

CS124TeamProject May 18, 2019

Contents

1	/home/debian/CS124TeamProject/CPPaddHeaderData.cpp	1
2	/home/debian/CS124TeamProject/CPPcheckBalance.cpp	3
3	/home/debian/CS124TeamProject/CPPemptyHeaps.cpp	5
4	/home/debian/CS124TeamProject/CPPfillHeaps.cpp	7
5	/home/debian/CS124TeamProject/CPPgetMedian.cpp	10
6	/home/debian/CS124TeamProject/CPPheapRebalance.cpp	13
7	/home/debian/CS124TeamProject/CPPheapStuff.cpp	18
8	/home/debian/CS124TeamProject/CPPmain.cpp	21
9	/home/debian/CS124TeamProject/CPPreadFiles.cpp	29
10	/home/debian/CS124TeamProject/CPPreadInFiles.cpp	32
11	/home/debian/CS124TeamProject/Hlab.h	35
12	/home/debian/CS124TeamProject/HmaxHeap.h	37
13	/home/debian/CS124TeamProject/HminHeap.h	51

1 /home/debian/CS124TeamProject/CPPaddHeaderData.cpp

```
*****
* File:   CPPaddHeaderData.cpp
* Author(s):  Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class:   CS-124-02
* Term:    Spring 2019
* Lab:     Team Project
* Date:    17 May 2019
* Description:
Using a median filter, this program filters out something blocking
the desired focal point of a series of photos.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```
*****
**** addHeaderData() Function Definition ****
*****

void addHeaderData(string header[], ofstream& newImage)
{
    for(int i = 0; i < 3; i++)
    {
        newImage.write(header[i].c_str(), header[i].length());
        newImage << "\n";
    }
}
```

2 /home/debian/CS124TeamProject/CPPcheckBalance.cpp

```
*****
* File:   CPPcheckBalance.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class:  CS-124-02
* Term:   Spring 2019
* Lab:    Team Project
* Date:   17 May 2019
* Description:
The definition of the function that checks the size of the min heap
and max heap and calls the function to rebalance them if the
difference is greater than 1 or less than -1.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** checkBalance() Function Definition ****
*****

void checkBalance(MinHeap<int>& minHeap, MaxHeap<int>& maxHeap)
{
    int check = minHeap.size() - maxHeap.size();

    cout << endl << "check is:  " << check << endl;

    if (check > 1 || check < (-1))
    {
        cout << "before heapRebalance is called" << endl;
        cout << "size of minHeap:  " << minHeap.size() << endl;
        cout << "size of maxHeap:  " << maxHeap.size() << endl;
        heapRebalance(minHeap, maxHeap);
        cout << "heapRebalance called" << endl;
        cout << "size of minHeap:  " << minHeap.size() << endl;
        cout << "size of maxHeap:  " << maxHeap.size() << endl;
    }

    else
    {
        return;
    }
}

```

3 /home/debian/CS124TeamProject/CPPEmptyHeaps.cpp

```
*****
* File:   CPPEmptyHeaps.cpp
* Author(s):  Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class:   CS-124-02
* Term:    Spring 2019
* Lab:     Team Project
* Date:    17 May 2019
* Description:
The definitions of the functions that empty the max or min heap by
popping all values in the heap.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** emptyMinHeap() Function Definition ****
*****

void emptyMinHeap(MinHeap<int>& minHeap)
{
    pop all values in minHeap
    for(int b = 0; b < minHeap.size(); b++)
    {
        minHeap.pop();
    }
}

*****
**** emptyMaxHeap() Function Definition ****
*****

void emptyMaxHeap(MaxHeap<int>& maxHeap)
{
    pop all values in maxHeap
    for(int b = 0; b < maxHeap.size(); b++)
    {
        maxHeap.pop();
    }
}

```

4 /home/debian/CS124TeamProject/CPPfillHeaps.cpp

```
*****
* File:   CPPfillHeaps.cpp
* Author(s):  Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class:   CS-124-02
* Term:    Spring 2019
* Lab:     Team Project
* Date:    17 May 2019
* Description:
The definition of the function where the heaps are filled and then
calls the function to check if they need to be rebalanced.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```



```

*****
**** fillHeaps() Function Definition ****
*****

void fillHeaps(int a, MinHeap<int>& minHeap, MaxHeap<int>& maxHeap, vector<
    vector<int>>& pixelHolder)
{
    compare the current value of the first two files
    if(pixelHolder[0][a] > pixelHolder[1][a])
    {
        minHeap.push(pixelHolder[0][a]);

        maxHeap.push(pixelHolder[1][a]);
    }

    else if(pixelHolder[0][a] < pixelHolder[1][a])
    {
        maxHeap.push(pixelHolder[0][a]);

        minHeap.push(pixelHolder[1][a]);
    }
}

```

```

        iterate through rest of files
for(int i = 2; i < pixelHolder.size(); i++)
{
    if(pixelHolder[i][a] < minHeap.top())
    {
        maxHeap.push(pixelHolder[i][a]);
        checkBalance(minHeap, maxHeap);
    }

    else if(pixelHolder[i][a] > minHeap.top())
    {
        minHeap.push(pixelHolder[i][a]);
        checkBalance(minHeap, maxHeap);
    }
}

    cout << "fillHeaps: after filling the heaps: " << endl;
    cout << "size of minHeap: " << minHeap.size() << endl;
    cout << "size of maxHeap: " << maxHeap.size() << endl;
}

```

5 /home/debian/CS124TeamProject/CPPgetMedian.cpp

```
*****
* File:   CPPgetMedian.cpp
* Author(s):  Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class:   CS-124-02
* Term:    Spring 2019
* Lab:     Team Project
* Date:    17 May 2019
* Description:
The definition of the function that finds and print the median from
the roots of the min and max heaps.
If the min heap is larger, the root of the min heap is the median.
If the max heap is larger, the root of the max heap is the median.
If the heaps are of equal size, the average of the roots is the median.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** getMedian() Function Definition ****
*****

void getMedian(ofstream& newImage, MinHeap<int>& minHeap, MaxHeap<int>&
               maxHeap, int count)
{
    cout << "getMedian: before size comparisons and median prints: " << endl;
    cout << "size of minHeap: " << minHeap.size() << endl;
    cout << "size of maxHeap: " << maxHeap.size() << endl;

    if(minHeap.size() > maxHeap.size()) if minHeap is larger
    {
        cout << "MinHeap.top: " << minHeap.top() << endl;
        newImage << minHeap.top();
    }

    else if(minHeap.size() < maxHeap.size()) if maxHeap is larger
    {
        cout << "MaxHeap.top: " << maxHeap.top() << endl;
        newImage << maxHeap.top();
    }

    else if the heaps are equal size
    {
        int tmp = (maxHeap.top() + minHeap.top()) / 2;
        cout << "Median is: " << tmp << endl;
        newImage << tmp;
    }
}

```

```
    if(count%3 == 0)
    {
        newImage << "\n";
    }

    else
    {
        newImage << " ";
    }
}
```

6 /home/debian/CS124TeamProject/CPPheapRebalance.cpp

```
*****
* File: CPPheapRebalance.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The definition of the function that rebalances the min and max heap
when the size of one is greater than the other by two or more values.
This is accomplished by removing and storing the root value of the
larger heap, reHeaping the reduced heap to maintain the desired
priority order and then pushing the stored value to the other heap.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** heapRebalance() Function Definition ****
*****

void heapRebalance(MinHeap<int>& minHeap, MaxHeap<int>& maxHeap)
{
    if(minHeap.size() > maxHeap.size()) if minHeap is larger
    {
        cout << "more min" << endl;

        minHeap.fix_heap_public();

        cout << "heapRebalance: after fixHeap: " << endl;
        cout << "size of minHeap: " << minHeap.size() << endl;

        int temp = minHeap.top();

        cout << "temp value: " << temp << endl;

        minHeap.pop();

        cout << "heapRebalance: after minHeap.pop(): " << endl;
        cout << "size of minHeap: " << minHeap.size() << endl;
        cout << "size of maxHeap: " << maxHeap.size() << endl;
    }
}

```

```
        maxHeap.push(temp);

        cout << "heapRebalance: after maxHeap.push(temp): " << endl;
    cout << "size of minHeap: " << minHeap.size() << endl;
    cout << "size of maxHeap: " << maxHeap.size() << endl;
    cout << "MinHeap" << endl;
    minHeap.printHeapPublic();
    cout << "MaxHeap" << endl;
    maxHeap.printHeapPublic();
}
```



```

else if(minHeap.size() < maxHeap.size()) if maxHeap is larger
{
    cout << "more max" << endl;

    maxHeap.fix_heap_public();

    cout << "heapRebalance: after fixHeap: " << endl;
    cout << "size of maxHeap: " << maxHeap.size() << endl;

    int temp = maxHeap.top();

    cout << "temp value: " << temp << endl;

    maxHeap.pop();

    cout << "heapRebalance: after maxHeap.pop(): " << endl;
    cout << "size of minHeap: " << minHeap.size() << endl;
    cout << "size of maxHeap: " << maxHeap.size() << endl;
}

```

```

        minHeap.push(temp);

        cout << "heapRebalance: after minHeap.push(temp): " << endl;
cout << "size of minHeap: " << minHeap.size() << endl;
cout << "size of maxHeap: " << maxHeap.size() << endl;
cout << "MinHeap" << endl;
minHeap.printHeapPublic();
cout << "MaxHeap" << endl;
maxHeap.printHeapPublic();
    }

    else if heaps are equal
    {
        cout << "Heaps are equal" << endl;
    }
}

```

7 /home/debian/CS124TeamProject/CPPheapStuff.cpp

```
*****
* File: CPPheapStuff.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The definition of the function that calls functions to create and fill
the min and max heaps, find, retrieve and print the median values, and
reset the heaps.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** heapStuff() Function Definition ****
*****

void heapStuff(ofstream& newImage, MinHeap<int>& minHeap, MaxHeap<int>&
maxHeap, vector<vector<int>>& pixelHolder)
{
    MinHeap<int> minHeapTemp = minHeap;
    MaxHeap<int> maxHeapTemp = maxHeap;
    int count = 1;

    for(int a = 0; a < pixelHolder[0].size(); a++)
    { iterate through all lines/vector values

        ***** Heaps filled *****
        fillHeaps(a, minHeapTemp, maxHeapTemp, pixelHolder);

        cout << "avoided segv 01" << endl;

        ***** Median Obtained *****
        cout << "heapStuff: before getMedian: " << endl;
        cout << "size of minHeap: " << minHeapTemp.size() << endl;
        cout << "size of maxHeap: " << maxHeapTemp.size() << endl;
        if(minHeapTemp.size() != 0 && maxHeapTemp.size() != 0)
        {
            getMedian(newImage, minHeapTemp, maxHeapTemp, count)
            ;

            count++;
        }

        cout << "avoided segv 02" << endl;
    }
}

```

```

        ***** Heaps emptied *****
        emptyMinHeap(minHeapTemp);

        cout << "avoided segv 03" << endl;

        minHeapTemp = minHeap;

        cout << "avoided segv 04" << endl;

        emptyMaxHeap(maxHeapTemp);

        cout << "avoided segv 05" << endl;

        maxHeapTemp = maxHeap;

        cout << "avoided segv 06" << endl;
    }

    cout << "avoided segv 07" << endl;
}

```

8 /home/debian/CS124TeamProject/CPPmain.cpp

```
*****
* File: CPPmain.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
Using a median filter, this program filters out something blocking
the desired focal point of a series of photos.
*****

Compilation Command:
g++ -g -I/home/debian -o lab CPP*.cpp -I/home/debian/fttk-1.3.4-2 'fttk-config --cxxflags
--ldflags --use-images --use-cairo'

** System Libraries **
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"
```

```

** FLTK Libraries**
#include "config.h"
#include "FL/Fl_Cairo_Window.H"
#include "FL/Fl_Button.H"
#include "FL/Fl_Value_Input.H"
#include "FL/Fl_Text_Display.H"
#include <FL/Fl_Input.H>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_Box.H>

** Namespace Declaration **
using namespace std;

** Global Variable(s) **
vector<vector<int>> pixelHolder;
string file_list;

* FLTK Global Variables **
const int WIDTH = 600;
const int HEIGHT = 600;
Fl_Input* prefix = nullptr;
Fl_Button* submit = nullptr;
Fl_Button* findBut = nullptr;
Fl_Button* clearBut = nullptr;
Fl_Button* showBut = nullptr;
Fl_Text_Display* files = nullptr;
Fl_Text_Buffer* buff = nullptr;
Fl_Cairo_Window* cw = nullptr;
Fl_PNG_Image* png = nullptr;
Fl_Box* imgBox = nullptr;

```

```

* Global Pointers (for use in FLTK callbacks) *
ofstream *newImagePointer;
MinHeap<int>* minHeapPointer;
MaxHeap<int>* maxHeapPointer;

* FLTK Callback declarations *
void submitCB(void*, void*);
void findCB(void*, void*);
void showCB(void*, void*);
void clearCB(void*, void*);

*****
***** Main Program *****
*****

int main()
{
    ofstream debug; debug.open("debug.txt");

    cout << "avoided segv A" << endl;

    ofstream newImage;

    cout << "avoided segv B" << endl;

    newImage.open("output.ppm");

    cout << "avoided segv C" << endl;

    MinHeap<int> minHeap;

    cout << "avoided segv D" << endl;

```



```

MaxHeap<int> maxHeap;

cout << "avoided segv E" << endl;

newImagePointer = &newImage;
minHeapPointer = &minHeap;
maxHeapPointer = &maxHeap;

readFiles(newImage, pixelHolder, debug);
/* cout << "avoided segv F" << endl;
/* for(int i = 0; i < pixelHolder.size(); i++)
cout << "There are " << pixelHolder[i].size() << " pixels in each picture" << endl;
    cout << "avoided segv G" << endl;
heapStuff(newImage, minHeap, maxHeap, pixelHolder);

cw = new Fl_Cairo_Window(WIDTH, HEIGHT);

int w, h, x, y;

w = WIDTH * 0.3;
h = HEIGHT * 0.1;
x = WIDTH * 0.6;
y = HEIGHT * 0.65;
findBut = new Fl_Button(x, y, w, h, "Find Files");
findBut->callback((Fl_Callback*)findCB);

```

```

w = WIDTH * 0.3;
h = HEIGHT * 0.1;
x = (WIDTH - w) * 0.5;
y = HEIGHT * 0.85;
submit = new Fl_Button(x, y, w, h, "Submit");
submit->callback((Fl_Callback*)submitCB);

w = WIDTH * 0.3;
h = HEIGHT * 0.1;
x = WIDTH * 0.6;
y = HEIGHT * 0.5;
clearBut = new Fl_Button(x, y, w, h, "Clear");
clearBut->callback((Fl_Callback*)clearCB);

w = WIDTH * 0.3;
h = HEIGHT * 0.1;
x = WIDTH * 0.6;
y = HEIGHT * 0.3;
showBut = new Fl_Button(x, y, w, h, "Show");
showBut->callback((Fl_Callback*)showCB);

w = WIDTH * 0.5;
h = HEIGHT * 0.5;
x = WIDTH * 0.1;
y = HEIGHT * 0.1;
files = new Fl_Text_Display(x, y, w, h, "Files Found:");

```

```

        w = WIDTH * 0.3;
        h = HEIGHT * 0.1;
        x = WIDTH * 0.3;
        y = HEIGHT * 0.65;
        prefix = new Fl_Input(x, y, w, h, "File Prefix:");
        buff = new Fl_Text_Buffer;

        files->buffer(*buff);
        cw->show();

        return Fl::run();
    } End of Main

    *****
    ** FLTK Callback Definitions **
    *****

    ***** Submit Button's Callback *****

    void submitCB(void*, void*)
    {
        cout << "removing noise" << endl;
        for(int i = 0; i < pixelHolder.size(); i++)
        {
            cout << "There are " << pixelHolder[i].size() << "
                pixels in each picture" << endl;
        }
        heapStuff(*newImagePointer, *minHeapPointer, *maxHeapPointer,
            pixelHolder);
        cw->redraw();
        cout << "noise removed" << endl;
    }

```

***** Find Button's Callback *****

```
void findCB(void*, void*)
{
    cout << "finding" << endl;
    readFiles(*newImagePointer, pixelHolder, file_list, prefix->value())
    ;
    buff->append(file_list.c_str());
    file_list = "";
    cw->redraw();
    cout << "found" << endl;
}
```

***** Clear Button's Callback *****

```
void clearCB(void*, void*)
{
    cout << "clearing" << endl;
    buff->text("");
    file_list = "";
    cw->redraw();
    cout << "cleared" << endl;
}
```

```

***** Show Button's Callback *****

void showCB(void*, void*)
{
    cout << "converting" << endl;
    system("convert output.ppm output.jpg");
    cout << "converted" << endl;
    //png = new Fl_PNG_Image("output.png");
    cout << "displaying" << endl;
    //imgBox = new Fl_Box(WIDTH,0,WIDTH,HEIGHT);
    //imgBox->image(png);
    //cw->image(png);
    //cw->redraw();
    system("display output.jpg");
    cout << "displayed" << endl;
}

```

9 /home/debian/CS124TeamProject/CPPreadFiles.cpp

```
*****
* File: CPPreadFiles.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The definition of the function that locates all image source files
through a system command, pipes the names to another file, reads
those names into a vector and then calls the function to read in
each file's contents to another set of vectors.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** readFiles() Function Definition ****
*****

//void readFiles(ofstream& newImage, vector<vector<int>>& pixelHolder,
    ofstream& debug)
void readFiles(ofstream& newImage, vector<vector<int>>& pixelHolder, string
    &file_list, string filePrefix)
{
    vector<string> numFiles;  This vector will hold the names of the
    files
    string line = " ";
    int counter = 0;

    system("ls image*.ppm > file.txt"); // Grabs all PPM images in current
    directory
    system("ls test*.ppm > file.txt"); //Grabs all test PPM images in
    current directory
    string command = "ls " + filePrefix + "*.ppm > file.txt";
    system(command.c_str());  Grabs all test PPM images in current
    directory

    ifstream ifs("file.txt");  Opens the newly created text file

    while(getline(ifs, line))  Grabs the file names from text file and
    pushed it into vector
    {
        numFiles.push_back(line);
        file_list += line + "\n";
        counter++;
    }
}

```

```

        cout << "There are " << counter << " PPM files " << endl; //Confirms
        how many PPM Files are in directory
for(int i = 0; i < counter; i++)
cout << numFiles[i] << endl; //Prints out all file names
cout << "There are " << numFiles.size() << " Files" << endl;
        readInFiles(numFiles, newImage, pixelHolder);  Calls to next
        function
}

```


10 /home/debian/CS124TeamProject/CPPreadInFiles.cpp

```
*****
* File: CPPreadInFiles.cpp
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The definition of a function that opens input streams for all the
source files, reads a line of input from each file, then pushes it to
a vector to store the int values of the pixels for later processing.
*****

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "Hlab.h"
#include "HmaxHeap.h"
#include "HminHeap.h"

** namespace declaration **
using namespace std;
```

```

*****
**** readInFiles() Function Definition ****
*****
//void readInFiles(vector<string>& fNames, ofstream& newImage, vector<vector
<int>>& pixelHolder, ofstream& debug)
void readInFiles(vector<string>& fNames, ofstream& newImage, vector<vector<
int>>& pixelHolder)
{
    string header[3] = {};

    int numbers = 0;

    vector<int> tempFileHolder; Temp vector which holds all the pixels for one
    image

    for(int i = 0; i < fNames.size(); i++)
    {
        //debug << "File #" << i << endl;

        const char * fileOpen = fNames[i].c_str();

        cout << "Opening: " << fileOpen << endl;

        ifstream data(fileOpen);

        for(int j = 0; j < 3; j++) Grabs the header for each picture so
        only pixels are left in each picture
        {
            getline(data, header[j]);
            cout << "The header value is: " << header[j] <<
            endl;
        }
    }
}

```

```

while(data >> numbers)
{
    tempFileHolder.push_back(numbers);
}

//debug << "size of temp vector: " << tempFileHolder.size()
    << endl;

pixelHolder.push_back(tempFileHolder);

//debug << "size of pixelHolder: " << pixelHolder.size() <<
    endl;

tempFileHolder.clear();
}

addHeaderData(header, newImage);
}

```

11 /home/debian/CS124TeamProject/Hlab.h

```
*****
* File: Hlab.h
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The class file containing declarations of all the functions needed for
the project.
*****

** Duplication Protection **
#ifndef LAB_H
#define LAB_H

** System Libraries **
#include <fstream>
#include <iostream>
#include <vector>

** User Libraries **
#include "HmaxHeap.h"
#include "HminHeap.h"

** Namespace Declaration **
using namespace std;
```

```

** Function Declarations **

void addHeaderData(string[], ofstream&);

//void readInFiles(vector<string>&, ofstream&, vector<vector<int>>&,
//                ofstream&);
void readInFiles(vector<string>&, ofstream&, vector<vector<int>>&);

//void readFiles(ofstream&, vector<vector<int>>&, ofstream&);
void readFiles(ofstream&, vector<vector<int>>&, string &file_list, string
               filePrefix);

void heapRebalance(MinHeap<int>&, MaxHeap<int>&);

void checkBalance(MinHeap<int>&, MaxHeap<int>&);

void getMedian(ofstream&, MinHeap<int>&, MaxHeap<int>&, int);

void fillHeaps(int, MinHeap<int>&, MaxHeap<int>&, vector<vector<int>>&);

void emptyMinHeap(MinHeap<int>&);

void emptyMaxHeap(MaxHeap<int>&);

void heapStuff(ofstream&, MinHeap<int>&, MaxHeap<int>&, vector<vector<int>>&);

#endif  end of LAB.H

```

12 /home/debian/CS124TeamProject/HmaxHeap.h

```
*****
* File: H-maxHeap.h
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The class file containing declarations of all the functions needed to
create a max heap.
*****

** Duplication Protection **
#ifndef MAXHEAP_H
#define MAXHEAP_H

** System Libraries **
#include <vector>
#include <iostream>

** Namespace Declaration **
using namespace std;
```

```

*****
* This class implements a heap. *
*****

template<typename T>

class MaxHeap
{
    public:

        ***** Public Member Functions *****

        Constructs an empty heap.
        MaxHeap();

        Adds a new element to this heap.
        element:  the element to add
        void push(T element);

        Gets the maximum element stored in this heap.
        returns:  the maximum element
        T top() const;

        Removes the maximum element from this heap.
        void pop();

```

Gets the size of this heap.

returns: the size

```
int size() const;
```

Re-heaps to preserve desired priority order

```
void fix_heap_public();
```

Print all entries in the heap

```
void print_heap_public();
```

```
private:
```

```
***** Private Variable(s) *****
```

```
vector<T> elements;
```

```
***** Private Member Functions *****
```

Turns the tree back into a heap, provided only the root

node violates the heap condition.

```
void fix_heap();
```

Returns the index of the left child.

index: the index of a node in this heap

returns: the index of the left child of the given node

```
int get_left_child_index(int index) const;
```

Returns the index of the right child.

index: the index of a node in this heap

returns: the index of the right child of the given node

```
int get_right_child_index(int index) const;
```

Returns the index of the parent.

index: the index of a node in this heap

returns: the index of the parent of the given node

```
int get_parent_index(int index) const;
```

Returns the value of the left child.

index: the index of a node in this heap

returns: the value of the left child of the given node

```
T get_left_child(int index) const;
```

Returns the value of the right child.

index: the index of a node in this heap

returns: the value of the right child of the given node

```
T get_right_child(int index) const;
```

Returns the value of the parent.

index: the index of a node in this heap

returns: the value of the parent of the given node

```
T get_parent(int index) const;
```

```

*****

Print all entries in the heap
    void print_heap();

};

***** MaxHeap() Constructor *****

template<typename T>

MaxHeap<T>::MaxHeap()

{
    T dummy;
    elements.push_back(dummy);
}

```

```

*****
***** Public Member Functions *****
*****

***** push() Function *****

template<typename T>

void MaxHeap<T>::push(T new_element)
{
    Add a new leaf
    T dummy;

    elements.push_back(dummy);

    int index = elements.size() - 1;

    Demote parents that are smaller than the new element
    while (index > 1 && get_parent(index) < new_element)
    {
        elements[index] = get_parent(index);

        index = get_parent_index(index);
    }

    Store the new element into the vacant slot
    elements[index] = new_element;
}

```

```

***** top() Function *****

template<typename T>

T MaxHeap<T>::top() const
{
    return elements[1];
}

***** pop() Function *****

template<typename T>

void MaxHeap<T>::pop()
{
    Remove last element

    int last_index = elements.size() - 1;

    T last = elements[last_index];

    elements.pop_back();

    if (last_index > 1)
    {
        elements[1] = last;
        fix_heap();
    }
}

```

```

***** size() Function *****

template<typename T>

int MaxHeap<T>::size() const
{
    return elements.size() - 1;
}

***** fixHeapPublic() Function *****

template<typename T>

void MaxHeap<T>::fix_heap_public()
{
    fix_heap();
}

***** printHeapPublic() Function *****

template<typename T>

void MaxHeap<T>::print_heap_public()
{
    print_heap();
}

```

```

*****
***** Private Member Functions *****
*****

***** fixHeap() Function *****

template<typename T>

void MaxHeap<T>::fix_heap()
{

    T root = elements[1];

    int last_index = elements.size() - 1;

    Promote children of removed root while they are smaller than root

    int index = 1;

    bool done = false;

    while (!done)
    {
        int child_index = get_left_child_index(index);
        if (child_index <= last_index)
        {
            Get larger child

            Get left child first
            T child = get_left_child(index);

```

```

        Use right child instead if it is larger
    if (get_right_child_index(index) <= last_index
    && child < get_right_child(index))
    {
        child_index = get_right_child_index(index);
        child = get_right_child(index);
    }

    Check if larger child is larger than root
    if (root < child)
    {
        Promote child
        elements[index] = child;
        index = child_index;
    }

    else
    { Root is larger than both children
      done = true;
    }
}

else
{ No children
  done = true;
}
}

Store root element in vacant slot
elements[index] = root;
}

```



```

***** getLeftChildIndex() Function *****

template<typename T>

int MaxHeap<T>::get_left_child_index(int index) const
{
    return 2 * index;
}

***** getRightChildIndex() Function *****

template<typename T>

int MaxHeap<T>::get_right_child_index(int index) const
{
    return 2 * index + 1;
}

***** getParentIndex() Function *****

template<typename T>

int MaxHeap<T>::get_parent_index(int index) const
{
    return index / 2;
}

```

```

***** getLeftChild() Function *****

template<typename T>

T MaxHeap<T>::get_left_child(int index) const
{
    return elements[2 * index];
}

***** getRightChild() Function *****

template<typename T>

T MaxHeap<T>::get_right_child(int index) const
{
    return elements[2 * index + 1];
}

***** getParent() Function *****

template<typename T>

T MaxHeap<T>::get_parent(int index) const
{
    return elements[index / 2];
}

```

```

***** printHeap() Function *****

template<typename T>

void MaxHeap<T>::print_heap()
{
    for(int i = 1; i < elements.size(); i++)
    {
        std::cout << elements[i] << std::endl;
    }
}

#endif end of MAXHEAP.H

```

13 /home/debian/CS124TeamProject/HminHeap.h

```
*****
*
* File: H-minHeap.h
* Author(s): Natalie Peck, Gurpreet Singh, Justin Anaya, Sheharyar Khan
* Class: CS-124-02
* Term: Spring 2019
* Lab: Team Project
* Date: 17 May 2019
* Description:
The class file containing declarations of all the functions needed to
create a min heap.
*****

** Duplication Protection **
#ifndef MINHEAP_H
#define MINHEAP_H

** System Libraries **
#include <vector>
#include <iostream>

** Namespace Declaration **
using namespace std;
```

```

*****
* This class implements a heap. *
*****

template<typename T>

class MinHeap
{
    public:

        ***** Public Member Functions *****

        Constructs an empty heap.
        MinHeap();

        Adds a new element to this heap.
        Parameter element:  the element to add
        void push(T element);

        Gets the minimum element stored in this heap.
        returns:  the minimum element
        T top() const;

        Removes the minimum element from this heap.
        void pop();

```

Gets the size of this heap.

returns: the size

```
int size() const;
```

Re-heaps to preserve desired priority order

```
void fix_heap_public();
```

Print all entries in the heap

```
void print_heap_public();
```

```
private:
```

```
***** Private Variable(s) *****
```

```
vector<T> elements;
```

```
***** Private Member Functions *****
```

Turns the tree back into a heap, provided only the root

node violates the heap condition.

```
void fix_heap();
```

Returns the index of the left child.

Parameter index: the index of a node in this heap

returns: the index of the left child of the given node

```
int get_left_child_index(int index) const;
```

Returns the index of the right child.

Parameter index: the index of a node in this heap

returns: the index of the right child of the given node

```
int get_right_child_index(int index) const;
```

Returns the index of the parent.

Parameter index: the index of a node in this heap

returns: the index of the parent of the given node

```
int get_parent_index(int index) const;
```

Returns the value of the left child.

Parameter index: the index of a node in this heap

returns: the value of the left child of the given node

```
T get_left_child(int index) const;
```

Returns the value of the right child.

Parameter index: the index of a node in this heap

returns: the value of the right child of the given node

```
T get_right_child(int index) const;
```

Returns the value of the parent.

Parameter index: the index of a node in this heap

returns: the value of the parent of the given node

```
T get_parent(int index) const;
```



```

*****

Print all entries in the heap
    void print_heap();

};

***** MinHeap() Constructor *****

template<typename T>
MinHeap<T>::MinHeap()
{
    T dummy;
    elements.push_back(dummy);
}

```

```

*****
***** Public Member Functions *****
*****

***** push() Function *****

template<typename T>

void MinHeap<T>::push(T new_element)
{
    Add a new leaf
    T dummy;

    elements.push_back(dummy);

    int index = elements.size() - 1;

    Demote parents that are bigger than the new element
    while (index > 1 && get_parent(index) > new_element)
    {
        elements[index] = get_parent(index);

        index = get_parent_index(index);
    }

    Store the new element into the vacant slot
    elements[index] = new_element;
}

```

```

***** top() Function *****

template<typename T>

T MinHeap<T>::top() const
{
    return elements[2];
}

***** pop() Function *****

template<typename T>

void MinHeap<T>::pop()
{
    Remove last element

    int last_index = elements.size() - 1;

    T last = elements[last_index];

    elements.pop_back();

    if (last_index > 1)
    {
        elements[1] = last;
        fix_heap();
    }
}

```

```

***** size() Function *****

template<typename T>

int MinHeap<T>::size() const
{
    return elements.size() - 1;
}

***** fixHeapPublic() Function *****

template<typename T>

void MinHeap<T>::fix_heap_public()
{
    fix_heap();
}

***** printHeapPublic() Function *****

template<typename T>

void MinHeap<T>::print_heap_public()
{
    print_heap();
}

```

```

*****
***** Private Member Functions *****
*****

***** fixHeap() Function *****

template<typename T>

void MinHeap<T>::fix_heap()
{

    T root = elements[1];

    int last_index = elements.size() - 1;

    Promote children of removed root while they are bigger than root

    int index = 1;

    bool done = false;

    while (!done)
    {
        int child_index = get_left_child_index(index);
        if (child_index <= last_index)
        {
            Get smaller child

            Get left child first
            T child = get_left_child(index);

```

```

        Use right child instead if it is smaller
    if (get_right_child_index(index) <= last_index
    && child > get_right_child(index))
    {
        child_index = get_right_child_index(index);
        child = get_right_child(index);
    }

    Check if smaller child is smaller than root
    if (root > child)
    {
        Promote child
        elements[index] = child;
        index = child_index;
    }

    else
    { Root is smaller than both children
      done = true;
    }
}

else
{ No children
  done = true;
}
}

Store root element in vacant slot
elements[index] = root;
}

```

```

***** getLeftChildIndex() Function *****

template<typename T>

int MinHeap<T>::get_left_child_index(int index) const
{
    return 2 * index;
}

***** getRightChildIndex() Function *****

template<typename T>

int MinHeap<T>::get_right_child_index(int index) const
{
    return (2 * index) + 1;
}

***** getParentIndex() Function *****

template<typename T>

int MinHeap<T>::get_parent_index(int index) const
{
    return index / 2;
}

```

```

***** getLeftChild() Function *****

template<typename T>

T MinHeap<T>::get_left_child(int index) const
{
    return elements[2 * index];
}

***** getRightChild() Function *****

template<typename T>

T MinHeap<T>::get_right_child(int index) const
{
    return elements[2 * index + 1];
}

***** getParent() Function *****

template<typename T>

T MinHeap<T>::get_parent(int index) const
{
    return elements[index / 2];
}

```



```
***** printHeap() Function *****  
  
template<typename T>  
  
void MinHeap<T>::print_heap()  
{  
    for(int i = 1; i < elements.size(); i++)  
    {  
        std::cout << elements[i] << std::endl;  
    }  
}  
  
#endif end of MINHEAP.H
```