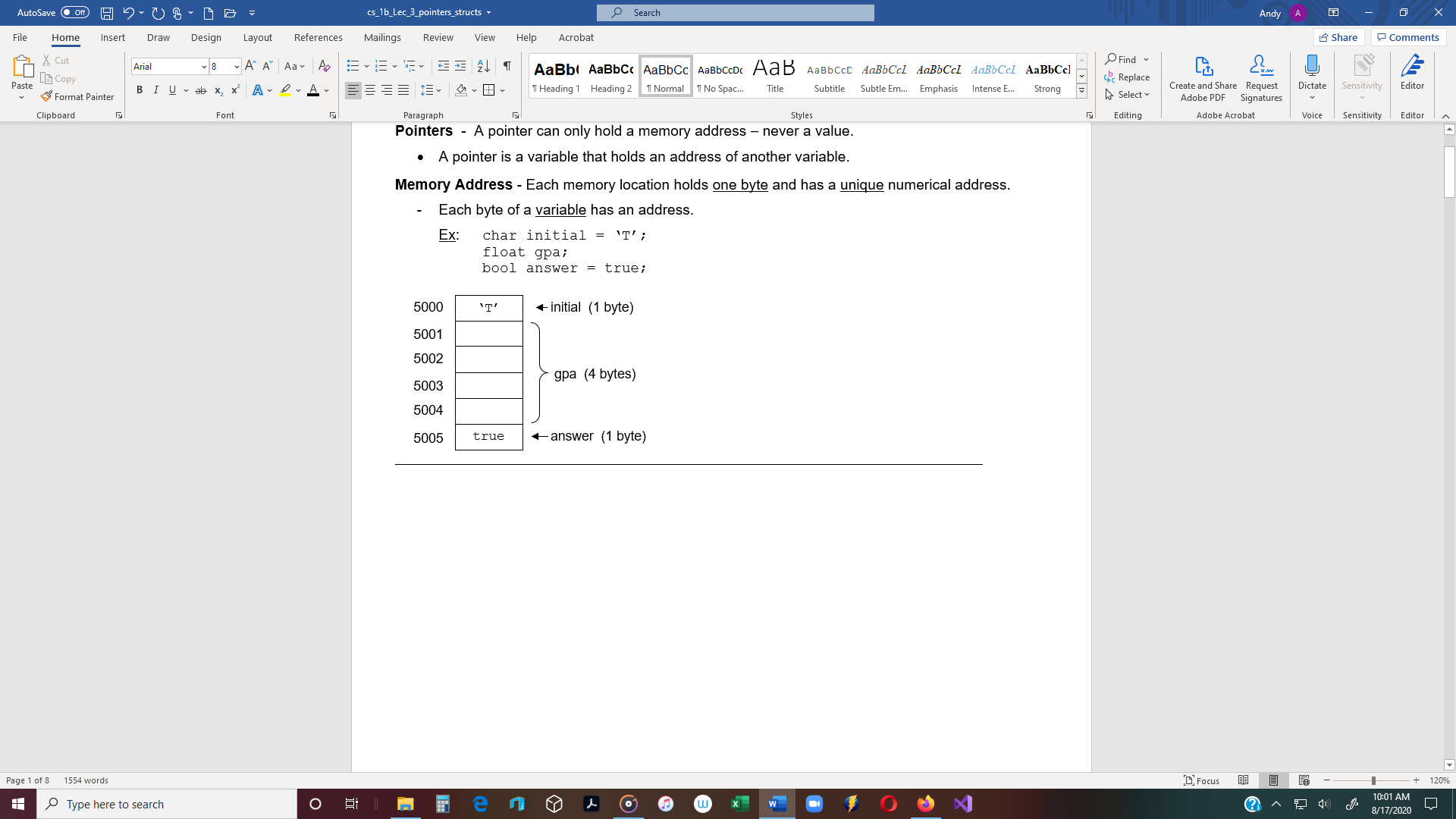
# Lecture 3 – Pointers - Structures - Header files

**Pointers** - A pointer can only hold a memory address – never a value.

* A pointer is a variable that holds an address of another variable.

**Memory Address** - Each memory location holds one byte and has a unique numerical address.

* Each byte of a variable has an address.

Ex: char initial = ‘T’;

float gpa;

bool answer = true;

**To declare a pointer:**

char \*ptr; // Declares a pointer variable.

// The pointer points to variables of char data type.

// An asterisk must be in front of each pointer variable.

float \*floatPtr;

char\* charPtr; All of these declarations are OK.

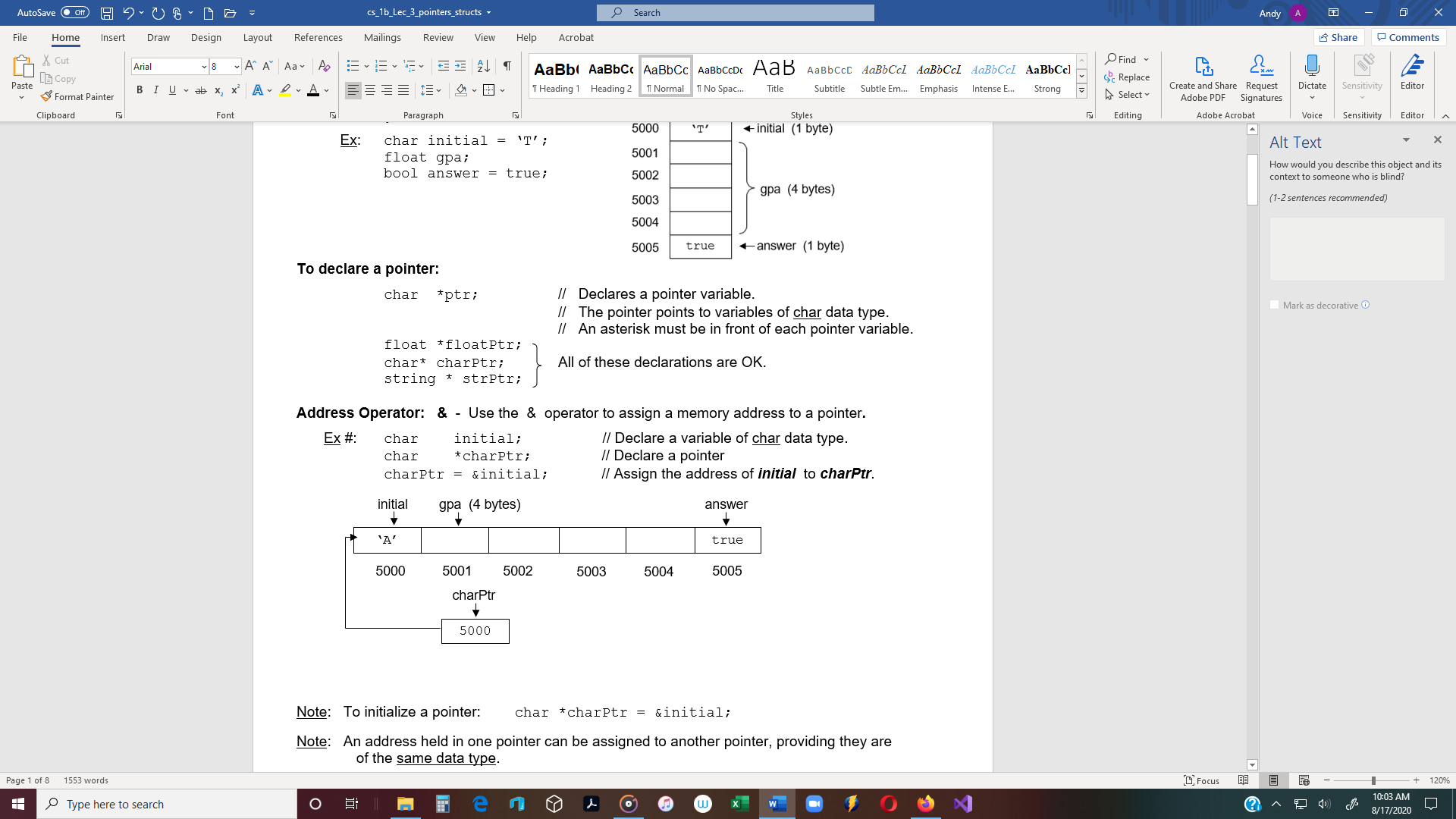
string \* strPtr;

**Address Operator:** **&** - Use the & operator to assign a memory address to a pointer**.**

Ex #: char initial; // Declare a variable of char data type.

char \*charPtr; // Declare a pointer

charPtr = &initial; // Assign the address of ***initial***  to ***charPtr***.



Note: To initialize a pointer: char \*charPtr = &initial;

Note: An address held in one pointer can be assigned to another pointer, providing they are

of the same data type.

**De-reference Operator** ( **\* )** - (also called **indirection operator**)

* When a pointer points to a variable, the value held in the variable can be accessed

(de-referenced), by using the de-reference operator ( \* ).

Ex #: double deposit; // Declare a variable of double data type.

double \* ptr; // Declare a pointer that points to variables of double type

ptr = & deposit; // Assign the address of ***number*** to the pointer.

\* ptr = 1050.0; // Assigns 1050.0 to number.

Ex #: cout << deposit; // Output: 1050.0

cout << \*ptr; // Outputs the value in the variable pointed to // by ***ptr*** (which in this case is 1050.0)

Ex #: int num1; // Declare an int variable.

int \* ptr = & num1; // Declare a pointer and assign the address

// of ***number*** to ***ptr***.

int \* temp; // Declare a pointer that points to int variables.

temp = ptr; // This works because they are both int pointers.

float \* amtPtr; // Declare a pointer that points to float variables.

amtPtr = ptr; // Error – different data types.

**Pointers, Arrays, and Pointer Arithmetic**

* Assign a pointer to the first element in an array.
* Access array elements by incrementing the pointer.

Ex #: int main()

{ No **&** when assigning a array address.

const int SIZE = 5;

int numbers [SIZE] = {6, 8, 9, 3, 7};

int \*ptr;

ptr = numbers; 🡨 Assigns the address of numbers[0] to ptr.

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

{

cout << \*ptr << endl;

ptr++; 🡨 Increments the address of *numbers*[0]

} to *numbers* [1].

return 0; (Therefore, the address is incremented

} by 4, because integers are 4 bytes.)

**Pointers as function parameters**

* A pointer can be used as a function parameter.
* When a pointer is passed to a function, the pointer holds an address of a variable that can

then be accessed by the function.

* A pointer gives a function access to the original variable, much like a reference variable does.

Note: When a variable is passed by reference, the reference variable acts as an alias to

the original variable. This gives the function access to the original variable.

* Generally, passing reference variables is easier than passing pointers as arguments.
  + However, pointers to c\_strings work well and are easy.

#include <iostream>

using namespace std;

void getName(char \*);

void displayName(char \*);

const int SIZE = 30;

int main()

{

char name[SIZE];

char \* ptr = name; 🡨 The address of the first byte of the array is

assigned to the pointer.

getName(ptr);

displayName(ptr);

return 0;

}

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

/\* **OUTPUT:**

Enter name: Tom Lee

Hi Tom Lee.

Press any key to continue \*/

void getName(char \* pName)

{

cout << "Enter name: ";

cin.getline(pName, SIZE);

}

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

void displayName(char \* pName)

{

cout << "Hi " << pName << ".\n";

}

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

**struct** (structures) - A structured data type.

- A collection of components referred to by a single name.

- class and struct are two structured data types (classes later).

**struct** – Reserved word for structure.

* **Object** - A struct variable is called an object.
* **struct** – A data type in which each object is a collection of components, called data members.

**Data members** of a struct object are **public**. (Class data members are **private** by default).

* **public** – Means that any function has access to the object’s data members.
* **private** – Means only member functions can have access to data members.
  + Private data members are more secure.

**General convention:** Use structs when there are no member functions.

Use classes when there are member functions.

**To define a structure:**

Ex #: struct Time // Identifier starts with uppercase letter

{

int hours;

int minutes;

int seconds;

}; // Semicolon is required

Ex #: **Using a struct in a program**

#include <iostream>

using namespace std;

struct Date // The data type is **Date**.

{ // The struct can be defined before main, but usually in a header file.

int month; // **Memory is not allocated when the struct is defined**.

int day;

int year; // month, day and year are **data members** of struct Date.

};

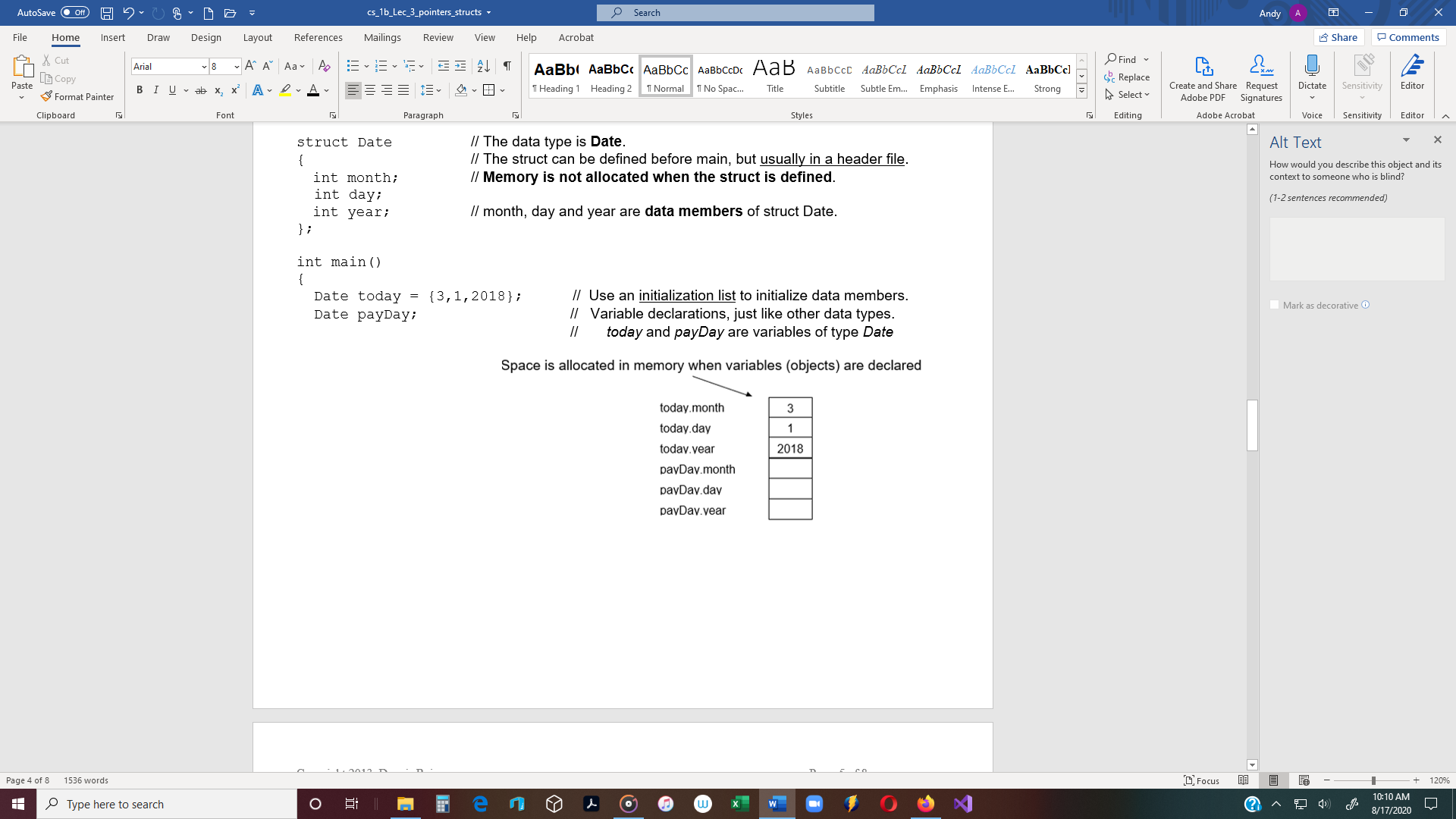
int main()

{

Date today = {3,1,2018}; // Use an initialization list to initialize data members.

Date payDay; // Variable declarations, just like other data types.

// *today* and *payDay* are variables of type *Date*



**Dot notation** – Another way to assign values to a struct object is to use the Dot Operator ( **.** )

payDay.month = 3;

payDay.day = 1;

payDay.year = 2010;

cout << “Enter today’s date (mmddyy): ”

cin >> today.month >> today.day // cin >> Date; 🡨**Wrong**

>> today.year;

if (((today.month == payDay.month) &&

(today.day == payDay.day)) &&

(today.year == payDay.year)))

{

cout << “Today is pay day!”;

}

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

**Pass a struct object to a function by reference.**

* The memory address of the first byte of the struct object is passed.

Ex #: void getDate(Date & date);

int main()

{

Date today;

getDate(today);

return 0;

}

-------------------------

void getDate(Date & date)

{

cout << “Enter the date (mmddyy): ”

cin >> date.month >> date.day >> date.year;

}

2 ways to access the individual data members: (see struct Box below)

1. **Dot operator**

Ex #: Box tackleBox**;** // Declare a box object (see struct Box)

tackleBox.width = 10**;**

1. **Arrow Operator** - Use a **pointer** with the arrow operator to access an

object’s data members

* + The Arrow Operator consists of a hyphen ( - ) and a greater-than symbol ( > ).

Ex #: Box shoeBox**;** // Declare a box object (see struct Box)

Box \*ptr;// Declare a pointer that points to Box objects

ptr = & shoeBox // Assign the address of shoeBox to ptr.

ptr -> width = 10;// arrow operator to access shoeBox**.**width

// Use the following struct with the program on the next page.

struct Box

{

int width;

int height;

int length;

};

int main()

// Declare a pointer that can hold

// the address of a Box object.

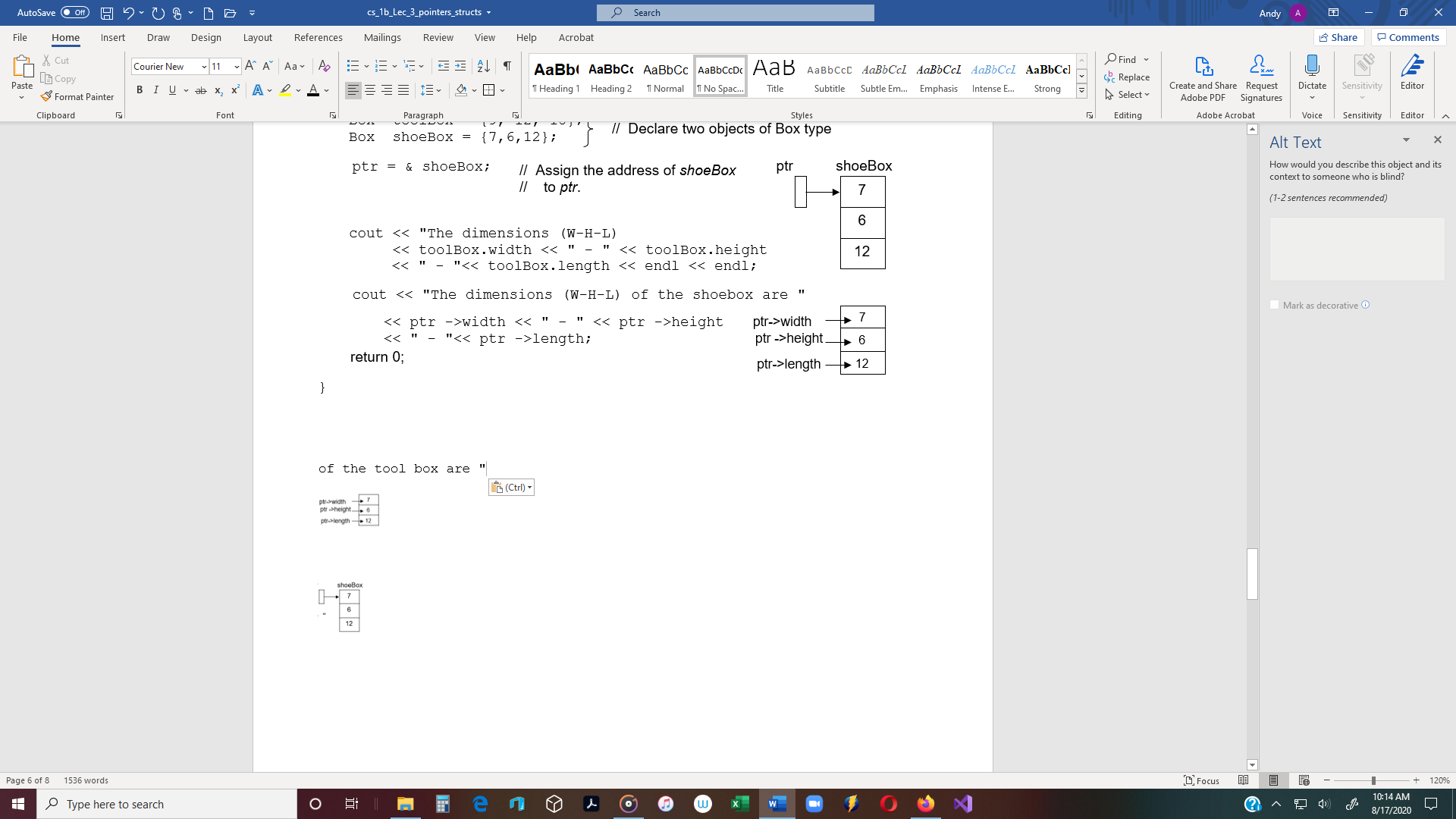
{

Box \* ptr;

Box toolBox = {9, 12, 18};

// Declare two objects of Box type

Box shoeBox = {7,6,12};



ptr = & shoeBox;

// Assign the address of *shoeBox*

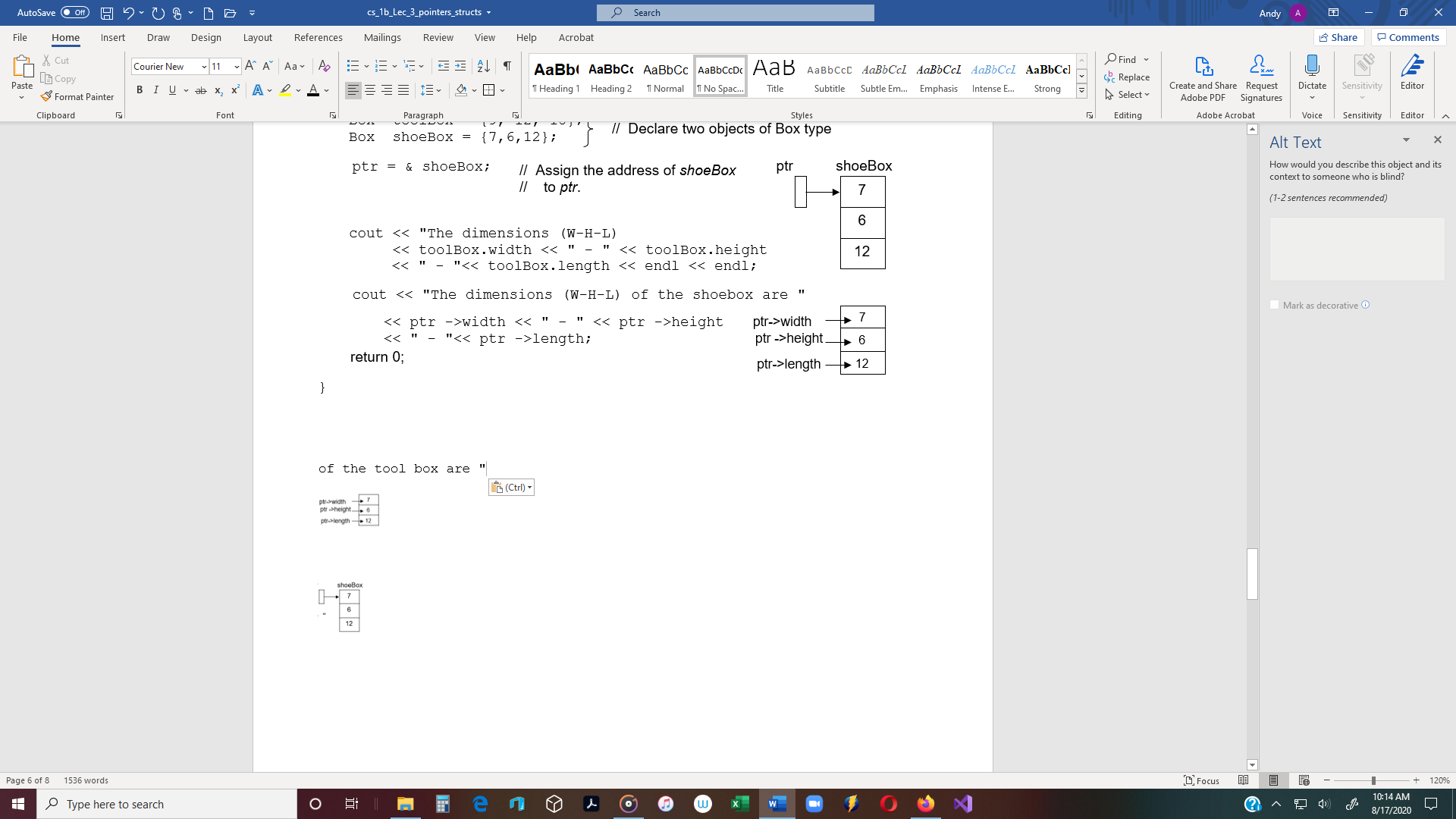
// to *ptr*.

cout << "The dimensions (W-H-L) of the tool box are "

<< toolBox.width << " - " << toolBox.height

<< " - "<< toolBox.length << endl << endl;

cout << "The dimensions (W-H-L) of the shoebox are "

 << ptr ->width << " - " << ptr ->height

<< " - "<< ptr ->length;

return 0;

}

/\* OUTPUT

The dimensions (W-H-L) of the tool box are 9 - 12 - 18

The dimensions (W-H-L) of the shoebox are 7 - 6 - 12

Press any key to continue \*/

**Constructor function** - A struct specification can include a constructor.

* A constructor is a function with the same name as the struct itself.
* A constructor allows a struct object’s data members to be initialized at the time

the object is declared.

Ex #: Because of the constructor below, when a Box object is declared, its data

members are assigned zeros.

struct Box

{

int width;

int height;

int length;

// Constructor

Box()

{

width = 0;

height = 0;

length = 0;

}

};

**Array of struct objects** - A list of struct objects (records) can be held in an array.

- Each array element holds one struct object.

struct Record

{

string name;

int age;

};

const int SIZE = 5;

int main()

{

Record records[SIZE];

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

{

cout << "AGE: ";

cin >> records[i].age;

cin.ignore();

cout << "NAME: ";

getline(cin, records[i].name); // string type

**Header Files** – Programmer-defined header files can be used to hold C++ declarations of

constants, function prototypes / implementations and class specifications, etc.

**#include <iostream>** - **#include** is a directive to the preprocessor to insert the contents of

the iostream header file.

Ex:Create a header file and name it: **Box.h**

* + A new header file can be created the same way as a new .cpp file, except

select **C++ Header File** instead of C++ Source File.

Place the struct Box specification in the file.

Then include the following preprocessor directive: Enclose in quotes

#include "Box.h"

int main()