Project 4 Scores Have Been Released
- The Average Was 85.392 But Half The Scores Were 91 Or Better With 6 Perfect Scores Of 100!

Project 5 Is Coming Due Wednesday
- The Goal Is FootballTeam.h, FootballTeam.cpp and main.cpp
- Major Hint:
  ```cpp
class FootballTeam {
  public:
    std::string getName();
  private:
    std::string myName;
};
```
  As In:
  ```cpp
  int i=15;
  string s = std::to_string( i );
  assert( s == "15" );
  ```
- Please Use The Sample main.cpp To Start Testing Your Code
- Most Common Reported Problem Is Related To Scoping
  It Is Not:
  ```cpp
  std::string getName( )
  {
    return( myName );
  }
  ```
  It Is:
  ```cpp
  std::string FootballTeam::getName( );
  ```

Game Plan:
- Various Topics Today
- Project 6 Discussed And Released Wednesday. Due Next Friday
- Sample Final Review Problems Released Wednesday
- Review Monday
- Final Exam Planned For Wednesday 10-Noon Here In Our Classroom
  ○ No Computer, No Devices
  ○ You Can Bring 2 Pages Of Notes, 8 1/2 x 11 Pages, Printed, Typed Or Scribbled
  ○ Out Of 100 Points
  ○ More Information About It Once I Finish Making It...

- Bank Account class

<table>
<thead>
<tr>
<th>bankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>- my_name : string</td>
</tr>
<tr>
<td>- my_balance : double</td>
</tr>
<tr>
<td>+ bankAccount( )</td>
</tr>
<tr>
<td>+ bankAccount( initName : string, initBalance : double )</td>
</tr>
<tr>
<td>+ withdraw( amount : double ) : void</td>
</tr>
<tr>
<td>+ deposit( amount : double ) : void</td>
</tr>
<tr>
<td>+ balance( ) : double</td>
</tr>
<tr>
<td>+ name( ) : string</td>
</tr>
<tr>
<td>+ setName( name : string ) : void</td>
</tr>
</tbody>
</table>

- But What If Something Goes Wrong....

    // DRIVER CODE...
    bankAccount muffin( "", -100.00 );
    // RIGHT HERE..... Should not continue....

cout << "The bankAccount for " << muffin.getName( ) << " has " << muffin.getBalance( );
muffin.deposit( 10 );
muffin.withdraw( 10000000 );
muffin.withdraw( -100000 );
// CLASS CODE
bankAccount::bankAccount( ) ///// public
{
   // role…. Initialize all your member variables....
   my_name = "";
   my_balance = 0.00;
}

bankAccount::bankAccount( string name, double balance ) ///// public
{
   // role…. Initialize all your member variables....
   my_name = name;
   my_balance = balance;
   if (balance < 0 )
   {
      cerr << "An error happened..." << endl;
   }
}

- Introducing std::logic_error
#include <stdexcept>

std::logic_error
table

- message : string
+ logic_error( message : string )
+ what( ) : string

logic_error carRanOutOfGas( "ran out of gas..." );
cout << carRanOutOfGas.what( ); // returns the message

// BETTER CLASS CODE
bankAccount::bankAccount( string name, double balance )    
public
{
    if (balance >= 0)
    {
        /// role.... Initialize all your member variables....
        my_name = name;
        my_balance = balance;
    }
    else
    {
        /// patterns to how our C++
        /// class code throws
        std::logic_error
            e( "you can't have a balance less than 0!" );
        throw( e );    /// like a return communicating failure
        /// bad...
        cout << e.what( );    /// not reachable
    }
}

/// Better DRIVER CODE...
try
{
    /// driver code s job to catch the mistake...
    bankAccount muffin( "muffin", -100.00 );
    /// fail....
    /// throwing logic_error
    ///    catch logic_error
    ///    die dead in tracks...
    cout << "The bankAccount for " << muffin.getName( )
        << " has " << muffin.getBalance( );
    muffin.deposit( 10 );
    muffin.withdraw( 1000000 );
} catch( logic_error error ) {
    cout << "something went wrong.... " << error.what( ) << endl;
} catch( overdrawnAccount variable ) {
    variable;
} catch( ... ) {
}
try
{
FootballTeam f;
f.gameScore(-100, -1000000, true);
f.gameScore(10, 10, true);
}
catch(logic_error)
{

}

Class FootballTeam
{
Public:
    // undeclared exceptions...
    // once in a lifetime event
    /// regularly expect driver code to aware of
    void gameScore(int us, int them, bool homeGame)
    throw(std::logic_error);
};

void FootballTeam::gameScore(int us, int them, bool homeGame)
throw(std::logic_error)
{
    if (us < 0)
    {
        cout << "the score is wrong... " << endl;
        std::logic_error le("no negative scores...");
        throw(le); /// return communicating failure...
    }
    else if (us > them) /// who won??
POINTER VARIABLES....

Int a = 12;
Int * ptrInt;    /// unusable...
ptrInt = nullptr;    /// test for checking
ptrInt = & a;
Cout << *ptrInt;

a = 13;
/// same thing done.... Using ptr variable ptrInt
*ptrInt = 13;       /// dereference   walks the arrow

// first heap command

int * ptr;
ptr = new int( 12 );    /// dynamic   variable
                /// do not die off until you say so...
*ptr = 13;
/// give this memory back....
delete(   ptr  );
delete ptr;

/// deal:  whatever you new, you must delete

// same example again:  classtype rather than int
bankAccount b;
b.deposit( 100 );
bankAccount * ptr;
ptr = & b;
Ptr   -> deposit( 100 );
(*ptr).deposit( 100 );      // same thing....

///// -> will crash your program if the pointer you arrowing
///// is null or not defined....

bankAccount * b = new bankAccount( "muffin", 100 );
b->withdraw( 100 );
delete b;
delete( b );

///// a set of items in advance how big that blob
int array[ 10 ];

Int I = 10;
Int array[ I ];    // could not be based on a variable...
// illegal

// new and delete break this constraint
///// dynamic array

Int I = 10;
Cin >> I;

///// dynamic array
int i;
Cin >> i;
int * array = new int[ i ];
array[ 0 ] = 12;
array[ 1 ] = 13;
Can Have Array of classtype variable
Not every class can be made into an array
    std::logic_error( string );
    Xstd::logic_error( ); /// not supported...

Passenger p;
Passenger array[ 90 ]; // array of passenger
    /// C++ REQUIRES Passenger::Passenger( ); public

Class Airplane
{
    Public:
        Airplane( int size );
        ~Airplane( ); /// destructor
    Private:
        // Passenger passengers[ 50 ]; /// look the same.... different
        Passenger * passengers; //// pointer variable
            //// the constructor new [ ]
    }

Airplane::Airplane( int size )
{
    passengers = new Passenger[ size ];
}

Airplane::~Airplane( )
{
    /// clean up whatever you new'ed...
    delete [ ] passengers;
}

// driver code....
int main()
{
    Airplane smallPlane( 5 );
    Airplane airbus350( 450 );  // control the size of the passenger
}

LIFECYCLE STEPS
- declare
- init            constructor call  we called ourselves
- use
- dies off

Airplane a;  //-->  Airplane::Airplane( );
Airplane seven47( 400 );  //-->  Airplane::Airplane( int size );

Airplane * ptrAirplane = new Airplane( 10 );  // 10 seats
delete( ptrAirplane );  //  ..........  ~Airplane

// never call this directly yourself....
Airplane::~Airplane( )  // destructor
{
    /// purpose: hook called when your object is about to die off
    /// place clean up
    delete [ ] myPassengers;
}

<table>
<thead>
<tr>
<th>Pass-By-Value</th>
<th>Declared As:</th>
<th>Called As:</th>
<th>&quot;Very Safe&quot; Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void foo(int i);</td>
<td>foo( 12 );</td>
<td>Nothing The Callee Does Affects The Caller's Variable Value</td>
</tr>
<tr>
<td></td>
<td>void foo( int );</td>
<td>foo( 'A' );</td>
<td>The Only Communication</td>
</tr>
<tr>
<td>Bar b;</td>
<td>foo( j );</td>
<td>foo( j+1 );</td>
<td></td>
</tr>
</tbody>
</table>

August 6 Page 9
### Pass-by-Reference

<table>
<thead>
<tr>
<th>Declared As:</th>
<th>Called As:</th>
<th>Means:</th>
</tr>
</thead>
<tbody>
<tr>
<td>void foo( int &amp; a );</td>
<td>foo( i ); foo( j );</td>
<td>&quot;unsafe&quot; can cause some &quot;side effects&quot; &quot;strict&quot;</td>
</tr>
<tr>
<td>void foo( Bar &amp; bar );</td>
<td>Double d; foo( d );</td>
<td>The Caller Can Change The Value Of The Caller's Variable What Needs To Be Passed Is The Exact Right Matching Type l-value Of The Right Type Variable Of The Exact Right Type No Copy The &quot;Actual Thing&quot; Is What Is Being Sent If You Want The Caller To See The Changed Value, This Is The Way You Send INBOUND - OUTBOUND</td>
</tr>
</tbody>
</table>

### Pass-by-const-reference

<table>
<thead>
<tr>
<th>Void foo( const int &amp; i );</th>
<th>Int value = 12; foo( value );</th>
<th>Read only value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/// C++ programmers send objects by value....</td>
<td>Bar b; foo( b );</td>
<td></td>
</tr>
<tr>
<td>/// save a few eye blinks...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/// not copy that argument</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Void foo( const Bar &amp; bar )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/// limited to just the const items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/// bad...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bar.setValue( 323 );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bar.setMessage( &quot;asdf&quot; );</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Class Bar
{
    Public:
        Bar( );

        int getValue( ) const;
        void setValue( int v );

        string getMessage( ) const;
        void setMessage( string s );

        void operation( int v, string s );

    Private:
        int value;
        string message;
        bool flag;

};