Midterm Review:

Covers up to functions and it does not include arrays
Project 2 kind of exam
C++ is very lowercase oriented
Errors: many thousands
C in the name Compile errors "spelling mistakes"
   typically flag the line number where it thinks the problem was
making functions
said you were make something and the compiler let you call it and you didn't make it....
LNK2019 ----> linker had trouble creating your executable
   running the program when you try to rebuild it...
   declared: void today();
called it: main { today(); }
   created: void today( int I ) { }
funny looking no source
   declared: void today( int I, int j, int k, double d=12.4, double e=1.356 );
created: void today( int I, int j, int k, double d, double e ) { }

Error and warning

Linker Error- lethal bad must fix
Syntax Warning - bee sting not good
   bat out all our warnings...
Logic Errors just because your code actually builds and runs shallow compliment
   testing important
   assert all these different possibilities

    if (a < b)
      a = 12;
      b = 13;
      c = 14;
    cout << "ae" << a << " b=" << b << " c=" << c << endl;

Variables
   specific rule _[A-Za-z]([0-9][A-Za-z]*)* good: h123 bad 123h *12%^$ 
   lifecycle flow: declare initialize use it dies off
   has a type ------> C++ information space is required int I; on the stack
   C++ is pretty mean: don't initialize it, garbagey value into all your variables
   int I = 12;
   int j( 5 );
   { } scope.... Declared inside a scope
   declare variables even outside a scope altogether

    //// bad
    int bad = 12;
int main()
{
    int I = 12;
    int k = 12;
}

Math Symbols:
+  -  *  /  
+= -= *+= /=  
++  -- 
/  %

double d = 13 / 5;  // place we're putting has no effect....
double floatanswer = 13 / 5.0;
double anotherway = static_cast<double>(13) / 5;
double alternate = double(13) / 5;  /// C way of doing it.... Less safe...
int I = 13 / 5;  /// 2 no errors no warning
int remainder = 13 % 5;  /// 3

int j += k++ - --m;  //// l-values (variable) and r-values (variable, expression, constant)

m = m - 1;  /// order listed
j = j + (k - m);
k = k + 1;

int j += ++k - --m;

k = k + 1;
m = m - 1;
j = j + (k - m);

for (I = 0; I < 10; i++)
{
}

for (I = 0; I < 10; ++i)  /// no noticeable difference
{
}

Unravel all the contractions
++  + 1
--  - 1
+=  _____ = _____ +
One half of a: $\frac{1}{2} a$  \hspace{1cm} \frac{1}{2} * a \hspace{1cm} 1.0 / 2.0 * a$

Number line: \hspace{1cm} <---------|------------------|-------------------> \hspace{1cm} 2 \hspace{1cm} 3

$(2 < x < 3)$  \hspace{1cm} &&  \hspace{1cm} $(2 < x) \hspace{1cm} && \hspace{1cm} (x < 3)$

$(x == 4, 5)$ \hspace{1cm} $(x == 4) \hspace{1cm} || \hspace{1cm} (x == 5)$

$0.5 * a + .75 * b$

string s = "assafdsdfsaddf";
string t = "";
char c = '4';
char bad = 'sdfddf';
char escape = \n; \hspace{1cm} //// endl
char tab = \t; \hspace{1cm} //// tab

Process text in C++

If we use >> to read in a string get the first word terminates on whitespace
\hspace{1cm} >> consecutively leaves behind the whitespace
\hspace{1cm} defaults work perfect consecutively >> to read the data....

Getline whenever we process text
\hspace{1cm} breaks on newline
\hspace{1cm} Cin.ignore( 10000, '\n' ); \hspace{1cm} /// pushes past the newline

Mixing >> and calls to getline

Flow of control choices
\hspace{1cm} convert from one to another

If $(a == 12)$ \hspace{1cm} -------> switch( a ) \hspace{1cm} {
\hspace{1cm} \hspace{1cm} case 12: \hspace{1cm} /// case fixed CONSTANT
\hspace{1cm} \hspace{1cm} case 11:
\hspace{1cm} \hspace{1cm} case 10:
\hspace{1cm} \hspace{1cm} case 1:

Else if $(a == 100)$
\hspace{1cm} b = 100;
\hspace{1cm} // break; bugs

Else if $(a == 50)$
\hspace{1cm} b = 12;
\hspace{1cm} break;

Else
\hspace{1cm} b = a;
\hspace{1cm} break;

default:
\hspace{1cm} b = 1;
\hspace{1cm} break;

}
For (I = 0; I < j; i++) =======>    I = 0;  while( I < j)                       =====>    I = 0;   do
{                                                             {
   foo( );                                                   foo( );
    i++;                                                                      i++;
}                                                              }
} while( I < j );

While( I < j )  ;
{               
   I = I + 1;
}

< > <= =>=

<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>void foo(int i);</td>
<td>foo( 12 );</td>
<td>foo( 'A' );</td>
<td>- Nothing The Callee Does</td>
</tr>
<tr>
<td>void foo( int );</td>
<td>foo( j );</td>
<td>foo( j+1 );</td>
<td>Affects The Caller's Variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- The Only Communication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Between The Caller And The</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Callee Will Be With The Return</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Argument, If Not void And</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Captured By The Caller</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- What Is Sent Is An &quot;R-Value&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Of The Desired Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- What Arrives Is An</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Independent Copy Of The</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Caller's Original Value</td>
</tr>
<tr>
<td>Pass-By-Reference</td>
<td>Declared As: void foo( int &amp; a );</td>
<td>Called As: foo( i ); foo( j ); Double d; foo( d );</td>
<td>Means: &quot;unsafe&quot; can cause some &quot;side effects&quot; &quot;strict&quot; TheCaller 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>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Array Parameter</td>
<td>Declared As: void foo( int array[ ], int size );</td>
<td>Called As: int array[12]; int other[100]; foo( array, 12 ); foo( other, 100 );</td>
<td>Means: The &quot;whole&quot; array gets passed to the function Because the function doesn't know how big it is, we need to send along a companion size parameter to control the loops in the function Kinda Like Pass-By-Reference If any of the elements of the array are changed, the caller &quot;sees&quot; those changed element values.. INBOUND - UPDATEABLE</td>
</tr>
<tr>
<td>Pass-By-Constant</td>
<td>Declared As: void foo( const int &amp; i );</td>
<td>Called As: int i( 12 ); foo( i );</td>
<td>Means: &quot;locked down reference&quot; &quot;strict&quot;</td>
</tr>
</tbody>
</table>
What Gets Sent Is A Reference To The Value BUT IT IS LOCKED DOWN
The Caller Cannot Change The Value Of The Caller's Variable Because It Is Read-Only
It Would Be An Error To Do So... No Copy
INBOUND - READONLY

<table>
<thead>
<tr>
<th>Const Array Parameter</th>
<th>Declared As:</th>
<th>Called As:</th>
<th>Means:</th>
</tr>
</thead>
</table>
| void foo( const int array[ ], int size ); | int array[12]; int other[100]; foo( array, 12 ); foo( other, 100 ); | The "whole" array gets passed to the function
Because the function doesn't know how big it is, we need to send along a companion size parameter to control the loops in the function
Kinda Like Pass-By-Const-Reference
The Function cannot change any of the elements of the array
It would be an error to do so!
INBOUND - READONLY |

Recall This Code To Loop Through A String
```cpp
#include <string>

string s;        //// string is a "class"
                 ///    a blueprint that helps you 'versions'
```
/// of that type
/// variable s s is an "object"
string t, yourString, myString;
s.length();    /// operations are invoked by the dot
guy
s.empty();
s.find( "pattern" );