpp:22: passed: queue->tail->key == "gamma" for: "gamma" == "gamma"
testCase.cpp:24: passed: queue->tail->key == "delta" for: "delta" == "delta"
testCase.cpp:26: passed: queue->tail->key == "epsilon" for: "epsilon" == "epsilon"
testCase.cpp:31: passed: queue->head->key == "alpha" for: "alpha" == "alpha"
testCase.cpp:33: passed: queue->head->key == "beta" for: "beta" == "beta"
testCase.cpp:35: passed: queue->head->key == "gamma" for: "gamma" == "gamma"
testCase.cpp:37: passed: queue->head->key == "delta" for: "delta" == "delta"
testCase.cpp:39: passed: queue->head->key == "epsilon" for: "epsilon" == "epsilon"
testCase.cpp:41: passed: queue->head == nullptr for: nullptr == nullptr
Passed all 3 test cases with 15 assertions.
```

```
Queue* queue;

TEST_CASE("Make an empty Queue")
{
        queue = new Queue;
        REQUIRE(queue != nullptr);
        REQUIRE(queue->head == nullptr);
        REQUIRE(queue->tail == nullptr);
}
```

```
TEST_CASE("Pushing to the Queue")
{
        REQUIRE(queue->head == nullptr);
        queue->push("alpha");
        REQUIRE(queue->tail != nullptr);
        queue->push("beta");
        REQUIRE(queue->tail->key == "beta");
        queue->push("gamma");
        REQUIRE(queue->tail->key == "gamma");
        queue->push("delta");
        REQUIRE(queue->tail->key == "delta");
        queue->push("epsilon");
        REQUIRE(queue->tail->key == "epsilon");
}

TEST_CASE("Pop from the Queue")
{
        REQUIRE(queue->head->key == "alpha");
        queue->pop();
        REQUIRE(queue->head->key == "beta");
        queue->pop();
        REQUIRE(queue->head->key == "gamma");
        queue->pop();
        REQUIRE(queue->head->key == "delta");
        queue->pop();
        REQUIRE(queue->head->key == "epsilon");
        queue->pop();
        REQUIRE(queue->head == nullptr);
}
```