**CSCI 1411 – Lab 06 – PA2 Prep and Visual Studios**

**Goals:**

* Use Visual Studios to compile a program with multiple files
* Provide an outline for CSCI 1410 PA2

**Development Environment:** all students must use Visual Studios 2015

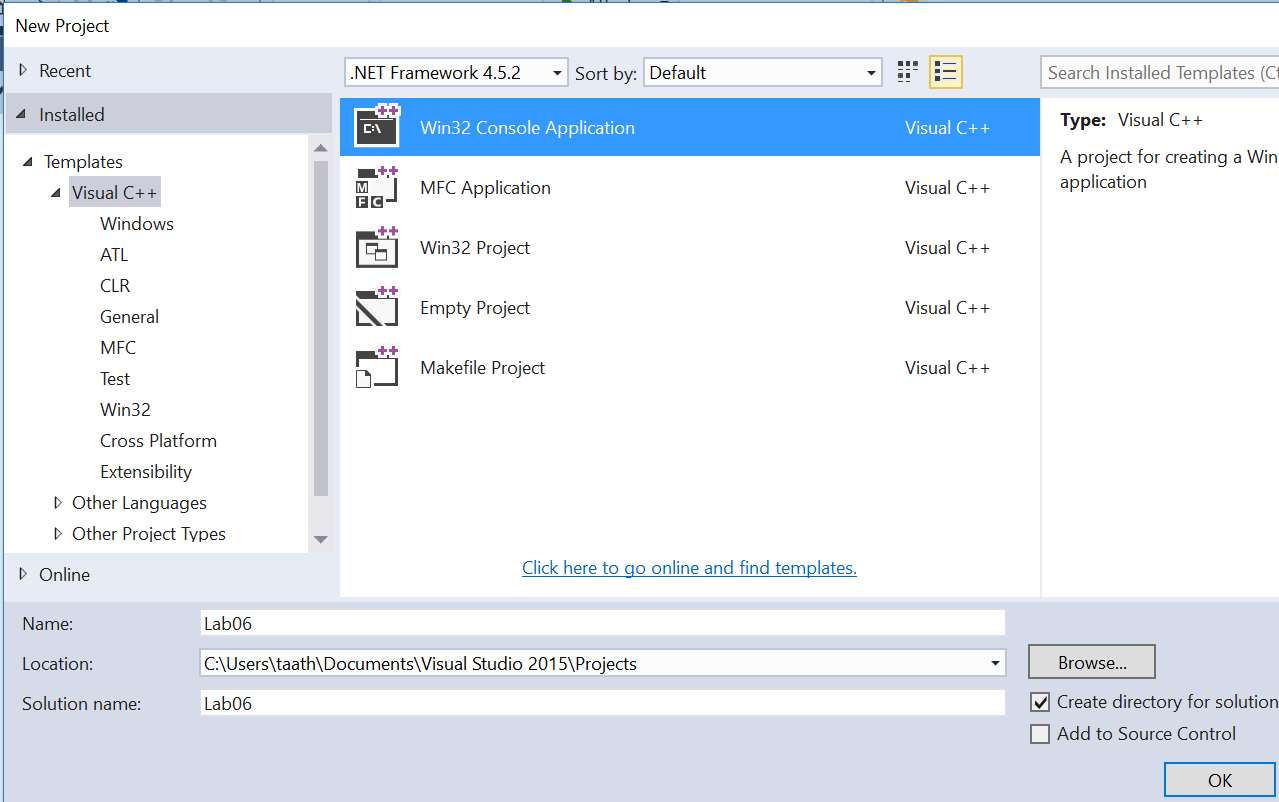
**Skills:** compilation of multiple files in Visual Studios, while loops, functions with multiple files, arrays, structs, use of random variables. (note: arrays, structs just from an English Psuedocode, since we haven’t studied yet).

**Reading**: CSCI 1410 PA2 Prep/CSCI 1410 PA2

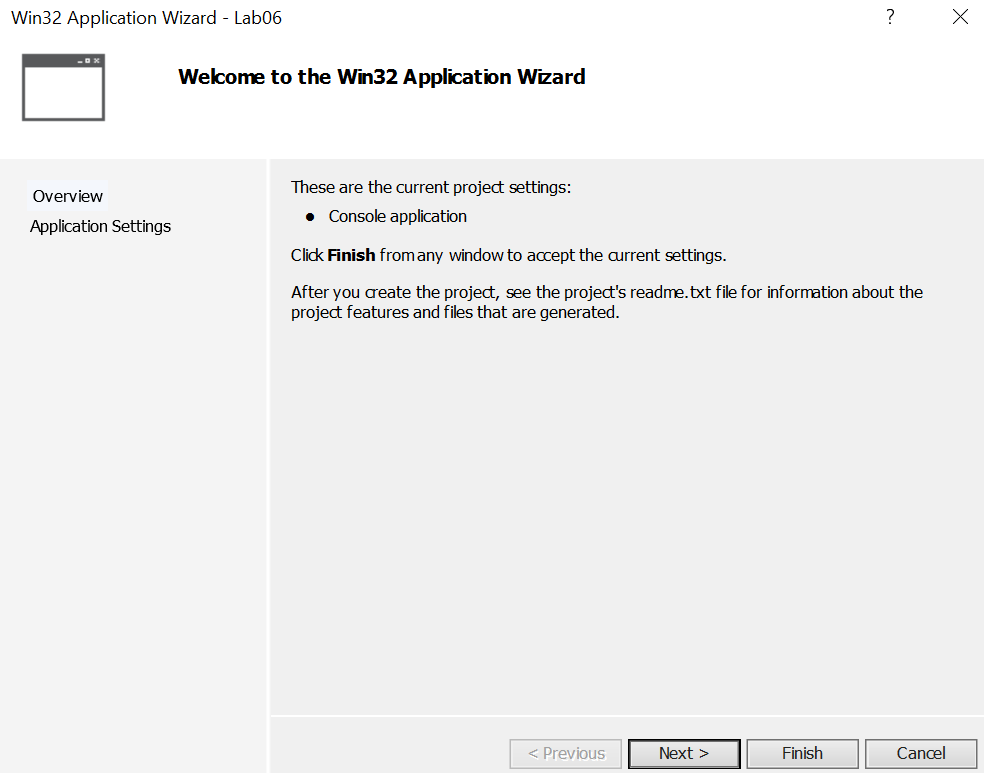
**Deliverables:** 1) This lab with two screen shots 2) lastnameFirstLab06.cpp

**Part I – Skills Practice (10 points)**

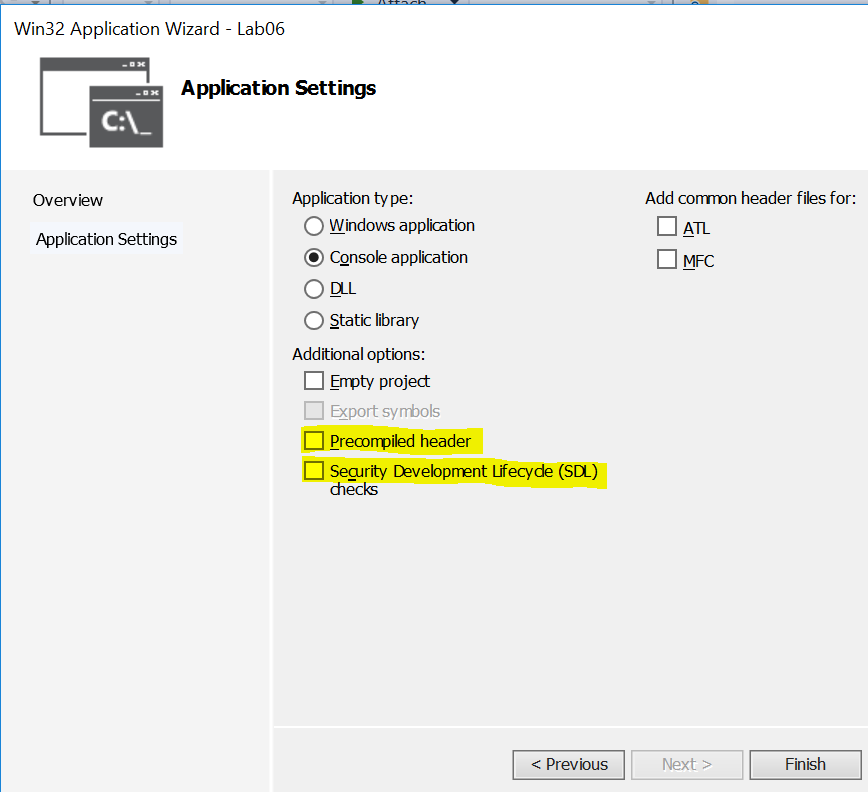
* From a Windows lab machine, click into the Visual Studios 2017
* Open a new project called **lab06**



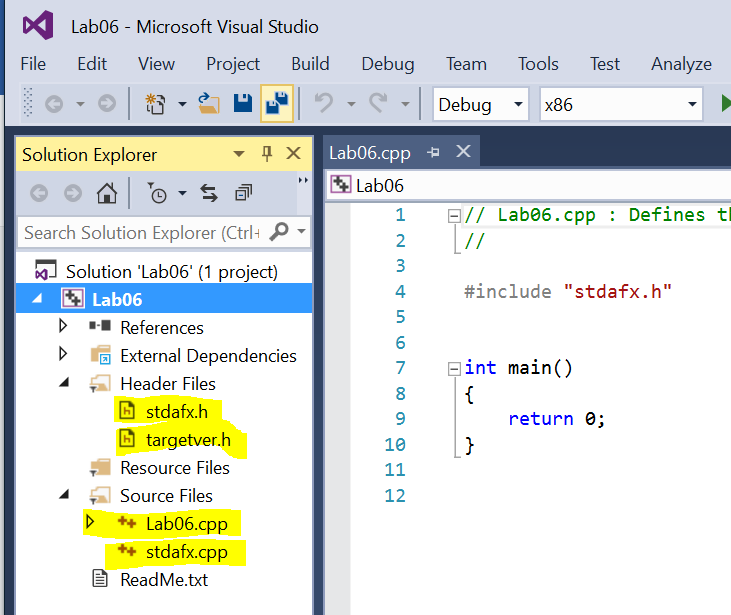
* Click **Next** (Not Finish)

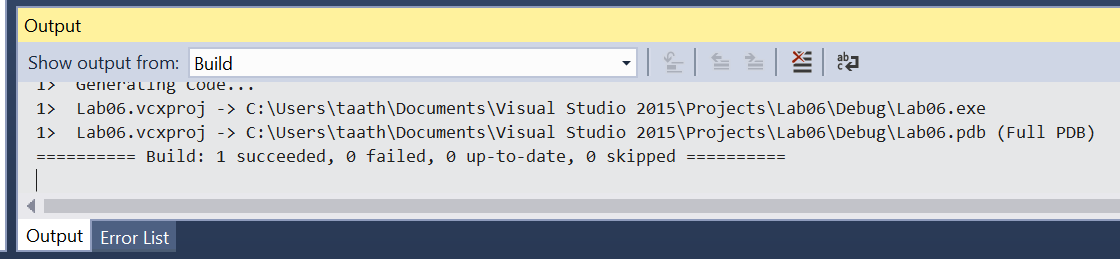


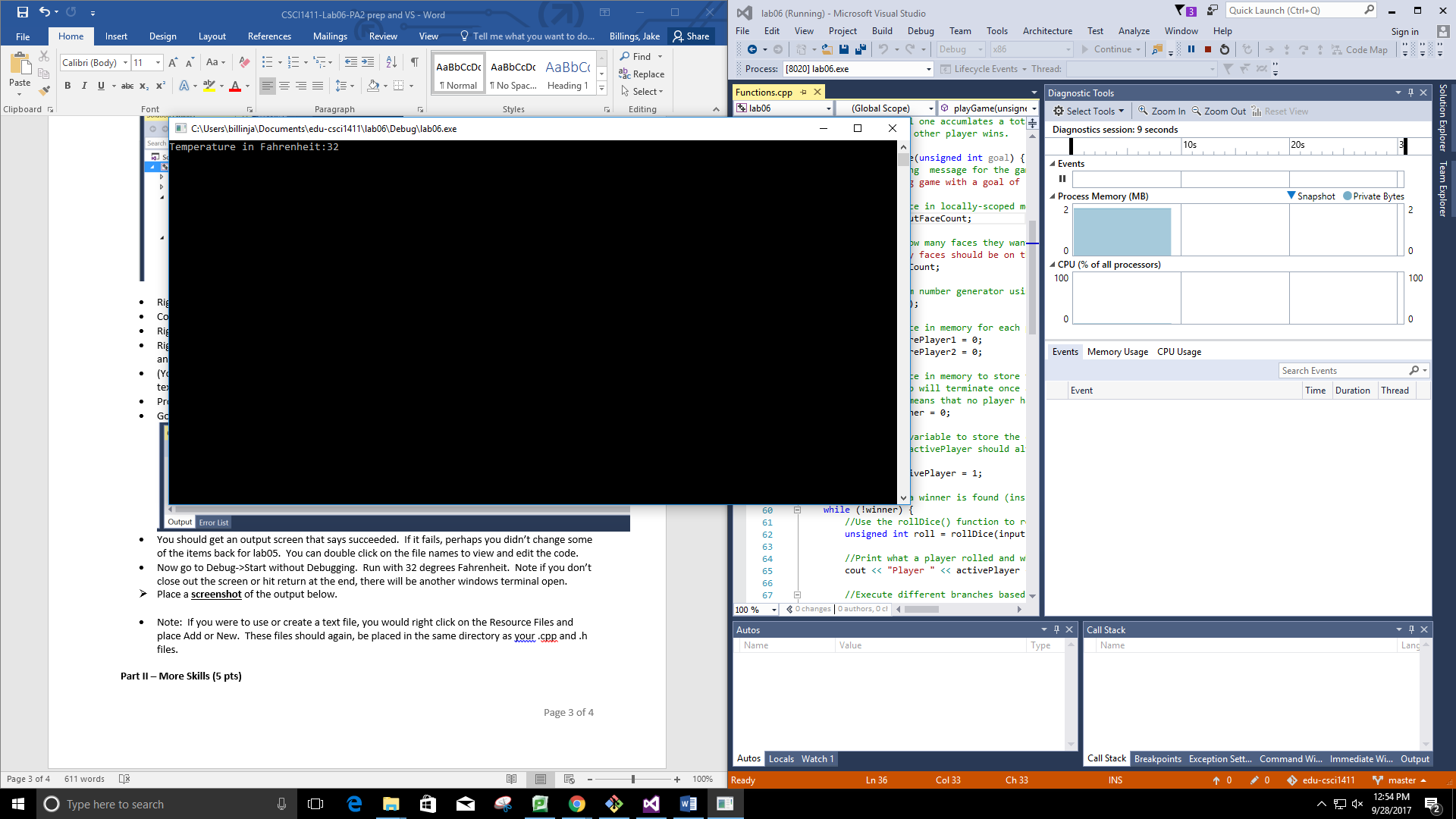
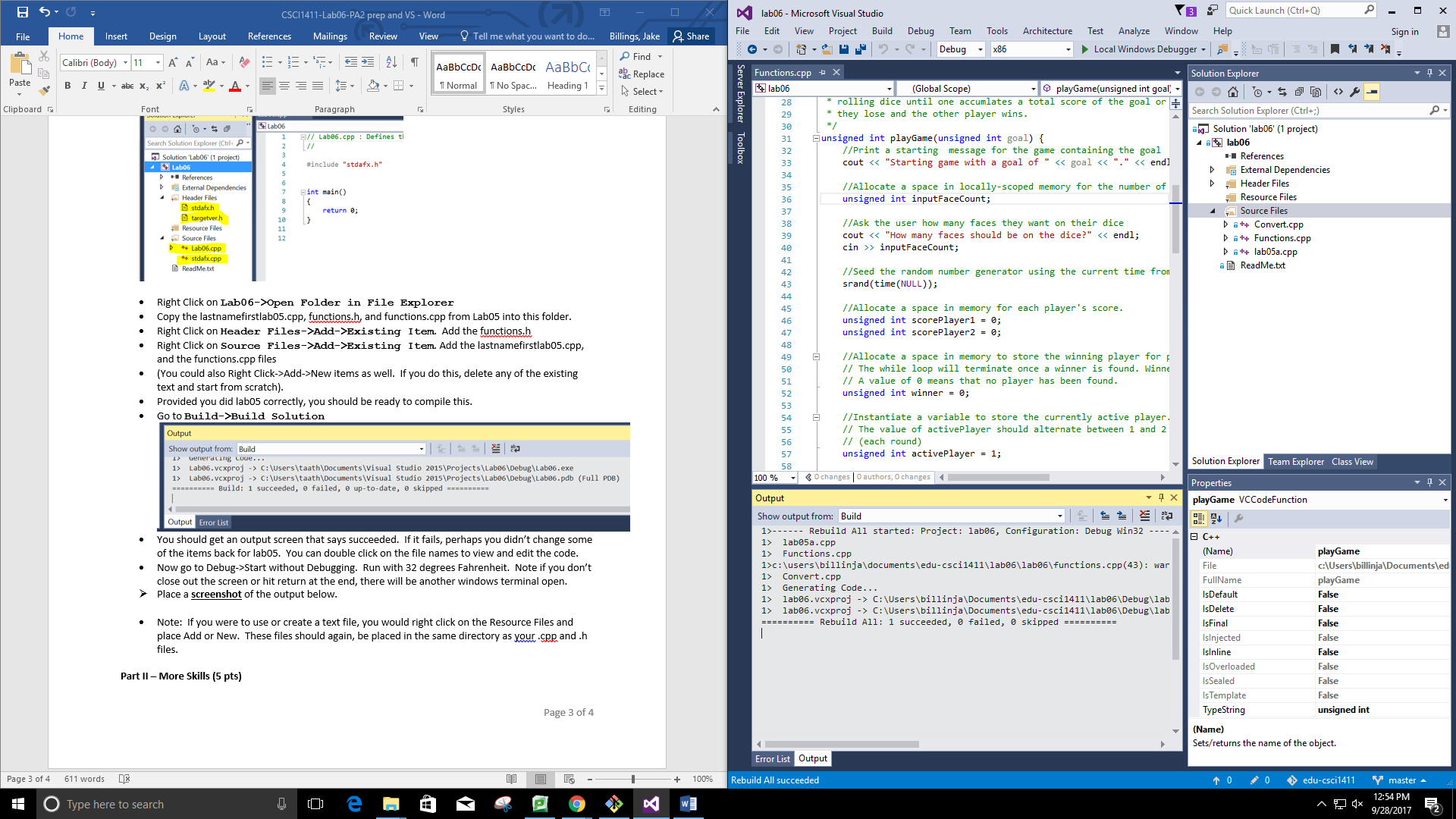
* Click on Next then **Uncheck the precompiled header and Security Development Lifecycle**. This is important, otherwise the code will not compile on a non-Visual Studios compiler. Check the Empty project box



* Go to **View->Solution Explorer**
* Right Click and **Remove->Permanently Delete** the stdafx.h, targetver.h, Lab06.cpp, stdafx.cpp files (if you did not check the empty project box)



* Right Click on **Lab06->Open Folder in File Explorer**
* Copy the lastnamefirstlab05.cpp, functions.h, and functions.cpp from Lab05 into this folder.
* Right Click on **Header Files->Add->Existing Item**. Add the functions.h
* Right Click on **Source Files->Add->Existing Item**. Add the lastnamefirstlab05.cpp, and the functions.cpp files
* (You could also Right Click->Add->New items as well. If you do this, delete any of the existing text and start from scratch).
* Provided you did lab05 correctly, you should be ready to compile this.
* Go to **Build->Build Solution**
* You should get an output screen that says succeeded. If it fails, perhaps you didn’t change some of the items back for lab05. You can double click on the file names to view and edit the code.
* Now go to Debug->Start without Debugging. Run with 32 degrees Fahrenheit. Note if you don’t close out the screen or hit return at the end, there will be another windows terminal open.
* Place a **screenshot** of the output below.

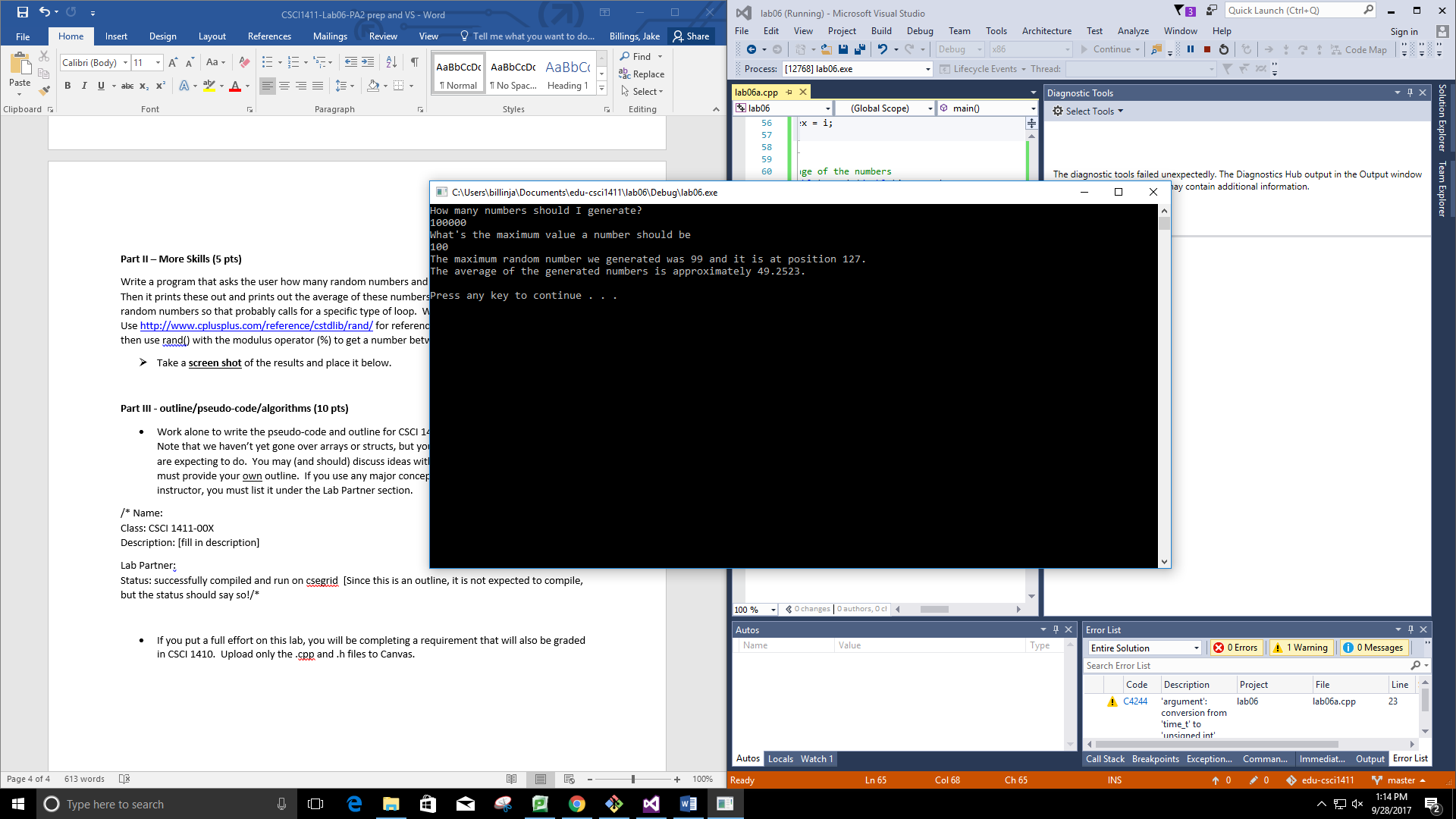


* Note: If you were to use or create a text file, you would right click on the Resource Files and place Add or New. These files should again, be placed in the same directory as your .cpp and .h files.

**Part II – More Skills (5 pts)**

Write a program that asks the user how many random numbers and the maximum random number. Then it prints these out and prints out the average of these numbers. Hint: You know how many random numbers so that probably calls for a specific type of loop. Write this program in Visual Studios. Use <http://www.cplusplus.com/reference/cstdlib/rand/> for reference. You will use srand ONLY once, then use rand() with the modulus operator (%) to get a number between 0 and size -1

* Take a **screen shot** of the results and place it below.



**Part III - outline/pseudo-code/algorithms (10 pts)**

* Work alone to write the pseudo-code and outline for CSCI 1410 Programming Assignment 2. Note that we haven’t yet gone over arrays or structs, but you can put in plain English what you are expecting to do. You may (and should) discuss ideas with your lab partner, but each of you must provide your own outline. If you use any major concepts from someone other than your instructor, you must list it under the Lab Partner section.

/\* Name:  
Class: CSCI 1411-00X  
Description: [fill in description]

Lab Partner:  
Status: successfully compiled and run on csegrid [Since this is an outline, it is not expected to compile, but the status should say so!/\*

* If you put a full effort on this lab, you will be completing a requirement that will also be graded in CSCI 1410. Upload only the .cpp and .h files to Canvas.

//**Main Plan**

//Include Functions

//Include cout

//Include srand

//Use the standard namespace

//Seed the random number generator

//Set a cap on the number of responses we can store that is way bigger than we would ever need

//Keep track of how many responses are in our array

//Allocate an array in memory to store our potential responses that is far bigger than

// we would ever need.

//Print a welcome message

//Keep executing the menu iteration function until it returns false.

//Print a goodbye message

//Return status code 0 because nothing went wrong.

//**Functions plan**

/\*\*

\* Name: Jake Billings

\* Class: CSCI 1411-001

\* Description: Function implementation file for Lab06 (see billingsjlab06.cpp for program description)

\* Status: compiles and runs on VS, Clion, and csegrid

\*/

//Include cout

//Include ifstream

//Include Functions.h

//Use the standard namespace

/\*\*

\* Prints an error message for when a user enters an invalid selection

\*/

/\*\*

\* Reads responses from file into memory

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\* @param maxResponses The maximum number of responses that can fit in the array

\*/

void readResponsesFromFile(Response\* responses, unsigned int\* responseCount, const unsigned int maxResponses) {

//Find out what file to read from

cin.ignore(); //Ignore since we're using getline()

//Default to responses.bin

//Initialize an input stream pointed at the input file

//Initialize a counter to track how many responses we read

//Read the responses until we reach the end of the file or fill up the allocated memory.

while (input.peek() != EOF && (\*responseCount < maxResponses)) {

//Initialize a local copy of the response

//Read into the local copy

//Store the copy in the array

responses[\*responseCount] = r;

//Increment the counter

\*responseCount = \*responseCount+1;

//Close the input stream

//If the array is full, warn the user that it's full and it's possible not all responses loaded.

if (\*responseCount >= maxResponses) {

//Print a message stating we loaded the responses

cout << endl << "Loaded "<< count << " responses. There are now " << \*responseCount << " total responses." << endl;

/\*\*

\* Executes one round of magic eight ball

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\*/

void playGame(Response\* responses, unsigned int\* responseCount) {

//Abort if there are no responses.

if (\*responseCount<1) {

//Pick a random response

unsigned int responseIndex = rand() % \*responseCount;

//Print the response

/\*\*

\* Adds one response for the magic eight ball

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\* @param maxResponses The maximum number of responses that can fit in the array

\*/

void addResponse(Response\* responses, unsigned int\* responseCount, const unsigned int maxResponses) {

//If we've exceeded available memory, print an error message and prevent response creation.

if (\*responseCount >= maxResponses) {

//Allocate a local variable to store incoming data

//Get inputs from user

//Validate the type

//Append it to the array

responses[\*responseCount] = r;

\*responseCount = \*responseCount + 1;

/\*\*

\* Prints all responses in the responses array

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\*/

void printResponses(Response\* responses, unsigned int\* responseCount) {

//Iterate through each response index (0, responseCount)

for (unsigned int i = 0; i < \*responseCount; i++) {

//Print the response

/\*\*

\* Prints all responses in the responses array alphabetically by alphabetizing the array by text in memory then printing it

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\*/

void printResponsesAlphabetically(Response\* responses, unsigned int\* responseCount) {

//Sort the responses in memory using bubble sort by text

//Perform a double iteration over the array

for (unsigned long i = 0; i < \*responseCount; i++) {

for (unsigned long j = 0; j+1 < \*responseCount; j++) {

//Cache the response at index j

//Compare the alphabetical index of responses[j] and responses[j+1] using the > operator

//Swap the responses if they are in the wrong order

//Print the responses in the order they're in in memory, which is now alphabetical

/\*\*

\* Prints all responses in the responses array alphabetically by alphabetizing the array by type in memory then printing it

\*

\* @param responses The array of responses to read into

\* @param responseCount The number of responses already in that array

\*/

void printResponsesByType(Response\* responses, unsigned int\* responseCount) {

//Sort the responses in memory using bubble sort by type

//Perform a double iteration over the array

for (unsigned long i = 0; i < \*responseCount; i++) {

for (unsigned long j = 0; j+1 < \*responseCount; j++) {

//Cache the response at index j

//Compare the alphabetical index of responses[j] and responses[j+1] using the > operator

//Swap the responses if they are in the wrong order

//Print the responses in the order they're in in memory, which is now alphabetical

/\*\*

\* Writes all responses in the responses array to a file

\*

\* @param responses The array of responses to read out of

\* @param responseCount The number of responses already in that array

\*/

void writeResponsesToFile(Response\* responses, unsigned int\* responseCount) {

//Find out what file to read from

cin.ignore(); //Ignore since we're using getline()

//Open an output stream to a file

//For each response, write it to the output stream in the same format as the input file

for (unsigned long i = 0; i < \*responseCount; i++) {

//Close the output stream

/\*\*

\* Deletes a response from the response array

\*

\* @param responses The array of responses to read out of

\* @param responseCount The number of responses already in that array

\*/

void removeResponse(Response\* responses, unsigned int\* responseCount) {

//Allocate a local variable and get user input for which response to delete

//Validate the target index

while (targetIndex >= \*responseCount) {

//Move the array data left in order to overwrite the deleted element and shift all the

// memory elements left

// Start at the source, which is the response after the target of deleting

// Move the data starting at this source to the location of the target overwriting it in the process

// Stop once you've copied the number of bytes equal to the all of the data in the array

(\*responseCount-targetIndex-1)\*sizeof(Response));

//Update the responseCount variable to reflect the shift

\*responseCount = \*responseCount - 1;

/\*\*

\* Executes one iteration of the menu loop.

\*

\* Returns a boolean value that is true if the program should continue execution.

\*

\* @param responses The array of responses to read out of

\* @param responseCount The number of responses already in that array

\* @param maxResponses The maximum number of responses that can fit in the array

\*/

bool menuIteration(Response\* responses, unsigned int\* responseCount, const unsigned int maxResponses) {

//Print the menu options.

//Allocate a space in memory for the user's answer and read it using cin

//Convert the selection to lowercase for easier switch cases

//Run the correct function based on the selection

//Return true because we want to complete the loop again

// We would have returned false if we didn't.

Working Program Screenshot (Finished at home on CLion, but it does run on Visual Studio in the lab)

