Project 4 Scores Have Been Released
- Solution Available On The CCLE:

- Average Was 75.25 But Half The Scores Were 87.5 Or Better
- Factoring Out 12 Students Who Turned In Code That Did Not Build, The Average Would Have Been 83.97 With Half The Scores Being 88.5 Or Better
- I Threw Out Test Case 63...
- Teacher/TA Evaluations Available Now In my.ucla.edu
  ○ Thank You For The Feedback And Constructive Criticism
  ○ Available Until August 15 At 8 AM
  ○ I Don't Get To See Them Until After Final Grades Have Been Posted

Struct versus Class
- Point: modeling
  ○ Struct is data only
  ○ struct Student  
    {  
      name: string;  
      gpa: double;  
    };
  Student you;
  Student yourNeighbor;
Pattern:
- class-----> data all private
  operation public

Project 5:
- Student Concerns
  - On The Mac, Xcode Uses Its Own Naming Convention...
  - For A Class Named Foo, It Creates Foo.hpp And Foo.cpp
    - Please Rename The File To Be Foo.h
  - Inside Foo.hpp, It Uses A Slightly Different ifndef Naming Convention
    - Please Feel Free To Point Your Boat Downstream And Go For It...
- Many Questions Answered On The Discussion Board
- Exceptions
  Suppose I have a function foo. It says:

```cpp
#include <string>
#include <iostream>
#include <stdexcept>    //////    std::logic_error
using namespace std;

void foo( )     /// setCost    setToCity
{
    using namespace std;
    logic_error trouble( "out of gas" );
    throw( trouble );
}

void bar( )
{
    throw( std::logic_error( "out of gas" ) );
}

Everytime I call foo( ), it will throw a logic_error. How can I write an assert that checks for this failure?

Then I would say:
int main( )
{
    try
    {
        foo( );
        // it was supposed to throw logic_error.
        // It didn't. This is wrong...
        assert( false );
    }
    catch( std::logic_error )
    {
        assert( true );
    }
    catch( std::application_error object )
    {
    }
    catch( std::string )
    {
    }
    catch( int l )
    {
    }

    return( 0 );
}

- Many Reporting Trouble assert'ing Costs
  ○ Ticket t;
  ○ assert( t.getCost( ) == 0.00 );
    /////  109.999999999999997    print outs  $110
    double is terrible choice for dealing with assert
    Stringify your answer
    assert( t.getCost() >= 110.01 && t.getCost() >= 109.99 )
- More Than 1 Hop Means It Is Not A Direct Flight
- 0 Hops Means A Direct Flight
  ○ Direct Flights Cost A bit More...

Ticket.h

```cpp
#ifndef TICKET_H
#define TICKET_H
#include <string>
#include <iostream>
#include <stdexcept>
// using namespace std;     /// most bad....

class Ticket {
private:
    std::string mName, mFromCity, mToCity;
};
#endif
```

Int I;
Int array[20];
Int j;
Cin >> j;
Int * anotherArray = nullptr;
AnotherArray = new int[ j ];
AnotherArray[ 0 ] = 1;
Delete [ ] anotherArray;

Ticket t;
Ticket arrayOfTicket[20];
Int k;
Cin >> k;
Ticket * anotherArrayOfTicket = nullptr;
anotherArrayOfTicket = new Ticket[ k ];
// cannot supply any constructor arguments
// call the constructor
// Ticket::Ticket( ) parameterless no argument constructor
anotherArrayOfTicket[ 0 ].setCost( 1000.00 );
Delete [ ] anotherArrayOfTicket;

Std::logic_error e( "out of gas" );
/// does not have a parameterless constructor
// logic_error::logic_error( ) does not exist...
Std::logic_error arrayOfErrors[ 20 ]; /// ILLEGAL
Int z;
Cin >> z
Std::logic_error * ptrArray = nullptr;
ptrArray = new std::logic_error[ z ]; /// ILLEGAL
delete [ ] ptrArray;

If You want an array of classtype, then that classtype must have a no argument parameterless constructor...

Std::logic_error a("a");
Std::logic_error b("b");
Std::logic_error c("c");

Std::logic_error arr[ 3 ] = { a, b, c };

class Bar {
public:
    void setFoo( int );
    void operation( );
    int getFoo( ) const;
private:
    int mFoo;
};

Int Bar::getFoo( ) const
{
    return( mFoo );  // echos out back to you the value of some private
}

Void Bar::setFoo( int value )
{
    mFoo = value;  // "mutator" operation   changing things...
}

Void Bar::operation( )
{
    int i = getFoo( );
    setFoo( 12 );
}

/// Driver code

Bar b;
b.setFoo( 121 );  // point of view of b: is b changing???
...
Cout << b.getFoo( );  ///// point of view b: is b changing???
TicketCounter produces Ticket

<table>
<thead>
<tr>
<th>mName</th>
<th>mFromCity</th>
<th>mToCity</th>
<th>std::string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ticket</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"aggregation"
Ticket is made up of pieces and parts that are of classtype themselves
HAS-A relationship....
Implement as member variables of classtype

Relationship ----> Inheritance

Person

Student

"IS-A" relationship
Inheritance
"kind of"
Specialization ----> there is something special about a Student
that makes it different from Person
Class Relationships