Overview  
--------  
  
For this assignment you will create a program to count the frequency  
of letters in a file.   
  
As with the last assignment, this will take some time.  There are lots  
of details to figure out.  Give yourself plenty of time to work on it.  
  
Details  
-------  
  
This program will start by asking the user to enter a file name.  Then  
the program will open the file for reading.  The program will examine  
each character in the file and count the number of times that each  
letter appears.  All puncuation and whitespace should be ignored.  The  
letters should be treated case-insensitively.  
  
Once the entire file has been processed then the program should output  
a table like this:  
  
A 10  
B 3

C 2  
D 4  
...  
Z 0  
  
You should do this be creating an array of integers for the counts of  
each letter.  Then read each character in the file.  If it's a letter,  
use toupper to convert it to an uppercase.  Then calculate the correct  
index value based on the letter.  For example A would be 0 and B would  
be 1, etc.  Use that index value with the array and increment that  
count.  
  
Example input  
-------------  
  
You can use the following as example input to your program for  
testing.  Copy the following text into a file and use that for  
testing.  
  
-------

The Greeks engaged in the sea-service were the following. The  
Athenians furnished a hundred and twenty-seven vessels to the fleet,  
which were manned in part by the Plataeans, who, though unskilled in  
such matters, were led by their active and daring spirit to undertake  
this duty; the Corinthians furnished a contingent of forty vessels;  
the Megarians sent twenty; the Chalcideans also manned twenty, which  
had been furnished to them by the Athenians; the Eginetans came with  
eighteen; the Sicyonians with twelve; the Lacedaemonians with ten; the  
Epidaurians with eight; the Eretrians with seven; the Troezenians with  
five; the Styreans with two; and the Ceans with two triremes and two  
penteconters. Last of all, the Locrians of Opus came in aid with a  
squadron of seven penteconters.    
-------  
  
The result from this should be:  
  
A 48      
B 4      
C 17      
D 23      
E 99      
F 11      
G 11      
H 50      
I 48      
K 3      
L 17      
M 9      
N 61

O 26      
P 7      
Q 1      
R 29      
S 45      
T 74      
U 12      
V 9      
W 23      
Y 10      
Z 1      
  
Turning in the assignment  
-------------------------  
  
When turning in the assignment you should turn in a zipfile of the  
folder containing your project.  An easy way to get to this is to  
right-click on the file tab (in the editor pane) and select "Open  
Containing Folder".  Then navigate up to the parent directory that  
contains the \*.sln file.  This is the folder that you need to zip up  
and turn in.  
  
By default this will contain a bunch of other files that Visual Studio  
uses during compilation.  This will make the zip file very large.  One  
easy way to reduce this size is to remove the \*.sdf file (it is in the  
same folder as the \*.sln file).  This is a generated file and can be  
safely removed before creating the zip file.

Top of Form

Bottom of Form

Overview  
--------  
  
This chapter covers a major new topis: arrays.  
  
Arrays are the first "structured data type" that we are covering.  We  
will not focus so much on two dimensional or multidimensional arrays  
because they are of limited general purpose use and the concepts of  
single dimensional arrays are easily extrapolated to them.  
  
Arrays  
------  
  
The example on page 470 (print numbers in reverse) is a good way to  
illustrate the value of arrays.  In particular, after reading through  
the code on page 470, try to think about how this would work if you  
now needed to do the same thing for 100 numbers.  Even if you managed  
to do that, what about 100,000 numbers?  Or an undefined number of  
numbers?

At some point you should be thinking "there must be an easier way".  
There is.  Arrays.  
  
An array is a collection of a fixed number of elements, all the same  
type.  For example, a list of 100 numbers.  There are several aspects  
to arrays:  
  
- declaring array variables  
- accessing array variables (read and write)  
- iterating over an array  
  
We'll look at each one of these.  
  
Declaring arrays  
----------------  
  
You declare an array variable just like declaring a non-array variable  
but with a tiny bit of added syntax.  The following declares an  
integer variable (non-array):  
  
int x;  
  
The following declares and array of 100 integer variables:  
  
int x[100];  
  
Notice the [] and the number of elements.  The type can be any type.  
It is not limited to integers.    
  
Accessing array variables (read and write)  
------------------------------------------  
  
Accessing array elements in an array variable is done using the  
subscript operator [].  You specify the name of the array and the  
element you want:

x[5];  
  
NOTE: ARRAYS ALWAYS START AT LOCATION 0!!!!!!  
  
You can use array subscript access for reading or writing an array  
value:  
  
if (x[5] == 10) {  
  x[5] = 25;  
}  
  
You can also use a variable as a subscript index:  
  
int position = 5;  
if (x[position] == 10) {  
  x[position] = 25;  
}  
  
Iterating over an array  
-----------------------  
  
A very common activity to perform is going through all of the elements  
of an array.  This is frequently done with a for loop.  The following  
goes through the elements of an integer array and sets each value to  
17;  
  
int x[100];  
for (int i=0;i<100;i++) {  
  x[i] = 17;  
}  
  
Loops for proccessing arrays are very very very common. Review the  
examples in the book example 9-3 and make sure you understand them.  
  
Array sizes  
-----------  
  
The size of an array must be known when the array is declared.  C++  
doesn't support dynamically sized arrays.  The following code will NOT  
work:

int size;  
cin >> size;  
int arr[size];  
  
You have to use a real number for the size.  However, it is strongly  
recommended that you use a constant:  
  
const int SIZE = 100;  
int arr[SIZE];  
  
This is helpful because you can now write your for loop like this:  
  
for (int i=0;i<SIZE;i++) ....  
  
If you ever need to change the size, you just have to do it in one  
place.  
  
Array Index Out of Bounds  
-------------------------  
  
Since an array has a given size there are limits to the valid index  
values.  Negative index values are not allowed.  Also, you can't  
request an index value beyond the size of the array:

int x[10];  
x[15] = 66;  // This fails because the valid index values are 0-9.  
  
NOTE: C++ will not automatically detect that you write out of bounds.  
Instead, you will just corrupt a part of your program and cause a  
crash.  
  
Copying arrays  
--------------  
  
You cannot do the following:  
  
int x[10];  
int y[10];  
  
x = y;  
  
In order to copy one array to the other you have to use a for loop  
(like above) to do the copying.  
  
Arrays As Parameters  
--------------------  
  
You can pass array values as parameters to a function.  However,  
arrays are ALWAYS passed by reference.  You do not include the &  
character for this.  It happens automatically.  
  
Base Address of an Array and Array in Computer Memory  
-----------------------------------------------------

This section is important.  It gives a detailed description of how  
arrays are actually stored and accessed in memory.  
  
Arrays As Return Types  
----------------------  
  
Arrays cannot by used as the return type for a function in C++.  
Sorry.  Can't be done.  
  
C-strings (Character Arrays)  
----------------------------  
  
This section is also important for understanding the fundamentals  
about strings in C++.  In terms of practical use, the section on  
"string Type and Input/Output Files" is very important.  This shows  
how to let the user provide a filename for reading/writing files.  
  
Parallel Arrays  
---------------  
  
Another common processing process is the use of "parallel" arrays.  
Two arrays are considered "parallel" if they hold related  
information.  For example, if you want to record a person's height as  
feet and inches you need two variables:  
  
int feet;  
int inches;  
  
If you want an array to hold this info you need two arrays:  
  
int feet[100];  
int inches[100];  
  
The data in these arrays is clearly related. That makes these parallel  
arrays.  
  
Processing parallel arrays can be tedious and error prone.  The index  
values must be kept in sync between the two.  For now this is the best  
we can do but in cs162 you will learn about structures and that will  
allow you to avoid parallel arrays.  
  
Two- and Multidimensional Arrays  
--------------------------------  
  
This section is interesting but it is not essential.