# CSCE 121: Final Project

The following paper demonstrates the design process for the development and implementation of the material learned in Introduction to Program Design and Concepts.

## **Team information**

**Team name**: “Pancake Pro”

**Team members**:

* Gerardo Vazquez
* Will Tallent
* Yuan Yao

**Assignments**:

In our first meeting, we assigned tasks. Gerardo Vazquez was assigned the scoreboard and free store functions such as collecting the users nickname and calculating the score. Will Tallent was assigned the buttons and callbacks; this means, Will was assigned the splash screen, instructions, and different windows throughout the game. Lastly, Yuan Yao was assigned the pancake algorithm to generate graphical objects and a function to sort them as desired by the user.

## **Problem Definition**

The assignment specified is to design a program using C++11 and the FLTK graphical user interface to implement a pancake sorting game. The main challenge was to create a stack of pancakes of unique sizes and let the user sort them in order from largest to smallest going from the bottom to the top.

**Constraints**:

The most significant restriction was time. The limited availability to meet due to personal schedules proved to be the most significant constraint. Furthermore, the lack of experience from our team required research and the use of functions not yet covered at the time implemented in our source code.

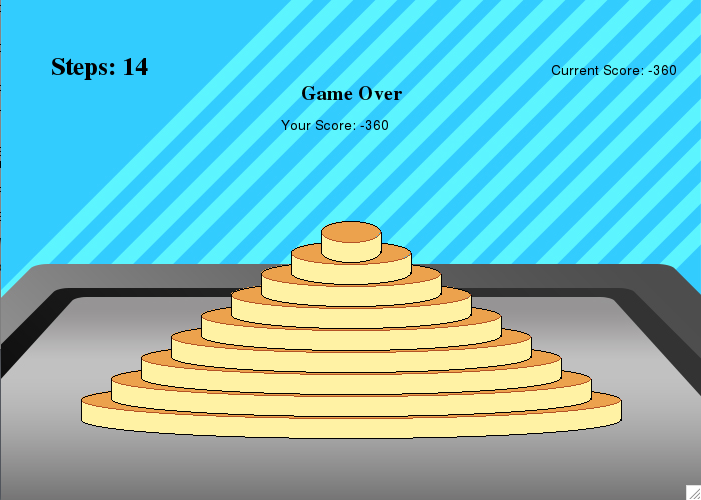
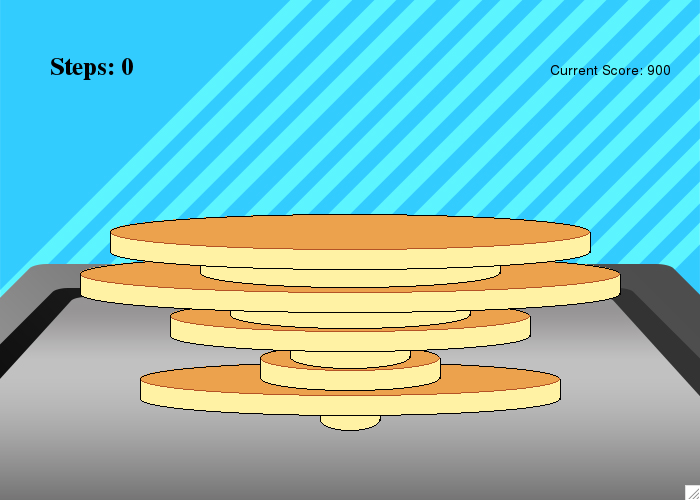
**Approach**:

We adopted the strategy of starting with small amounts of code and continually implementing more segments until a functional program was achieved. Our top priority was to get the pancake stack to create individual objects for each pancake, flipping them accordingly and concluding the project with a functional graphical game.

## **Program**

The program works using the files provided by Dr. Daugherity as parents for our Base\_screen, find\_solution, Game\_screen, Pancake, PancakeStack, Record, Scoreboard, and Splash\_screen source and header files.

**Screenshots**:



**Results and Analysis**:

The resulting program is practical in functionality. The core of the program revolves around the Pancake Stack source code. In it the pancake shape objects are created and shuffled using recursion to check it’s order. The main program transitions between windows and will conclude once the exit bottom is selected.

**Conclusion**:

The project was completed successfully on time as required in the specification. Initially, Yuan Yao took the lead implementing the pancake stack and sorting algorithm. We are convinced we worked as a team in good faith, but unfortunately the tasks planned shifted as Yuan completed the project with no trouble. The team learned the basic functions of the FLTK library, several standard template functions, and the implementation of callbacks.

**Improvements**:

The program could be improved by using formatted buttons to match the style of the game instead of overlapping them with an image. Furthermore, the algorithm used to create a vector of pancakes could be more efficient using less cycles and less memory to achieve the same results.

**Instructions**:

1. Click anywhere in the splash screen
2. Select new game, instructions, or exit.
3. If new game is clicked, input username and level.
4. Click on the pancake you want to flip. Clicking on the pancake will rotate that pancake and the pancakes above it.
5. The game will conclude once the pancakes are stacked in order starting from the bottom from largest to smallest using the least flips possible.

**Source Code:**

### Base\_screen.ccp

#include "Base\_screen.h"

using namespace Graph\_lib;

//initiate all private variables

Base\_screen::Base\_screen(Point xy, int w, int h, const string& title):

Window(xy, w, h, title),

//pcks(Point(w/2,h-50)),

back\_button(Point(555,455),100,25,"Back",

[](Address, Address pw) { reference\_to<Base\_screen>(pw).back\_pressed();}),

start\_button(Point(275,125),165,45,"New Game",

[](Address, Address pw) {reference\_to<Base\_screen>(pw).start\_pressed();}),

hc\_button(Point(275,225),165,45,"Credits/Help",

[](Address, Address pw) {reference\_to<Base\_screen>(pw).hc\_pressed();}),

exit\_button(Point(275,325),165,45,"Exit",

[](Address, Address pw) {reference\_to<Base\_screen>(pw).exit\_pressed();}),

main\_background(Point(0,0),"pancakepro-menu.jpg"),

hc\_background(Point(0,0),"pancakepro-credits.jpg"),

start\_pushed(false)

{ // attach hc\_screen

attach(back\_button);

attach(hc\_background);

// hide buttons aren't for main menu

}

void Base\_screen::main\_show()

{

attach(start\_button);

attach(hc\_button);

attach(exit\_button);

attach(main\_background);

}

void Base\_screen::main\_hide()

{

detach(main\_background);

detach(start\_button);

detach(hc\_button);

detach(exit\_button);

}

void Base\_screen::hc\_show()

{

back\_button.show();

attach(hc\_background);

}

void Base\_screen::other\_hide()

{

back\_button.hide();

detach(hc\_background);

}

bool Base\_screen::wait\_for\_button()

{

show();

start\_pushed = false;

#if 1

while (!start\_pushed) Fl::wait();

Fl::redraw();

#else

Fl::run();

#endif

return start\_pushed;

}

### Base\_screen.h

#ifndef BASE\_SCREEN\_GUARD

#define BASE\_SCREEN\_GUARD 1

#include "Window.h"

#include "Graph.h"

#include "PancakeStack.h"

#include "std\_lib\_facilities\_4.h"

#include "GUI.h"

using namespace Graph\_lib;

struct Base\_screen: Graph\_lib::Window{

Base\_screen(Point xy, int w, int h, const string& title);

bool wait\_for\_button();

bool start\_button\_pushed() { return start\_pushed;} //indicates if a name is entered (optional)

//and level is selected;

//----------------------------------------------------------------------------

//temporarily undefined due to the lack of

//support of universal initializer {} in VS2012

private:

bool start\_pushed;

Image hc\_background; //for credits & help screen

Image main\_background; //background for main menu

Button back\_button; //go back to main menu, not visible at main menu

//buttons of main menu

Button start\_button;

Button hc\_button; //help & credits button

Button exit\_button;

//----------------------------------------------------------------------------

//actions when buttons are pressed

void back\_pressed() { other\_hide(); main\_show();}

void start\_pressed() { hide(); start\_pushed = true;}

void exit\_pressed() { hide();}

void hc\_pressed() { main\_hide(); hc\_show();}

void main\_show(); //go back to main menu

void main\_hide(); //go to other screens

void hc\_show(); //credits & help screen

void other\_hide(); //hide everything else to go back to main menu

void game\_show(); //show game screen

};

#endif

### Game\_screen.ccp

#include "Game\_screen.h"

using namespace Graph\_lib;

Game\_screen::Game\_screen(Point xy, int w, int h, const string& title, PancakeStack::Difficulty dd):

Window(xy, w, h, title), pcks(Point(x\_max()/2,y\_max()-100)),

quit\_button(Point(0,0),x\_max(),y\_max()," ",

[](Address, Address pw) { reference\_to<Game\_screen>(pw).quit\_pressed();}),

steps(0),

score(0),

win(false),

lose(false),

difficulty\_level(dd),

congrats\_text(Point(220, 100),"Congratulations, You Win!"),

gameover\_text(Point(270,100),"Game Over"),

final\_score\_text(Point(280,y\_max()/5+30), "Your final score: "+ to\_string(score)),

score\_text(Point(550,75), "Current Score: "+ to\_string(100\*difficulty\_level)),

step\_text(Point(50,75),"Steps: 0"),

continue\_text(Point(245,460),"Press anywhere to continue"),

min\_move\_text(Point(230,150),"This could be done in: "),

background(Point(0,0),"pancakepro-pan.jpg")

{

step\_text.set\_font(FL\_TIMES\_BOLD);

step\_text.set\_font\_size(25);

pcks.set\_difficulty(difficulty\_level);

game\_show(); // show all objects on game screen

sorted\_pcks(); // calculate the sorted vector of pancakes

continue\_text.set\_font\_size(20); continue\_text.set\_font(FL\_TIMES\_BOLD);

continue\_text.set\_color(Color::white);

min\_moves = pcks.min\_flip();

}

Game\_screen::~Game\_screen()

{

for (int i=0; i < pancake\_buttons.size(); ++i) {

delete pancake\_buttons[i];

}

}

void Game\_screen::get\_buttons()

{

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[0]\*25,y\_max()-87-21\*0),

pcks.sizes()[0]\*25\*2,10+pcks.sizes()[0]," ", cb\_pancake0));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[1]\*25,y\_max()-87-21\*1),

pcks.sizes()[1]\*25\*2,10+pcks.sizes()[1]," ", cb\_pancake1));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[2]\*25,y\_max()-87-21\*2),

pcks.sizes()[2]\*25\*2,10+pcks.sizes()[2]," ", cb\_pancake2));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[3]\*25,y\_max()-87-21\*3),

pcks.sizes()[3]\*25\*2,10+pcks.sizes()[3]," ", cb\_pancake3));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[4]\*25,y\_max()-87-21\*4),

pcks.sizes()[4]\*25\*2,10+pcks.sizes()[4]," ", cb\_pancake4));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[5]\*25,y\_max()-87-21\*5),

pcks.sizes()[5]\*25\*2,10+pcks.sizes()[5]," ", cb\_pancake5));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[6]\*25,y\_max()-87-21\*6),

pcks.sizes()[6]\*25\*2,10+pcks.sizes()[6]," ", cb\_pancake6));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[7]\*25,y\_max()-87-21\*7),

pcks.sizes()[7]\*25\*2,10+pcks.sizes()[7]," ", cb\_pancake7));

pancake\_buttons.push\_back(new Button(Point(x\_max()/2-pcks.sizes()[8]\*25,y\_max()-87-21\*8),

pcks.sizes()[8]\*25\*2,10+pcks.sizes()[8]," ", cb\_pancake8));

}

void Game\_screen::button\_pressed(int i)

// actions when pancake flip buttons are pressed

{

pcks.flip(i);

//re-position and adjust the widths/heights of all buttons

// so that they will still fit underneath different pancakes

for (int j = 0; j < pcks.total\_pancakes(); ++j)

{

detach(\*pancake\_buttons[j]);

pancake\_buttons[j]->loc = Point(x\_max()/2-pcks.sizes()[j]\*25,y\_max()-87-21\*j);

pancake\_buttons[j]->width = pcks.sizes()[j]\*25\*2;

pancake\_buttons[j]->height = pcks.sizes()[j]/2+15;

attach(\*pancake\_buttons[j]);

}

++steps; //step plus one

step\_text.set\_label("Steps: "+ to\_string(steps));

score\_calc();

score\_text.set\_label("Current Score: " + to\_string(score));

redraw();

win\_check(); //check for win every time buttons are pressed

if (win || lose) {

attach(quit\_button);

final\_score\_text.set\_label("Your Score: " + to\_string(score));

min\_move\_text.set\_label("This could be done in " + to\_string(min\_moves) + " steps");

min\_move\_text.set\_font\_size(20); min\_move\_text.set\_font(FL\_TIMES\_BOLD);

attach(final\_score\_text); attach(continue\_text); attach(min\_move\_text);

}

}

void Game\_screen::sorted\_pcks()

{

vector<int> original; //store the original sizes() vector

for (int i = 0; i < pcks.total\_pancakes(); ++i)

original.push\_back(pcks.sizes()[i]);

sorted = original;

sort(sorted.begin(), sorted.end()); // ascending order (top to bottom)

reverse(sorted.begin(),sorted.end()); // descending order (bottom to top, the way we want)

}

void Game\_screen::win\_check()

{

vector<int> original; //store the original sizes() vector

for (int i = 0; i < pcks.total\_pancakes(); ++i) {

original.push\_back(pcks.sizes()[i]);

}

if (original == sorted && score >= 0) { // if true, player wins, then detach all pancake buttons

for (int k = 0; k < pcks.total\_pancakes(); ++k)

detach(\*pancake\_buttons[k]);

win = true;

attach(congrats\_text);

congrats\_text.set\_font\_size(20);

congrats\_text.set\_font(FL\_TIMES\_BOLD);

}

else if (score < 0) { // player loses when their score is below 0. detach all pancake buttons

for (int k = 0; k < pcks.total\_pancakes(); ++k)

detach(\*pancake\_buttons[k]);

lose = true;

attach(gameover\_text);

gameover\_text.set\_font\_size(20);

gameover\_text.set\_font(FL\_TIMES\_BOLD);

}

}

void Game\_screen::score\_calc()

{

if (steps <= min\_moves)

score = 100\*difficulty\_level;

else if (steps > min\_moves)

score = (100-10\*(steps-min\_moves))\*difficulty\_level;

}

void Game\_screen::game\_show()

{

get\_buttons(); // buttons can only be set after difficulty is set, since difficulty decides the size of buttons

score\_text.set\_label("Current Score: "+ to\_string(100\*difficulty\_level));

for (int i = 0; i < pcks.total\_pancakes(); ++i)

attach(\*pancake\_buttons[i]);

attach(background);

attach(step\_text);

attach(score\_text);

attach(pcks);

}

bool Game\_screen::wait\_for\_button()

{

show();

quit\_pushed = false;

win = false;

lose = false;

#if 1

while (!quit\_pushed) Fl::wait();

Fl::redraw();

#else

Fl::run();

#endif

return quit\_pushed;

}

### Game\_screen.h

#ifndef GAME\_SCREEN\_GUARD

#define GAME\_SCREEN\_GUARD 1

#include "PancakeStack.h"

#include "Graph.h"

#include "Window.h"

#include "GUI.h"

#include "std\_lib\_facilities\_4.h"

using namespace Graph\_lib;

struct Game\_screen: Graph\_lib::Window{

Game\_screen(Point xy, int w, int h, const string& title, PancakeStack::Difficulty dd);

~Game\_screen(); // deletes all pancake flipping buttons after out of scope

bool wait\_for\_button();

bool game\_win() const { return win;}

bool game\_lose() const { return lose;}

bool game\_quit() const { return quit\_pushed;}

int player\_score() const { return score;}

private:

//---------------------------------------------------------------------------

// for win screen

vector<int> sorted;

int steps; //number of steps for the player to complete the game

int score; //player's final score

int min\_moves;

Text congrats\_text;

Text gameover\_text;

Text final\_score\_text;

Text step\_text;

Text score\_text;

Text continue\_text;

Text min\_move\_text;

//---------------------------------------------------------------------------

// for game screen

PancakeStack pcks;

PancakeStack::Difficulty difficulty\_level;

Image background;

bool quit\_pushed; // default false, true if pushed

bool win; // false as default, true if won

bool lose; // false as default, true if lost

Button quit\_button;

vector<Button\*> pancake\_buttons; // store pancake seletors

//----------------------------------------------------------------------------

void get\_buttons(); //initialize all buttons for game screen

//----------------------------------------------------------------------------

//control inversions

void set\_difficulty(PancakeStack::Difficulty dd); //control inversion of difficulty selector buttons

void quit\_pressed() { hide(); quit\_pushed = true; }

void button\_pressed(int i); // actions for pancakes flip buttons

void win\_check(); // check if the player has won

void score\_calc(); // calculate player's final score

void game\_show(); //show game screen

void sorted\_pcks(); // order of sorted pancakes

static void cb\_pancake0(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(0); }

static void cb\_pancake1(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(1); }

static void cb\_pancake2(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(2); }

static void cb\_pancake3(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(3); }

static void cb\_pancake4(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(4); }

static void cb\_pancake5(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(5); }

static void cb\_pancake6(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(6); }

static void cb\_pancake7(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(7); }

static void cb\_pancake8(Address, Address pw)

{ reference\_to<Game\_screen>(pw).button\_pressed(8); }

};

#endif

### Pancake.ccp

#include "Point.h"

#include "Graph.h"

#include "Pancake.h"

#include "std\_lib\_facilities\_4.h"

namespace Graph\_lib

{

void Pancake::draw\_lines() const

{

// fill (a pancake is prefilled)

fl\_color(236,162,77); //fill brown-ish yellow

fl\_pie(point(0).x,point(0).y,w+w-1,h+h-1,0,360);

fl\_color(255,242,164); //fill light yellow

for (int i = 1; i < 20; ++i)

// fill the color of the side of the pancake

{

fl\_arc(point(0).x,point(0).y+i,w+w,h+h,180,360);

}

fl\_color(color().as\_int()); // reset color

if (color().visibility()) {

fl\_color(color().as\_int()); // by default black

fl\_arc(point(0).x,point(0).y,w+w,h+h,0,180);

fl\_color(180,83,38); // brown-ish color

fl\_arc(point(0).x,point(0).y,w+w,h+h,180,360);

fl\_color(color().as\_int()); // by default black

fl\_arc(point(0).x,point(0).y+20,w+w,h+h,180,360);

// the two straight down lines for the height of the 'cylindrical' shaped pancakes

fl\_line(point(0).x,point(0).y+h,point(0).x,point(0).y+h+20);

fl\_line(point(0).x+w+w,point(0).y+h,point(0).x+w+w,point(0).y+h+20);

}

}

}

### Pancake.h

#ifndef PANCAKE\_GUARD

#define PANCAKE\_GUARD 1

#include "Point.h"

#include "Graph.h"

#include "std\_lib\_facilities\_4.h"

namespace Graph\_lib

{

//a piece of pancake made of a ellipse, two straight lines and an arc line

struct Pancake: Shape{

Pancake(Point p, int sz) // center and size

:w(30\*sz), s(sz), h(10+sz) { add(Point(p.x-w,p.y-h)); }

void draw\_lines() const;

Point center() const { return Point(point(0).x+w,point(0).y+h);}

int size() const { return s;}

//void set\_side\_color(Color c);

private:

int w; //width (changed in ratio by size)

int s; //size

int h; //height

};

}

#endif

### pancakepro\_main.cpp

#include "PancakeStack.h"

#include "Base\_screen.h"

#include "Ready\_screen.h"

#include "Game\_screen.h"

#include "Splash\_screen.h"

#include "Scoreboard.h"

#include "find\_solution.h"

using namespace Graph\_lib;

const int xmax = 700;

const int xmid = 700/2;

const int ymax = 500;

void game\_assembler() //assemble all the windows together

{

Splash\_screen win0(Point(100,100),xmax,ymax,"Pancakepro");

win0.wait\_for\_button();

Base\_screen win1(Point(100,100),xmax,ymax,"Main Menu");

if (win0.continue\_button\_pushed())

{

win1.wait\_for\_button();

if (win1.start\_button\_pushed())

{

win1.hide();

Ready\_screen winr(Point(100,100),xmax,ymax,"Ready");

winr.wait\_for\_button();

if (winr.go\_pushed())

{

PancakeStack::Difficulty d = winr.difficulty\_level();

string name = winr.player\_name();

Game\_screen win2(Point(100,100),xmax,ymax,"Game",d);

win2.wait\_for\_button();

if (win2.game\_quit())

{

int score = win2.player\_score();

Scoreboard win3(Point(100,100),xmax,ymax, Record(name,score));

win3.wait\_for\_button();

if (win3.quit\_button\_pushed())

{

game\_assembler(); //recursion, go back to splash screen

}

}

}

}

}

}

int main()

try{

srand(time(0));

game\_assembler();

}

catch(exception& e){

cerr << "exception: " << e.what() << endl;

return 1;

}

catch(...){

cerr << "Some exception" << endl;

return 2;

}

### PancakeStack.cpp

#include "Point.h"

#include "Graph.h"

#include "PancakeStack.h"

#include "std\_lib\_facilities\_4.h"

namespace Graph\_lib

{

bool PancakeStack::is\_sorted()

{

vector<int> temp(s.begin(),s.begin()+n); //temporary vector to store shuffled nums

vector<int> orig = temp; // original part of sizes, up to n numbers of it

sort(temp.begin(),temp.end()); // sort and reverse temp

reverse(temp.begin(),temp.end()); // reverse because a "perfect" stack is from the greatest to smallest

if (orig == temp)

return true;

return false;

}

void PancakeStack::get\_size()

{

s.clear(); //clear sizes to ensure it's empty everytime it's called

for (int i = 1; i <= 9; ++i)

s.push\_back(i);

//make the random number different every time rand() is called

random\_shuffle(s.begin(), s.end());

if (is\_sorted())

get\_size();

}

void PancakeStack::stack\_pancake()

// stack pancakes from bottom to top

{

get\_size();

if (p.size() != 0) {

for (int i = 0; i < p.size(); ++i)

{

delete p[i]; //delete every Pancake object p[i] points to, preventing memory leak

}

p.clear(); //clear the vector preparing for a new stack

}

for (int i = 0; i < n; ++i)

{

p.push\_back(new Pancake(Point(c.x,c.y),s[i])); //create the pancakes at one same center

p[i]->move(0,-21\*i); //move each pancake 21 pixels up from the previous one

}

}

void PancakeStack::set\_difficulty(Difficulty d)

{

n = d; //n is difficulty

stack\_pancake(); //restack

}

void PancakeStack::draw\_lines() const

{

for (int i=0; i < p.size(); ++i)

{

p[i]->draw\_lines(); //use pancake's drawlines to draw each one

}

}

void PancakeStack::flip(int pn)

/\* pn is a number from 0 to 9. The bottom pancake is 0 with

an increment of 1 going up in order \*/

{

int d = p.size()-1-pn; //difference/2

for (int i = p.size()-1; i >= pn; --i)

{

p[i]->move(0,21\*(d-2\*(p.size()-1-i)));

}

reverse(p.begin()+pn, p.begin()+p.size());

// reverse the order of pointers so it can still be drawn from bottom to top

reverse(s.begin()+pn, s.begin()+p.size()); //reverse size vector as well for later use

}

vector<int> PancakeStack::fixed\_sizes() const // for find\_solution

{

vector<int> temp(s.begin(),s.begin()+n);

vector<int> result(n);

for (int i = 1; i <= n; ++i) {

auto it = max\_element(temp.begin(),temp.end()); //iterator of the maximum element

int pos = distance(temp.begin(),it); //index of the maximum element

result[pos] = i;

\*it = 0; //make the last maximum value 0 thus excluding it from the next max\_element

}

return result;

}

int PancakeStack::min\_flip() // the minimum flips needed to beat the game

{

return find\_solution(fixed\_sizes())->size();

}

}

### PancakeStack.h

#ifndef PANCAKESTACK\_GUARD

#define PANCAKESTACK\_GUARD 1

#include "Point.h"

#include "Graph.h"

#include "Pancake.h"

#include "std\_lib\_facilities\_4.h"

#include "find\_solution.h"

namespace Graph\_lib

{

struct PancakeStack: Shape{

//set each difficulty level correspond to the number of pancakes

enum Difficulty: unsigned char {

noob = 2,

easy = 3,

medium = 4,

hard = 5,

advanced = 6,

expert = 7,

extreme = 8,

chuck\_norris = 9

};

PancakeStack(Point cc):

c(cc), n(advanced) { stack\_pancake();}

void set\_difficulty(Difficulty d);

void draw\_lines() const;

void flip(int pn); // flip according to pancake numbers

vector<int> sizes() const {return s;}

vector<int> fixed\_sizes() const; //for find\_solution

int total\_pancakes() const {return n;}

int min\_flip(); // minimum moves to complete a pancake game

private:

Point c; //center of the first pancake (bottom one)

vector<int> s; //size (horizontal edge) of pancake(ellipse) assigned by get\_size()

vector<Pancake\*> p; //store pointers to Pancakes since Shape prevents copying

int n; //number of pancakes (difficulty)

void get\_size(); //assign w a size value

void stack\_pancake();

bool is\_sorted(); //checks if a stack is already a "perfect" stack before game starts, if true, re-shuffle until false

};

}

#endif

### Ready\_screen.cpp

#include "Ready\_screen.h"

using namespace Graph\_lib;

Ready\_screen::Ready\_screen(Point xy, int w, int h, const string& title):

Window(xy,w,h,title),

go\_bool(false),

go\_button(Point(417,192),156,35,"Go!",

[](Address, Address pw) {reference\_to<Ready\_screen>(pw).go\_pressed();}),

name\_box(Point(336,70),243,51,"Please enter your name:"),

difficulty\_out(Point(336,190),42,43, "Difficulty level:"),

level\_menu1(Point(105,347),161,33,Menu::horizontal,"Difficulty Levels"),

level\_menu2(Point(105,391),161,33,Menu::horizontal, "Difficulty Levels"),

level\_menu3(Point(186,438),162,33,Menu::horizontal, "Difficulty levels"),

d(PancakeStack::Difficulty::medium) //default medium level

{

get\_images();

attach(go\_button);

attach(name\_box);

attach(difficulty\_out);

attach\_levels(); //create and attach level\_menu1/2 's buttons

}

Ready\_screen::~Ready\_screen()

{

for (int i = 0; i < background.size(); ++i)

delete background[i];

}

void Ready\_screen::attach\_levels()

{

level\_menu1.attach(new Button(Point(0,0),0,0,"noob(2)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::noob);}));

level\_menu1.attach(new Button(Point(0,0),0,0,"easy(3)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::easy);}));

level\_menu1.attach(new Button(Point(0,0),0,0,"medium(4)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::medium);}));

level\_menu2.attach(new Button(Point(0,0),0,0,"hard(5)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::hard);}));

level\_menu2.attach(new Button(Point(0,0),0,0,"advanced(6)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::advanced);}));

level\_menu2.attach(new Button(Point(0,0),0,0,"expert(7)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::expert);}));

level\_menu3.attach(new Button(Point(0,0),0,0,"extreme(8)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::extreme);}));

level\_menu3.attach(new Button(Point(0,0),0,0,"Chuck Norris(9)",

[](Address, Address pw){reference\_to<Ready\_screen>(pw).set\_difficulty(PancakeStack::Difficulty::chuck\_norris);}));

attach(level\_menu1);

attach(level\_menu2);

attach(level\_menu3);

}

void Ready\_screen::get\_images()

{

background.push\_back(new Image(Point(0,0),"gamestart-00.jpg"));

background.push\_back(new Image(Point(336,0),"gamestart-336x.jpg"));

background.push\_back(new Image(Point(336,120),"gamestart-336x120.jpg"));

background.push\_back(new Image(Point(336,232),"gamestart-336x233.jpg"));

background.push\_back(new Image(Point(378,188),"gamestart-380x190.jpg"));

background.push\_back(new Image(Point(578,70),"gamestart-578x70.jpg"));

//attach all of them

for (int i = 0; i < background.size(); ++i)

attach(\*background[i]);

}

void Ready\_screen::set\_difficulty(PancakeStack::Difficulty dd)

{

d = dd;

stringstream ss;

ss << dd;

difficulty\_out.put(ss.str());

}

void Ready\_screen::go\_pressed()

{

hide();

go\_bool = true;

}

bool Ready\_screen::wait\_for\_button()

{

show();

go\_bool = false;

#if 1

while (!go\_bool) Fl::wait();

Fl::redraw();

#else

Fl::run();

#endif

return go\_bool;

}

### Ready\_screen.h

#ifndef READY\_SCREEN\_GUARD

#define READY\_SCREEN\_GUARD

#include "PancakeStack.h"

#include "Graph.h"

#include "Window.h"

#include "GUI.h"

#include "std\_lib\_facilities\_4.h"

using namespace Graph\_lib;

struct Ready\_screen: Graph\_lib::Window{

Ready\_screen(Point xy, int w, int h, const string& title);

bool wait\_for\_button();

bool go\_pushed() const { return go\_bool;}

PancakeStack::Difficulty difficulty\_level() const { return d;}

string player\_name() { return name\_box.get\_string();}

~Ready\_screen();

private:

// outputs

bool go\_bool;

string name;

PancakeStack::Difficulty d;

In\_box name\_box;

Out\_box difficulty\_out; //display current difficulty level

Button go\_button;

vector<Image\*> background; // all background images

void get\_images(); // initialize all image objects

Menu level\_menu1; //first row

Menu level\_menu2; //second row

Menu level\_menu3; //third row

void attach\_levels(); //attach all level selectors

void go\_pressed();

void set\_difficulty(PancakeStack::Difficulty dd); //control inversion of difficulty selector buttons

};

#endif

### Record.cpp

#include "Record.h"

namespace Graph\_lib

{

bool Record::operator>(const Record& r1) const

{

return score() > r1.score();

}

bool Record::operator<(const Record& r1) const

{

return score() < r1.score();

}

string Record::string\_record() const

{

ostringstream ost;

ost << n << "\t" << s;

return ost.str();

}

ostream& operator<<(ostream& os, const Record& r)

{

return os << r.name() << " , " << r.score() << endl;

// output format is "Name , Score"

}

istream& operator>>(istream& is, Record& rr)

{

string name;

int score;

char ch1;

is >> name >> ch1 >> score; // i.e. Tom , 100

if (!is) return is;

if (ch1 != ',')

is.clear(ios\_base::failbit);

rr = Record(name, score);

return is;

}

}

### Record.h

/\*

A Record is a user defined object that contains 2 basic

objects, a string for name and an integer for score.

It also has overloaded comparison and iostream operators

which make std::sort() able to sort a vector of Records

as well as keep reading from and writing to a file simpler.

\*/

#ifndef RECORD\_GUARD

#define RECORD\_GUARD 1

#include "std\_lib\_facilities\_4.h"

namespace Graph\_lib

{

// a record contains a name and a score

class Record{

string n; //name

int s; //score

public:

Record(): n("AAA"), s(0) { } //default record

Record(string nn, int ss): n(nn), s(ss) { }

string name() const { return n;}

int score() const { return s;}

string string\_record() const;

// overload > and < operators for std::sort() function to work

// only compare score values

bool operator>(const Record& r1) const;

bool operator<(const Record& r1) const;

};

ostream& operator<<(ostream& os, const Record& r);

istream& operator>>(istream& is, Record& rr);

}

#endif

### Scoreboard.cpp

#include "Scoreboard.h"

using namespace Graph\_lib;

Scoreboard::Scoreboard(Point xy, int w, int h, const Record& rr):

Window(xy, w, h, "Scoreboard"),

file("Scoreboard.txt"), r(rr),

background(Point(0,0),"pancakepro-hs.jpg"),

rcds(5), // Maximum only 5 records

quit\_button(Point(575,445),100,35," ",cb\_quit),

quit\_bool(false)

{

file >> rcds[0] >> rcds[1] >> rcds[2] >>

rcds[3] >> rcds[4]; // read from file to assign every record

file.close(); // close the file

attach(background); // attach background first

text\_records();

attach(quit\_button);

}

Scoreboard::~Scoreboard()

{

for (int i=0; i < text\_names.size(); ++i)

delete text\_names[i];

for (int j=0; j < text\_scores.size(); ++j)

delete text\_scores[j];

}

void Scoreboard::new\_records()

// called by text\_records()

{

sort(rcds.begin(), rcds.end()); //sort records, low score to high score

bool q = false; //qualification for the scoreboard

for (int i = 0; i < rcds.size(); ++i) {

if (r > rcds[i]) // r is the new record awaiting for qualification checking

q = true;

}

if (q == true) {

rcds[0] = r; // drop the lowest score

ofstream ofs("Scoreboard.txt");

ofs << rcds[0] << rcds[1] << rcds[2] << // save the new data

rcds[3] << rcds[4];

ofs.close();

sort(rcds.begin(), rcds.end()); // re-sort the records

}

}

void Scoreboard::text\_records()

{

new\_records();

for (int i = rcds.size()-1; i >= 0; --i){

text\_names.push\_back(new Text(Point(x\_max()/3,y\_max()/4+60\*(4-i)), rcds[i].name()));

text\_scores.push\_back(new Text(Point(x\_max()\*2/3,y\_max()/4+60\*(4-i)),to\_string(rcds[i].score())));

text\_names[4-i]->set\_font(FL\_BOLD); text\_scores[4-i]->set\_font(FL\_BOLD);

text\_names[4-i]->set\_font\_size(20); text\_scores[4-i]->set\_font\_size(20);

attach(\*text\_names[4-i]); //attach immediately after created

attach(\*text\_scores[4-i]);

}

}

bool Scoreboard::wait\_for\_button()

{

show();

quit\_bool = false;

#if 1

while(!quit\_button\_pushed()) Fl::wait();

Fl::redraw();

#else

Fl::run();

#endif

return quit\_bool;

}

### Scoreboard.h

/\*

Scoreboard is a Window taking Record values from Game\_screen

then display them in Text form. It first reads the saved

Records from "Scoreboard.txt", then compare them with the new

Record value it received from Game\_screen and decide on whether

it should replace the lowest score or not. At last it writes to

the txt file and save it.

\*/

#ifndef SCOREBOARD\_GUARD

#define SCOREBOARD\_GUARD 1

#include "Window.h"

#include "std\_lib\_facilities\_4.h"

#include "Graph.h"

#include "GUI.h"

#include "Record.h"

using namespace Graph\_lib;

struct Scoreboard: Graph\_lib::Window{

Scoreboard(Point xy, int w, int h, const Record& rr);

bool wait\_for\_button();

bool quit\_button\_pushed() { return quit\_bool;}

~Scoreboard(); // destructor, deallocate memory for all Text objects.

private:

vector<Text\*> text\_names; // names stored in Text

vector<Text\*> text\_scores; // scores stored in Text

vector<Record> rcds; // records

Record r; // the new record;

Image background;

ifstream file;

Button quit\_button; // a button with the same size as the screen and quit once pressed

bool quit\_bool;

void quit\_pressed() { hide(); quit\_bool = true;}

void new\_records(); // determines if the new record is a high score and re-arrange in order

void text\_records(); // convert the records to Text's and attach them

static void cb\_quit(Address, Address pw)

{ reference\_to<Scoreboard>(pw).quit\_pressed(); }

};

#endif

### Scoreboard.txt

AAA , 0

AAA , 0

AAA , 0

AAA , 0

AAA , 0

### Splash\_screen

#include "Splash\_screen.h"

using namespace Graph\_lib;

Splash\_screen::Splash\_screen(Point xy, int w, int h, const string& title):

Window(xy, w, h, title),

background(Point(0,0),"pancakepro.jpg"),

continue\_button(Point(0,0),x\_max(),y\_max()," ",

[](Address, Address pw){reference\_to<Splash\_screen>(pw).continue\_pressed();}),

continue\_bool(false)

{

attach(background);

attach(continue\_button);

}

bool Splash\_screen::wait\_for\_button()

{

show();

continue\_bool = false;

#if 1

while (!continue\_button\_pushed())

Fl::wait();

Fl::redraw();

#else

Fl::run();

#endif

return continue\_bool;

}

### Splash\_screen.h

#ifndef SPLASH\_SCREEN\_GUARD

#define SPLASH\_SCREEN\_GAURD 1

#include "GUI.h"

#include "std\_lib\_facilities\_4.h"

#include "Window.h"

using namespace Graph\_lib;

struct Splash\_screen: Graph\_lib::Window{

Splash\_screen(Point xy, int w, int h, const string& title);

bool wait\_for\_button();

bool continue\_button\_pushed() { return continue\_bool;}

private:

bool continue\_bool; //indicates whether continue button is pushed

void continue\_pressed() { hide(); continue\_bool = true;}

Image background;

Button continue\_button;

};

#endif