Be Sure To Turn In A Compileable main( )
Be Sure To Define Each Function, Even If You Implementation Says:
  int locateMinimum( string[] array, int n ) { return( 0 ); }
  void unionNoDuplicates( string array1[ ], int n1, string array2[ ], int n2,
                        string outputArray[], int& outputSize );

string CombinedArray[ n1 + n2 ];
String CombinedArray[ 10000 ];
  // size fixed compile time known amount
  // can't vary at run-time

  // one loop that would walk array 1
  // if (isTheValue I Am Trying To Copy NOT There Already)
  // blindly copy over all the values from array1 into the outputArray
  // one loop that would walk array 2
  // if (isTheValue I Am Trying To Copy NOT There Already)
  // blindly copy over all the values from array2 into the outputArray
  // update the outputSize correctly

- Late Work Accepted Until 7 AM Thursday Morning.
- I Appreciate Your Patience While The TA's And I Score Your Work
- Please Read The Discussion Board Questions
  ○ shiftRight can return either 0 or -1 if n <= 0. The scorer will accept either answer
Project 5 Has Been Released. It Is Due August 8th At 9 PM
Xcode Users: Beware Of The .hpp File. Rename It To .h Please!

Just One More Project Left...
The Final Exam Is Planned For Wednesday, August 15 - 10-Noon
- You Can Bring In 2 Sheets Of Notes
- More On The Exam Once We Get Abit Closer...
- Please Review My Solutions To Project 3 and Project 4 and Project 5, Once They Are Released...

```cpp
class bankAccount
{
    /// private:
    int datamember;
    void operation( );
    public:    ///// access modifiers
        void withdraw( int amount );
        void deposit( int amount );
        double getBalance( );
        void setBalance( int amount );
        string getName( ) const;    {     return( my_name );    }
    /// accessor method
        void setName( string name );    {    my_name = name;   }
    /// mutator method
};
```
private:    /// access modifiers
    string my_name;
    double my_balance;

};

bankAcount::bankAccount( )    //    ///    public
{
    /// role.... Initialize all your member varialbes....
    my_name = "";
    my_balance = 0.00;
}

bankAcount::bankAccount( string name, double balance )    //    ///    public
{
    /// role.... Initialize all your member varialbes....
    my_name = name;
    my_balance = balance;
}

// originally...
string bankAccount::getName( ) const
{
    return( my_Name );
}

enum WeekDay = {  MONDAY, TUESDAY, WEDNESDAY = 50,
                THURSDAY, FRIDAY, SATURDAY, SUNDAY  };
    /// start counting from 0    it will number each value differently

enum Suit = {  HEARTS=10, DIAMONDS=10,
              CLUBS=20, SPADES=20  };
// how you cannot use it....

Suit s = HEARTS;

Switch( s )
{
    case HEARTS:
    case CLUBS:
    case DIAMONDS:
    case SPADES:
}

enum Year = { FRESHMAN, SOPHOMORE, JUNIOR, SENIOR };

/// you....
Year y = FRESHMAN;
y = y + 1;    // seems legal    FRESHMAN->SOPHOMORE
Switch( y )
{
    case FRESHMAN:
        y = SOPHOMORE;
        break;
    case SOPHOMORE:
        y = JUNIOR;
        break;
    case

DRIVER CODE---- Consumer of the class main Class Designer ----
You .h .cpp
Int I;
BankAccount b;
b.setName( "Pixie" );  // b is being changed
String n = b.getName();  // b is in a read-only context
Cout << "the name is: " << b.getName( ) << endl;
b.deposit( 50.00 );
/// BAD!
b.myBalance = 50.00;

////////-----> opening balance:  0.00
//////// BankAccount( );  does not return anything....
           belongs inside a constructor call
          function
          precise name name as the class itself
          returns nothing not even void....
b.deposit( 100 );                //////-----> 100.00
b.withdraw( 50 );                //////-----> 50.00
Cout << "my bank account has " << b.getBalance( ) << " money over..." << endl;

bankAccount yours;
yours.deposit( 100000000 );
yours.setName( "You" );       //// setName adjusts the name member
variable
Cout << yours.getName( );        //// getName returns the name
value
Cout << yours.my_name << endl;    /// not build.....

bankAccount muffin( "muffin", -1000000000 );
//////// throw an exception....
//////// way we can communicate failure to our driver code
Muffin.getBalance( );
Muffin.deposit( 1 );
Muffin.
getName is a "getter" method.... Accessor method
setName is a "setter" method.... Mutator method

bankAccount mine;
mine.deposit( 0 );
mine.setName( "Howard" );

The thing in front of the dot Is a n "Object" a variable of classtype

#include <iostream>
------> cout object what is its class?
ostream_with_assign
------> cin object
istream_with_assign
Cout.setf( ios::showpoint );
Cout.precision( 2 );   /// cout and cin are "Objects"

String s;
s.size();
s.length();   /// s is an object
Constructor Rules
The role of a constructor "purpose" is to initialize all your member variables
    bool false counter 0 string ""
You get the constructor calls that you define
If you don't define any at all, C++ will give you one for free   default parameterless   empty
    always there to complete the lifecycle of an object

<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>

DRIVER CODE:::

bankAccount b;    ///// parameter-less version  no parenthesis at all...
bankAccount break( );    /// no arguments to send, no parenthesis needed
bankAccount yours( "yours", 100 );    /// use ( )  send arguments....
namespace std; // junk drawer
    cout  cin  string  endl

bankAccount not using a namespace declaration
    placed into the unnamed namespace
    create namespace of our own....

namespace cs31
{
    class BankAccount;
}

Car c;             ---->
Auto::Auto( )
c.setMake( "Volkswagen" );        string
    myMake;
c.setModel( "Golf" );            string
    myModel;
c.getMake( );                   --- returns string
    "Volkswagen"
c.getModel( );                   -- returns string
    "Golf"
String getMake( ) const;  /// read-only
Void setMake( string make );

String Car::getMake( ) const  /// read-only operation   echoing back the value of your member
{
    return( myMake );
}

Void Car::setMake( string make )   /// changing    mutating the underlying object
{
    myMake = make;
}

SETTERS can never be marked const
GETTERS we can choose to make them const    ----> we will be marking our getter like operators const

// set operation      ----> "setter"    ----> mutator methods
// get operation      ----> "getter"    ----> accessor methods

int i( 7 );
int * pI = nullptr;
pI = &i;
*pI = 8;

// without that i....
```cpp
int * ptrInt = nullptr;
ptrInt = new int(7);
*ptrInt = 8;        /// I = 8

// important step....
// recycle that memory....
delete( ptrInt );
/// delete ptrInt;

Int n1;
Cin >> n1;
Int n2;
Cin >> n2;
Int array[ n1 + n2 ];   /// doesn't work because n1 and n2 variables
// new and delete can do this...

/// change the equation    dynamic array    new and delete
Int * pVariable;
pVariable = new int[ n1 + n2 ];
pVariable[ 0 ] = 12;
pVariable[ n1 ] = 12;
// very improtant done....
delete (   pVariable   );   /// crash...
delete [ ]  pVariable;    /// dynamic array
```