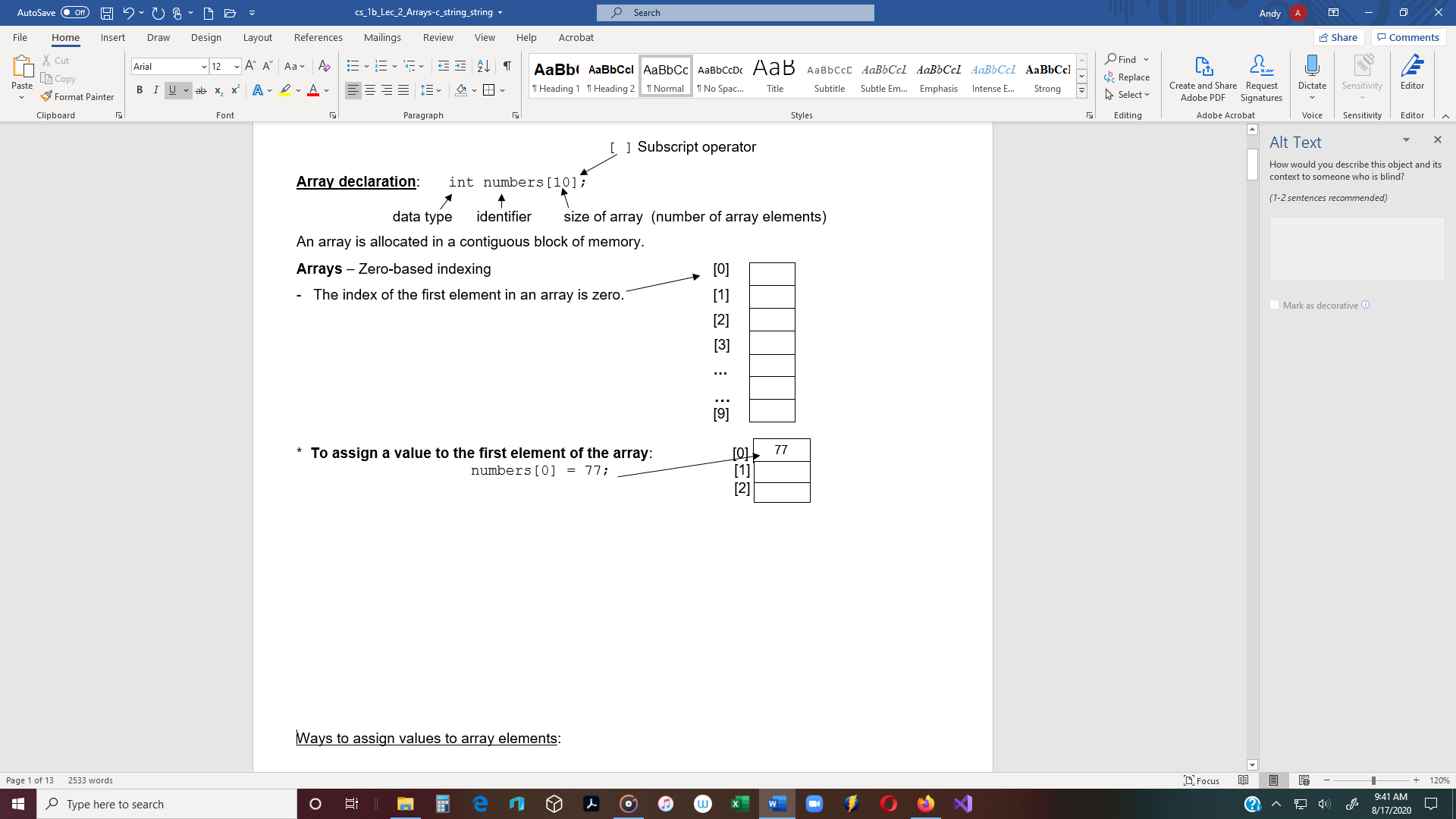
# Lecture #2 – Arrays - strings - c\_strings

**REVIEW - Array** – A structured data type.

**One-Dimensional Array** – A structured collection of components referred by a single name.

* All array components must be of the same data type.
* Each component (array element) is accessed by an index that indicates the

component’s position within the collection.



Ways to assign values to array elements:

1. **Write many assignment statements**

Ex #1: const int SIZE = 5;

double numbers[SIZE];

numbers[0] = 3.3;

numbers[1] = 4.0;

etc.

1. **Use an initialization list.**

Ex #2: double testScores[4] = {99.0,75.5,88.0,91.0};

Ex #3: float deposits[3] = {150.10, 130.00, 75.50};

Ex #4: string cities[3] = {"New York", "Boston", "Phoenix"};

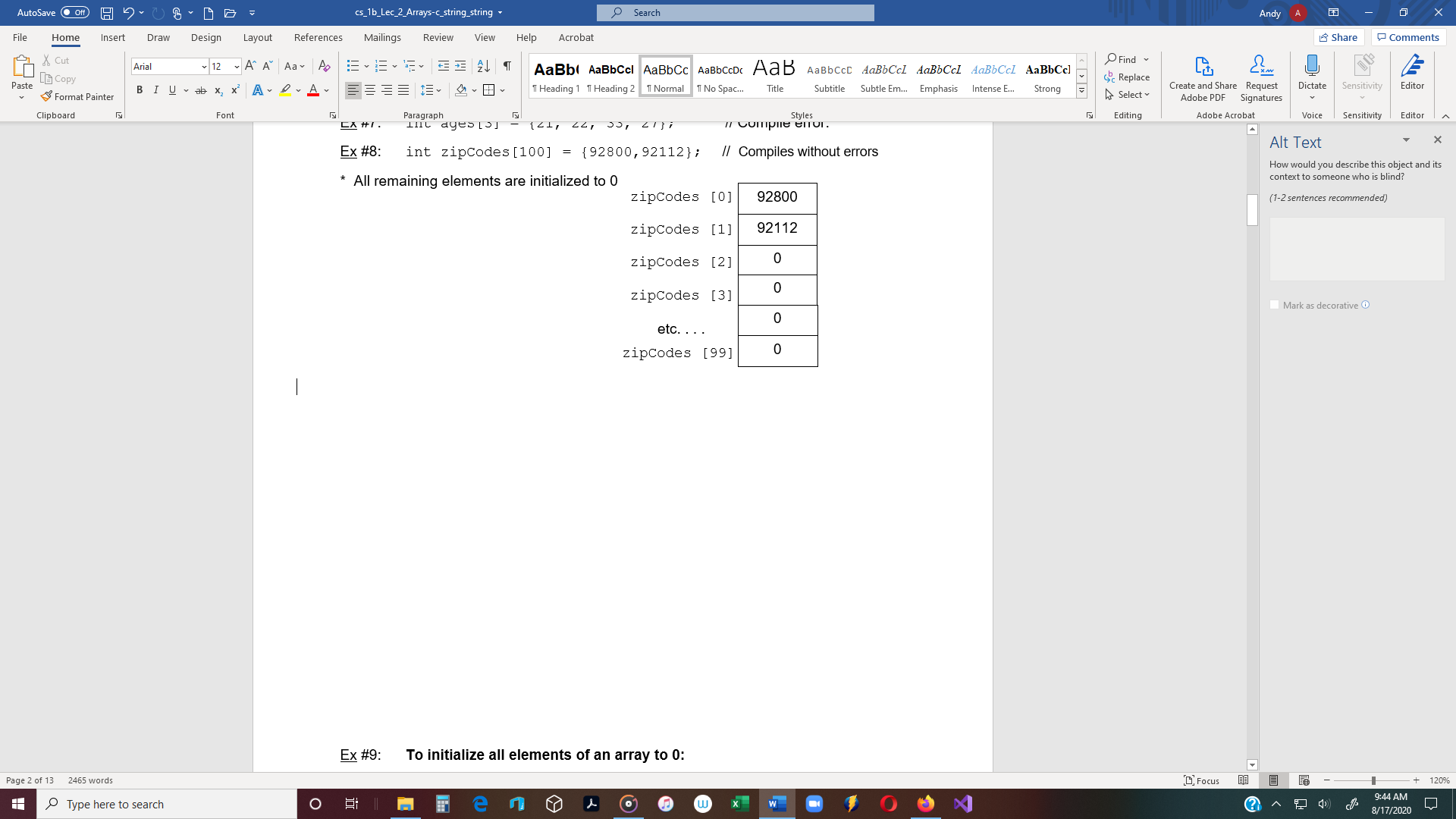
Ex #5: char grades[5] = {'A','C','B','B','A'};

Ex #6: int numbers[ ] = {1,2,3,4,5} // This will compile. The

// compiler assumes size is 5

Ex #7:int ages[3] = {21, 22, 33, 27}; // Compile error.

Ex #8: int zipCodes[100] = {92800,92112}; // Compiles without errors



Ex #9: **To initialize all elements of an array to 0:**

int zipCodes[100] = {0}; // Assigns 0 to all elements

1. **Use a *for loop*  to assign values to an array:**

(*while loops* can also be used.)

Ex #10: Use a *for loop* to assign values to an array or to access values already in an array.

for (int i = 0; i < 10; i++) // 1st for loop to enter the data into the array

{

cin >> numbers[i];

}

for (int i = 0; i < 10; i++) // 2nd for loop outputs the values.

{

cout << numbers[i] << endl;

}

**One array cannot be assigned to another array**

int array1[5] = {1,3,5,7,9};

int array2[5] = {0};

array1 = array2; 🡨 No (Compile error)

**One array cannot be compared to another array** – The following does not work:

if (array1 == array2) 🡨 No (Compile error)

**All numerical array elements cannot be input or output at the same time**

int numbers[5] = {2,6,4,5,8};

cout << numbers; 🡨 Displays the memory address of the first byte of the array

(referred to as the “base address”)

**Arrays and Functions** - Arrays can be passed to functions.

* Arrays are always **passed by reference**.
* Functions receive the **address** of the first element of an array.

Ex #11: **Pass an array to a function - Pass by Reference**

#include <iostream>

using namespace std;

// Include the subscript operator in the formal parameter list.

// 2 Parameters: (1) Address of the array element, and (2) the size of the array.

void displayNames(string anArray[ ], int numItems);

int main()

{

const int SIZE = 3;

string names[SIZE] = {“Mad Hatter”, “Alice”, “Queen of Hearts”};

displayNames(names, SIZE); // Pass the address of the array to a function.

return 0; // subscript operator [ ] not included

}

// === **displayNames Function** ========================================

void displayNames(string names[ ], int numItems)

{

/\* Output

Mad Hatter, Alice, Queen of Hearts

Press any key to continue \*/

for (int i = 0; i < numItems; i++)

cout << names[i] << ", ";

}

// ==================================================================

Note: Technically, when passing an array to a function, only one parameter is required.

* Only the address of the array is required (the first parameter in the above example).
* The size of the array is not a required parameter, but if often needed a function.

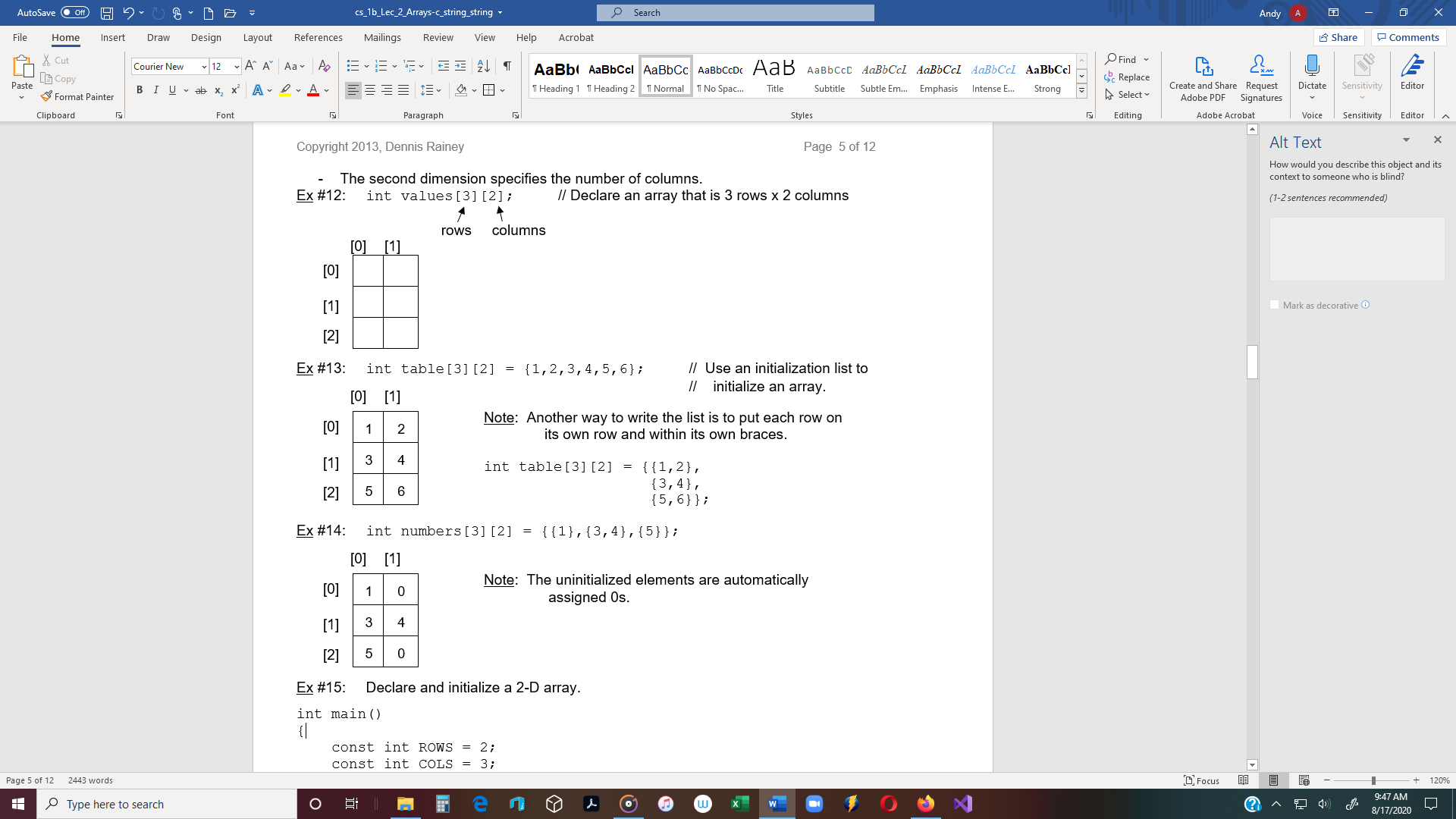
**Two-Dimensional Array** – A collection of components, all of the same data type, structured in

two dimensions (rows and columns)

* Each component is accessed by a pair of indexes that represent the components

position in each dimension.

* The first dimension specifies how many rows.
* The second dimension specifies the number of columns.



Ex #15: Declare and initialize a 2-D array.

int main()

{

const int ROWS = 2;

const int COLS = 3;

// Declare and initialize a 2-Dimensional array of 2 rows and 3 columns

int table[ROWS][COLS] = {{1,2,3}, // row [0]

{6,7,8}}; // row [1]

Ex #16: Use nested ***for loops*** to output all data in a 2-dimensional array.

The **outer *for loop*** determines the row, and the **inner *for loop*** the column.

for (int i = 0; i < ROWS; i++)

{

for (int j = 0; j < COLS; j++)

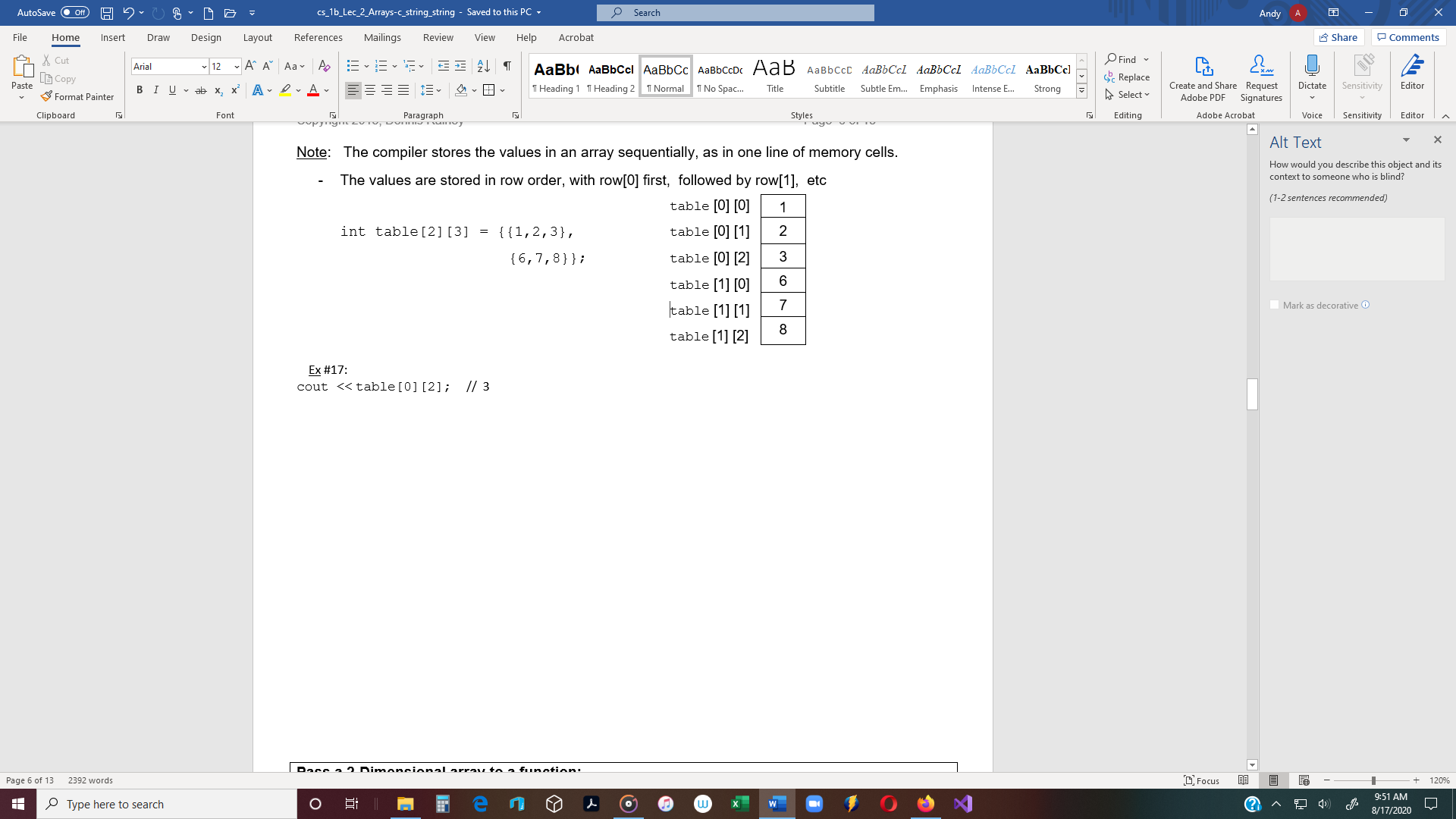
{

cout << table[i][j];

}

cout << endl;

}



Ex #17:

cout << table[0][2]; // 3

**Pass a 2-Dimensional array to a function:**

Note: The compiler stores data sequentially in memory.

* Therefore, the number of columns must be specified in the function prototype.

#include <iostream>

using namespace std;

const int ROWS = 2;

const int COLS = 5; # of columns

void showData(int numbers[ ][COLS]);

int main()

{

int numbers[ROWS][COLS] = {{ 1, 2, 3, 4, 5 }, // row[0]

{ 6, 7, 8, 9, 10 }, // row[1]

showData(numbers);

/\* OUTPUT:

1 2 3 4 5

6 7 8 9 10

Press any key to continue . . . \*/

return 0;

}

void showData(int numbers[ ][COLS])

{

for (int i = 0; i < ROWS; i++)

{

for (int j = 0; j < COLS; j++)

{

cout << numbers[i][j] << '\t';

}

cout << endl;

}

}

**Summing all elements of a 2-D array**

const int NUM\_ROWS = 3;

const int NUM\_COLS = 3;

int total = 0;

int numbers[NUM\_ROWS][NUM\_COLS] = {{5,4,1}, {2,8,7}, {1,3,2}};

for (int row = 0; row < NUM\_ROWS; row++)

{

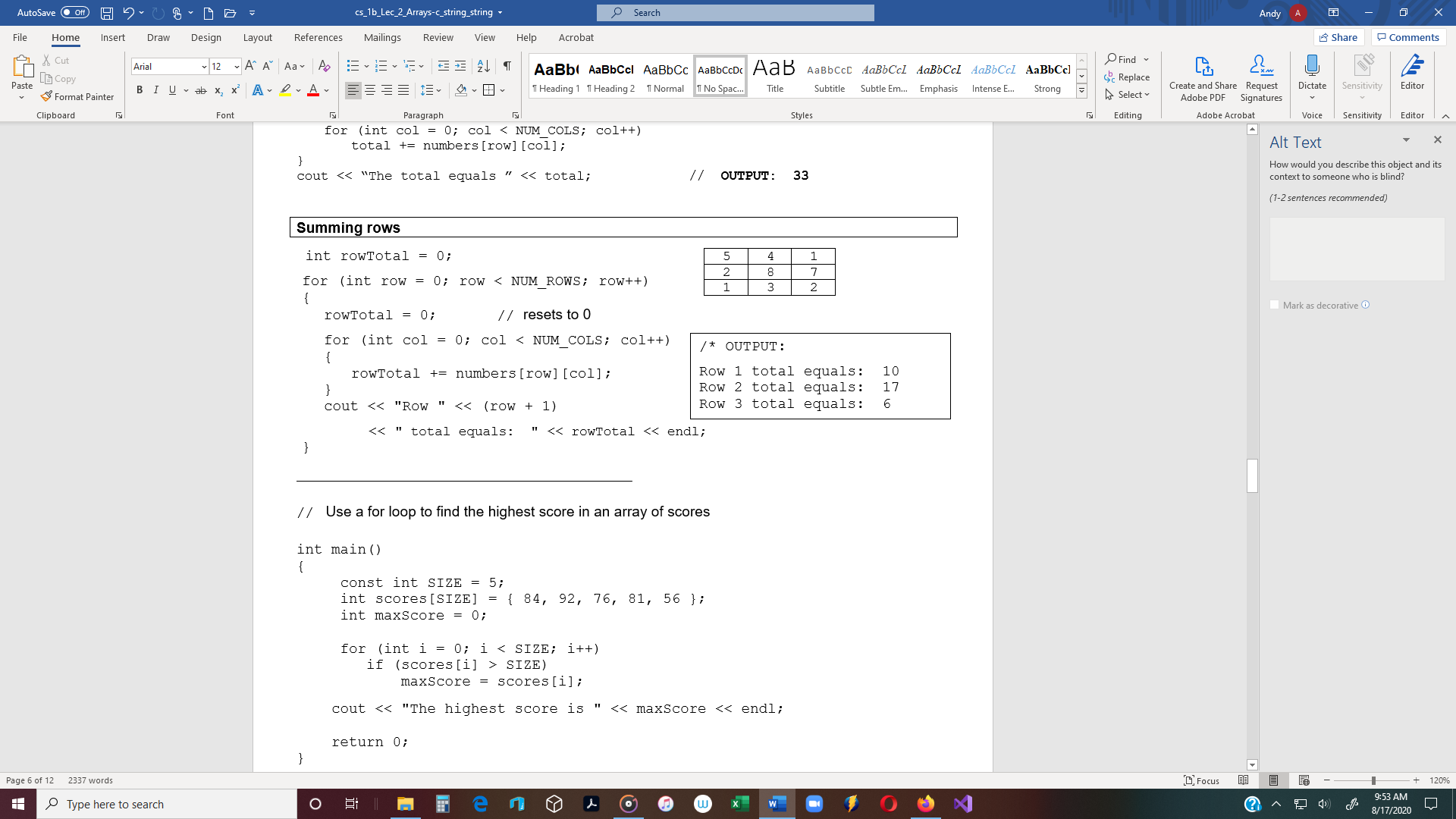
for (int col = 0; col < NUM\_COLS; col++)

total += numbers[row][col];

}

cout << “The total equals ” << total; // **OUTPUT: 33**

**Summing rows**



int rowTotal = 0;

for (int row = 0; row < NUM\_ROWS; row++)

{

rowTotal = 0; // resets to 0

for (int col = 0; col < NUM\_COLS; col++)

/\* OUTPUT:

Row 1 total equals: 10

Row 2 total equals: 17

Row 3 total equals: 6

{

rowTotal += numbers[row][col];

}

cout << "Row " << (row + 1)

<< " total equals: " << rowTotal << endl;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Use a for loop to find the highest score in an array of scores

int main()

{

     const int SIZE = 5;

     int scores[SIZE] = { 84, 92, 76, 81, 56 };

int maxScore = 0;

     for (int i = 0; i < SIZE; i++)

        if (scores[i] > SIZE)

            maxScore = scores[i];

    cout << "The highest score is " << maxScore << endl;

    return 0;

}

**for-each** **loop** (also called a **range-based for** **loop**) - Introduced in 2011.

- Provides another way to iterate through every element in an array (or other list-type structure).

For-each loop is used to access elements of an array quickly without performing initialization,

testing and increment/decrement.

* A for-each loop does something for every array element, rather than doing something n times.

Advantages of Foreach loop:  
1) Makes code more readable.  
2) Eliminates the possibility of programming errors.

***for-each*  syntax:** *variable* ***:*** *array*

#include <iostream>

using namespace std;

int main()

{

     float testScores[5] = { 92.5, 77.0, 88.0, 93.5, 81.0};

     for (float score **:** testScores) // iterate through the array

       cout << score << ‘\t’; // Output one score with each iteration

    return 0;

// Output: 92.5 77.0 88.0 93.5 81.0

}

The variable *score* is not an array index. It is assigned the value of the array element for the current loop. The loop will iterate through each array element, assigning the current array value to the variable.

Note: The variable type should be the same type as the array, otherwise type conversion will occur.

**For each loops and the auto keyword**

Because the variable *score* should have the same type as the array, this is an ideal case in which to use the auto keyword, and let C++ deduce the data type of score.

**The above example, using auto:**

int main()

{

     float testScores[5] = { 92.5, 77.0, 88.0, 93.5, 81.0};

     for (**auto** score **:** testScores) // iterate through the array

       cout << score << ‘\t’; // Output one score with each iteration

**For-each loops and references**

In the for-each examples above, our element declarations are declared by value.

This means that with each iteration, a value in the array is copied into the variable score. Copying array elements can be expensive, and most of the time we really just want to refer to the original element. The solution is to use references to the array and not by value.

int main()

{

     float testScores[5] = { 92.5, 77.0, 88.0, 93.5, 81.0};

     for (**auto** **&**score **:** testScores) // iterate through the array

       cout << score << ‘\t’; // Output one score with each iteration

Each output value will be a reference to the currently iterated array element and no

values are copied.

Note: A change to the variable score will change the array.

Therefore, make the variable score const if you don’t want to change the array values:

int main()

{

     float testScores[5] = { 92.5, 77.0, 88.0, 93.5, 81.0};

     for (const **auto** **&**score **:** testScores) // iterate through the array

       cout << score << ‘\t’; // Output one score with each iteration

Rule: In for-each loops, if the variable is struct or class (not int, float, char, etc.) use references or

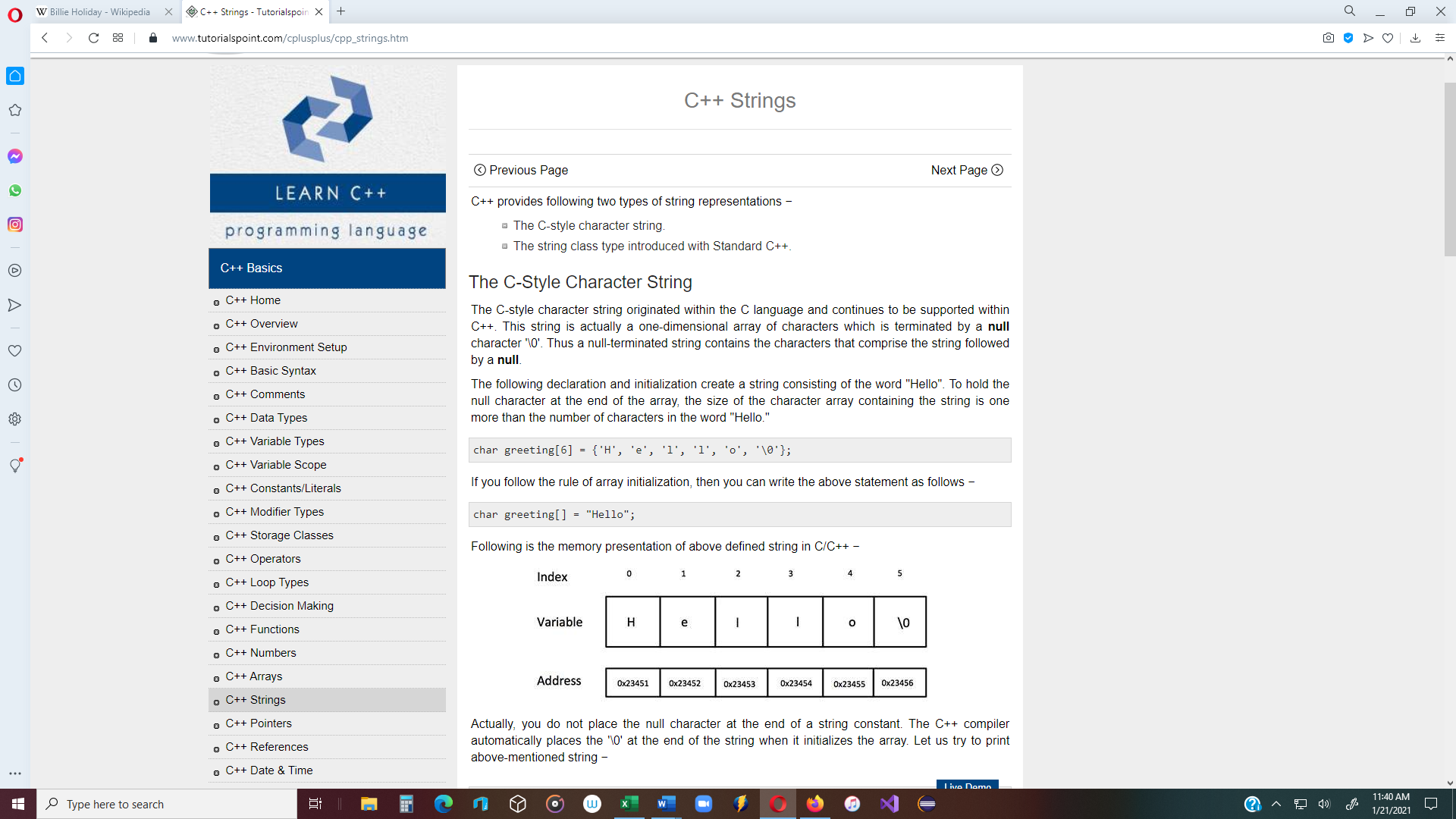
const references for performance reasons.

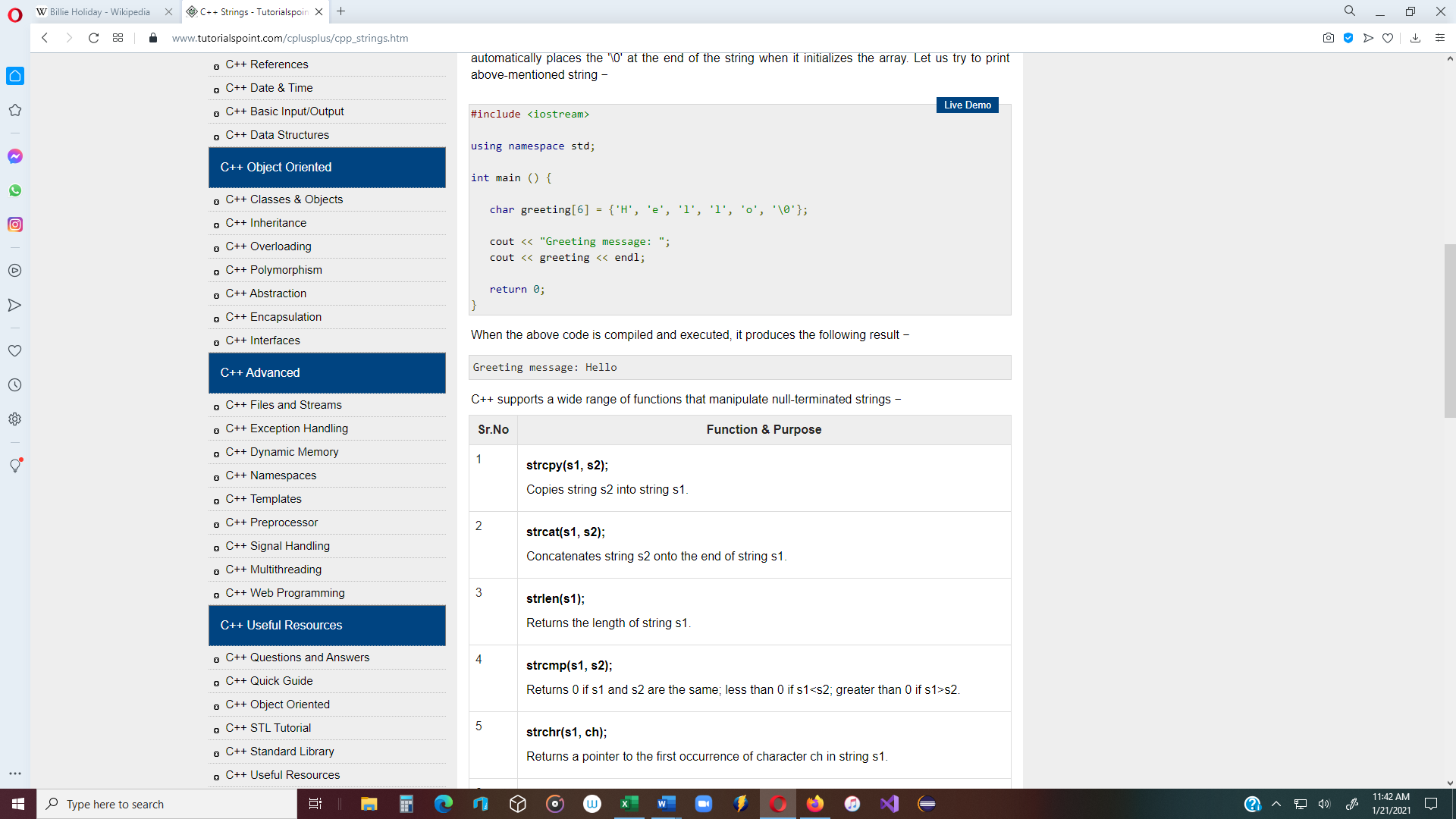
**Character Array** (Also called a c\_string)

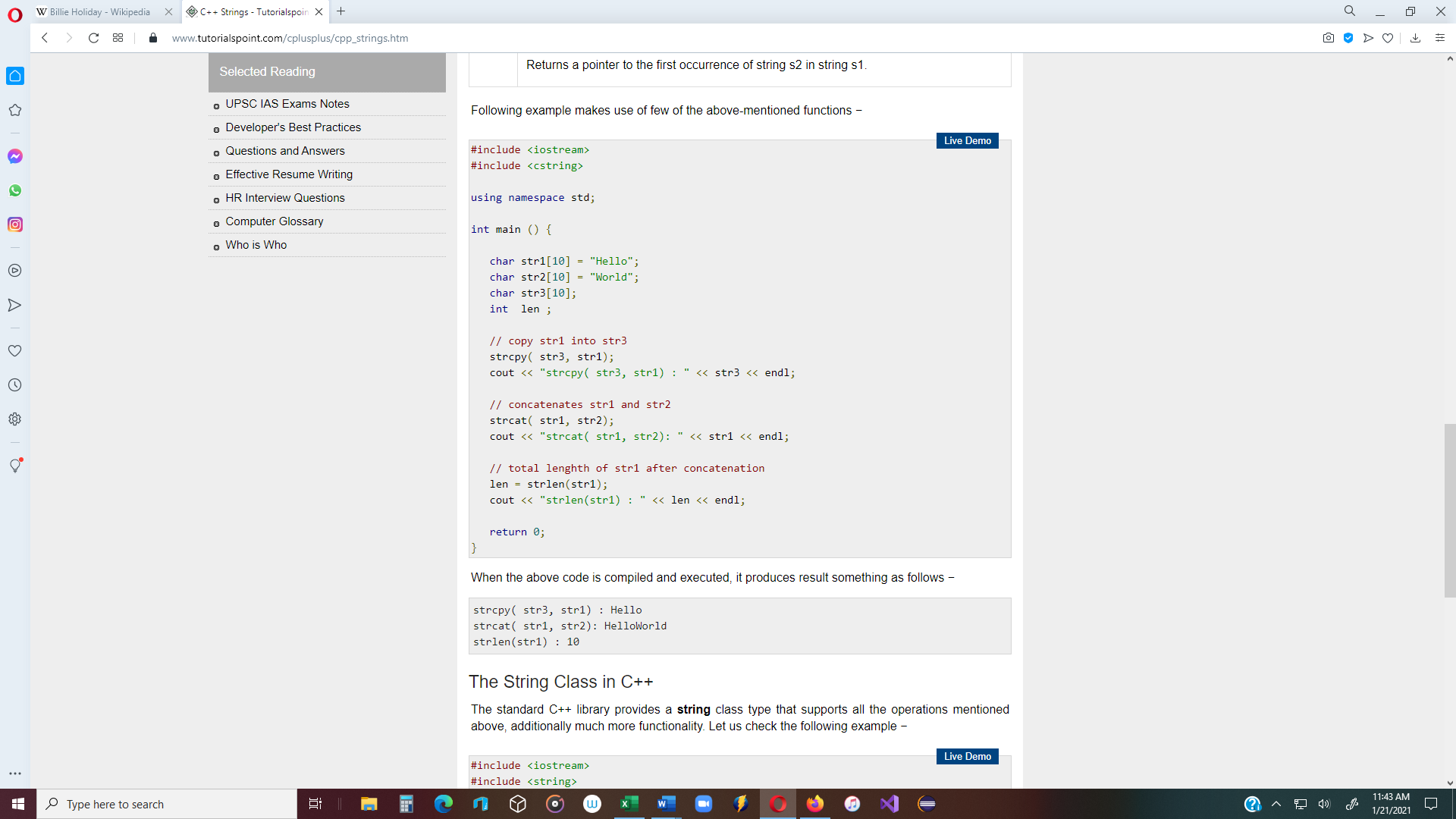
A sequence of characters stored in an array.

A c\_string is a **null-terminated character array**.

The computer places a **null character** at the end of the string of characters.







Ex: **Read character data into a c\_string.**

#include <iostream>

#include <string>

using namespace std;

void displayName(char name[]);

int main()

{

const int SIZE = 30;

char fullName[SIZE];

cout << "Enter your name: ";

// The **cin.getline()** function reads characters and stops at the **‘\n’** character.

// In this case, 29 characters are read in using the **cin.getline()** function.

// The last element is automatically assigned the NULL character.

cin.getline(fullName);

// Pass an array to a function.

displayName(fullName); // Pass the address of fullName[0] to

// the displayArray function

return 0;

} // end main()

// -------------------------------

void displayName(char name[])

{

cout << "Your name is: " << name << endl;

}

// -------------------------------

**string**  vs. **c\_string**

string fullName; **🡨 string type** requires: **#include<string>**

getline(cin, fullName); **🡨 getline( )** requires: **#include<string>**

- Does not require declaring the size of fullName

char fullName[SIZE];

cin.getline(fullName, SIZE); **🡨 cin.getline( ) Does not require #include<string>**

- The array size must be specified before using.

* **To initialize a c\_string to an empty string:**

char name[30] = {'\0'};

char name[30] = ""; // empty string produces the null character

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**String literal** - A sequence of characters that are type directly into a program are called

string literals.

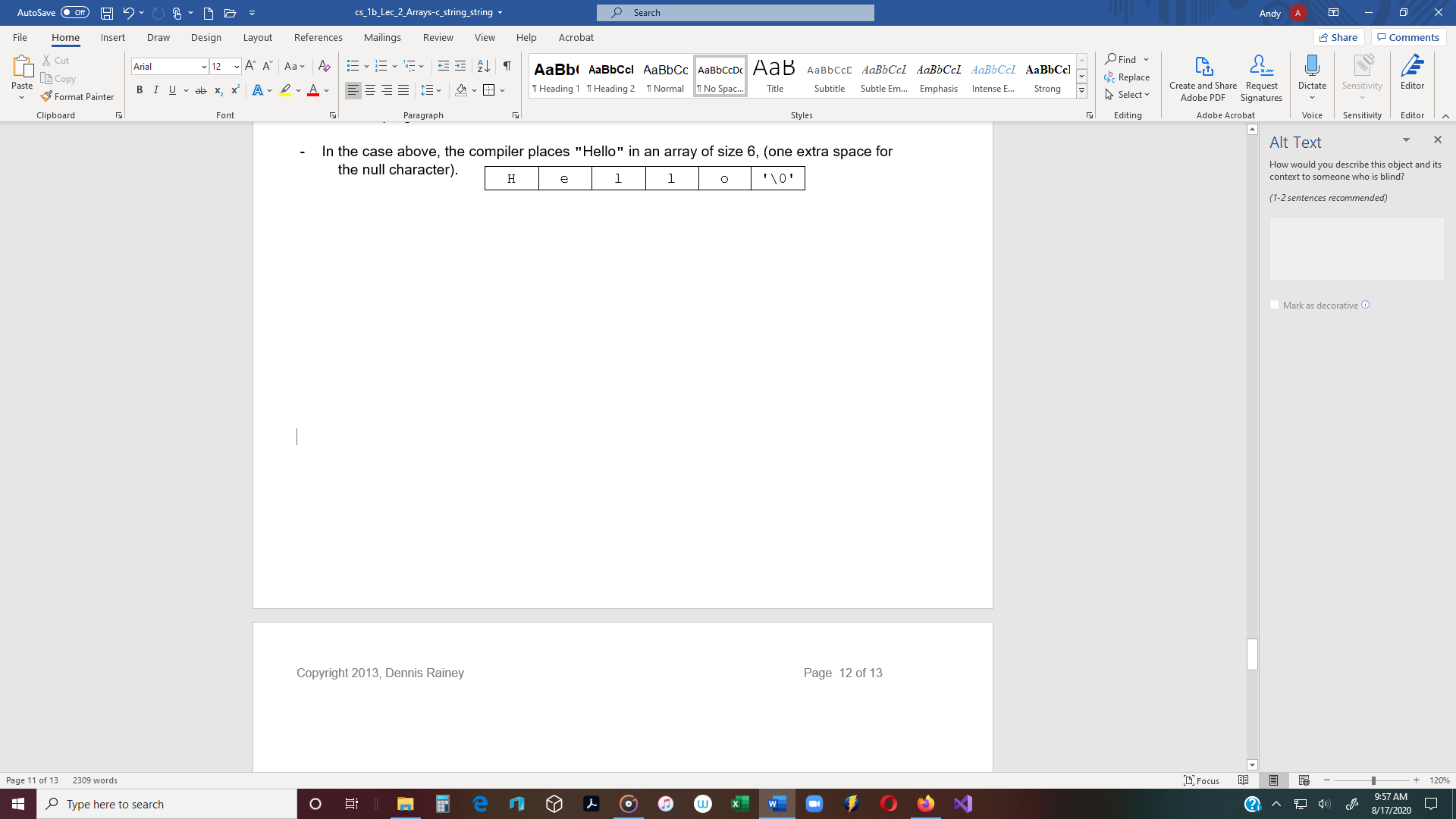
* String literals are null-terminated c\_strings.

Ex #21: cout << "Hello";

* When the compiler encounters a string literal, it places it in an array in the data section

of the program.

* In the case above, the compiler places "Hello" in an array of size 6, (one extra space for

 the null character).

**To put a value into a c\_string:**

1. Initialize a character array: char name[8] = "Tom Lee";

* This is the only time the assignment operator (=) can be used with c\_strings.
* The 8th element ( name[7]) is automatically assigned the NULL character.

2. Use a standard c\_string function – **strcpy** // #include <string>

* + strcpy(fullName, "Tom Lee");

3. Use **cin.getline** function

* + cin.getline(fullName, 30); // Reads up to 29 characters.
  + **cin.getline function** – Reads an entire line of text up to a newline ('\n'), or to

the length of the character array, whichever comes first.

* + **cin.getline function** – Replaces the newline ('\n') character with a NULL

**Four c\_string functions**

**1.** **strcpy(str1, str2)** - Copies a literal string from one to another.

(str1 is the destination and str2 is the source)

Ex #22: char name[30];

name = "Bill"; 🡨 Error! Assignment operator does

not work with c\_strings.

strcpy(name, "Bill"); 🡨 OK

**2.** **strlen(str1)** - Counts the number of array characters (excluding the NULL character).

Ex #23: cout << name << " has " << strlen(name) << " characters."

// Bill has 4 characters.

**3. strcat(str1, str2)**  - Concatenates the strings (combines the strings).

Ex #24: strcat(name, " "); // 🡪 "Bill "

strcat(name, "Jones"); // 🡪 "Bill Jones"

**4. strcmp(str1, str2)** - Compares the strings to see if they are equal. Returns 0 if equal.

(Case sensitive - ‘A’ != ‘a’)

Ex #25: char myName[30] = "Tom";

char yourName[30] = "Tim";

if ( strcmp(myName, yourName) == 0 )

cout << "Same name";

Ex #26: **To read in a c\_string, and then output a c\_string:**

#include <iostream>

#include <string> // Required in order to use string data type and getline()

using namespace std;

int main()

{

char name[30];

cout << "Enter your name: ";

cin.getline(name, 30);

cout << "Your name is " << name << endl;