Hands-on Activity 5.2	
Structures	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic & Design	**Date Performed: 10/2/2025**
Section: CPE11S1	**Date Submitted: 10/5/2025**
Name(s): Ralph Angelov F. Braganza	**Instructor: Engr. Jimlord M. Quejado**

Output

CODE 1: Sample Code of Using the structure member and structure pointer operators.

```
#include <iostream>
#include <string>
using namespace std;

struct Card {
   string face;
   string suit;
};

int main() {
   Card a;      // structure variable
   Card* aPtr;   // structure pointer

   // Assign values
   a.face = "Ace";
   a.suit = "Spades";

   // Pointer points to structure 'a'
   aPtr = &a;

   // Accessing members:
   //  Using the dot operator (.)
   cout << a.face << " of " << a.suit << endl;

   //  Using the arrow operator (->)
   cout << aPtr->face << " of " << aPtr->suit << endl;

   //  Using dereference (*) and dot
   cout << (*aPtr).face << " of " << (*aPtr).suit << endl;
   return 0;
}
```

RESULT:

```
Ace of Spades
Ace of Spades
Ace of Spades

--------------------------------
Process exited after 0.01706 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS:
We have our headers <iostream> for simple input/outputs (cin and cout) and we have the <string> to access the string library, this will allow us to create and manipulate the sequence of characters like “Ace” and “Spades”. Then we have using namespace std; so we only have to type out “cout” and “string” instead of std::cout etc. In this line of code is where the struct will be created to store our face member and our suit member which both are declared as strings and the name for the struct is called Card. Now onto our main function, here we have a structure variable named “a” of the type of Card and we have a structure pointer named aPtr (indicating it’s a pointer) that will store the memory address of a Card address but at this point, aPtr doesn’t really point anywhere it’s just declared. The next few lines are where we will assign the values to the members of the structure “a”, a.face = "Ace" and a.suit = "Spades". Next, the & operator will get the memory address of “a” then that will be stored in the pointer aPtr so now aPtr points to “a”.

cout << a.face << " of " << a.suit << endl;

This prints the values of the structure’s members using the “.” operator (this will connect the structure variable “a” to one of its members) since “a” is a direct structure variable and not a pointer.

The output will be:
Ace of Spades

cout << aPtr->face << " of " << aPtr->suit << endl;

The next one will print essentially the same thing but this time it uses a pointer. The arrow operator (->) is a shortcut/shorthand for dereferencing a pointer and then accessing its member

aPtr-> is basically equivalent to (*aPtr).face which will also output “Ace of Spades”

Then the next lines of code is just the same thing except we are using the long form of the arrow operator which is (*aPtr).face. It’s exactly the same as the previous line, just written differently.

cout << (*aPtr).face << " of " << (*aPtr).suit << endl;

CODE 2: Sample code of Accessing Structure Members.

```
#include <iostream>
#include <string>
using namespace std;

// Define the structure
struct Books {
   string title;
   string author;
   string subject;
   int book_id;
};

int main() {
   // Declare two Book variables
   Books Book1;
   Books Book2;

   // Book 1 specification
   Book1.title = "C Programming";
```

```
    Book1.author = "Nuha Ali";
    Book1.subject = "C Programming Tutorial";
    Book1.book_id = 6495407;

    // Book 2 specification
    Book2.title = "Telecom Billing";
    Book2.author = "Zara Ali";
    Book2.subject = "Telecom Billing Tutorial";
    Book2.book_id = 6495700;

    // Print Book 1 info
    cout << "Book 1 title   : " << Book1.title << endl;
    cout << "Book 1 author  : " << Book1.author << endl;
    cout << "Book 1 subject : " << Book1.subject << endl;
    cout << "Book 1 book_id : " << Book1.book_id << endl;

    cout << endl; // for spacing

    // Print Book 2 info
    cout << "Book 2 title   : " << Book2.title << endl;
    cout << "Book 2 author  : " << Book2.author << endl;
    cout << "Book 2 subject : " << Book2.subject << endl;
    cout << "Book 2 book_id : " << Book2.book_id << endl;

    return 0;
}
```

RESULT:

```
Book 1 title   : C Programming
Book 1 author  : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407

Book 2 title   : Telecom Billing
Book 2 author  : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

ANALYSIS:

<iostream> will allow us to use input/output features like cout for displaying the text then cin to input text, <string> allows us to use string data type for storing words or sentences and using namespace std; will directly let us type cout, string, and endl without using std:: infront of those functions.

```
// Define the structure
struct Books {
    string title;
    string author;
    string subject;
    int book_id;
};
```

This will create a structure (struct) and the name for the struct is called Books. Inside the structure we have title, author, subject, and book_id all declared a string except for book_id which is declared as an int data type.

```
// Declare two Book variables
Books Book1;
Books Book2;
```

Here we are declaring the two book variables where each one has its own copy of the title, author, subject, and book_id.

Each line assigns a value to one of Book1's members and the dot operator (.) is used to access a structure's members so it knows which book its assigned to. Then the next line of code is the specifications for Book2 which essentially does the same thing except the things about the book are different from book1.

```
// Book 1 specification
Book1.title = "C Programming";
Book1.author = "Nuha Ali";
Book1.subject = "C Programming Tutorial";
Book1.book_id = 6495407;

// Book 2 specification
Book2.title = "Telecom Billing";
Book2.author = "Zara Ali";
Book2.subject = "Telecom Billing Tutorial";
Book2.book_id = 6495700;
```

```
    // Print Book 1 info
    cout << "Book 1 title   : " << Book1.title << endl;
    cout << "Book 1 author  : " << Book1.author << endl;
    cout << "Book 1 subject : " << Book1.subject << endl;
    cout << "Book 1 book_id : " << Book1.book_id << endl;

    cout << endl; // for spacing

    // Print Book 2 info
    cout << "Book 2 title   : " << Book2.title << endl;
    cout << "Book 2 author  : " << Book2.author << endl;
    cout << "Book 2 subject : " << Book2.subject << endl;
    cout << "Book 2 book_id : " << Book2.book_id << endl;

    return 0;
}
```

Finally, the next block of code will display all the stored information for Book 1, using labels (title, author, subject, book_id) followed by the actual data values it was assigned earlier in the program and after the spacing it will also do this for Book2 then the program will end.

CODE 3: Sample Code of Structure Function Arguments.

```
#include <iostream>
#include <string>
using namespace std;
// Define a structure
struct Books {
    string title;
    string author;
    string subject;
    int book_id;
};
// Function declaration (structure passed by value)
void printBook(Books book);
int main() {
    // Declare two Book variables
    Books Book1;
    Books Book2;
    // Book 1 specification
    Book1.title = "C Programming";
    Book1.author = "Nuha Ali";
    Book1.subject = "C Programming Tutorial";
    Book1.book_id = 6495407;
    // Book 2 specification
    Book2.title = "Telecom Billing";
```

```
    Book2.author = "Zara Ali";
    Book2.subject = "Telecom Billing Tutorial";
    Book2.book_id = 6495700;
    // Print details by passing the structure to a function
    printBook(Book1);
    cout << endl; // just for spacing
    printBook(Book2);
    return 0;
}
// Function definition
void printBook(Books book) {
    cout << "Book title   : " << book.title << endl;
    cout << "Book author  : " << book.author << endl;
    cout << "Book subject : " << book.subject << endl;
    cout << "Book book_id : " << book.book_id << endl;
}
```

RESULT:

```
Book title   : C Programming
Book author  : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407

Book title   : Telecom Billing
Book author  : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

ANALYSIS:

Here we essentially have the same block of code from the last one with our headers <iostream>, <string> and using namespace std. And we have the same structure called Books with the members title, author, subject and book_id all declared as strings except for book_id.

The void printBook(Books book); is a passed by value, meaning the function gets a copy of the book, not the original. After the line int main we the two Books variables which are Books1 and Books2 and each one will store information about a different book.

```
    // Book 1 specification
    Book1.title = "C Programming";
    Book1.author = "Nuha Ali";
    Book1.subject = "C Programming Tutorial";
    Book1.book_id = 6495407;

    // Book 2 specification
    Book2.title = "Telecom Billing";
    Book2.author = "Zara Ali";
    Book2.subject = "Telecom Billing Tutorial";
    Book2.book_id = 6495700;
```

We then have all the given specifications of the Book1 and Book2.

printBook(Book1) is calling the function printBook() and passingBook1 to it, this will then passes a copy of Book1 to it and inside the function is the copy called book then a space between using endl, it does the same thing for Book2 and that will be the completion of the program. So, to explain it even further if you write printBook(Book1) or printBook(Book2) the

data from Book1 and Book2 is copied into the function parameter book, so inside the function, book is a new structure variable that has the same values as Book1 or Book2 but it's a separate copy. The block of code below that is the function, it receives a copy of the Books structured named book and it uses the dot operator to access the members inside that structure. Then it prints each piece of information to the console.

```
    // Book 2 specification
    Book2.title = "Telecom Billing";
    Book2.author = "Zara Ali";
    Book2.subject = "Telecom Billing Tutorial";
    Book2.book_id = 6495700;

    // Print details by passing the structure to a function
    printBook(Book1);
    cout << endl; // just for spacing
    printBook(Book2);

    return 0;
}

// Function definition
void printBook(Books book) {
    cout << "Book title   : " << book.title << endl;
    cout << "Book author  : " << book.author << endl;
    cout << "Book subject : " << book.subject << endl;
    cout << "Book book_id : " << book.book_id << endl;
}
```

Supplementary Activity

1. Create a program that uses a **structure** to store a rectangle's **length** and **width**. Write a **function** that accepts the structure as an argument and **computes the area and perimeter** of the rectangle.

CODE:

```
#include <iostream>

struct Rectangle {
   float length;
   float width;
};

void computeRectangle(Rectangle rect);

int main() {

   Rectangle rect;

   std::cout << "------Rectangle Area & Perimeter Calculator------\n";
   std::cout<<"Please Enter the Length & Width of the Rectangle\n\n";

   while (true) {
      std::cout << "Enter Length: ";
      std::cin >> rect.length;

      if (std::cin.fail()) {
         std::cin.clear();
         std::cin.ignore(10000, '\n');
         std::cout << "Invalid input. Please enter a numeric value!\n\n";
      } else {
         break;
      }
   }
```

```
    while (true) {
        std::cout << "Enter Width: ";
        std::cin >> rect.width;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input. Please enter a numeric value!\n\n";
        } else {
            break;
        }
    }

    computeRectangle(rect);

    return 0;
}

void computeRectangle(Rectangle rect) {
    float area = rect.length * rect.width;
    float perimeter = 2 * (rect.length + rect.width);

    std::cout << "\n--- Rectangle Details ---" << std::endl;
    std::cout << "Length   : " << rect.length << std::endl;
    std::cout << "Width    : " << rect.width << std::endl;
    std::cout << "Area     : " << area << std::endl;
    std::cout << "Perimeter: " << perimeter << std::endl;
    std::cout << "------------------------" << std::endl;
}
```

ANALYSIS:

First, I added <oistream> for the program to be able to use cin for inputs and cout for outputs. Here I created a structure where I defined a new data type named Rectangle that contains length and width declared as float. The next line of code is where I used the void function, this will tell the compiler that this function does not return any value so when the function finishes running it just goes back to where it was called so no result is sent back our outputted. The name of the function is computeRectangle and the (Rectangle rect) is the parameter list where Rectangle is the data type of the parameter (this is the one defined earlier from the struct) and rect is the name of that parameter inside the function.

Now for our main function, this creates a structure variable named rect using the Rectangle data type then the next lines simply print messages on the screen to introduce what the program essentially does and a little guide of what to do to start.

```
Rectangle rect;

std::cout << "------Rectangle Area & Perimeter Calculator------\n";
std::cout<<"Please Enter the Length & Width of the Rectangle\n\n";
```

Next, I've created a while loop that will keep running until a valid input is give and it will ask for the rectangle's length, then stores the input in rect.length but if the user inputs an invalid value it will print out "Invalid input. Please enter a numeric value!\n\n"

```
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input. Please enter a numeric value!\n\n";
        } else {
            break;
        }
    }
```

std::cin.fail() will checks the state of the input returns true if the last input operation failed (meaning if the user entered a text when the program is expecting a number).

std::cin.clear() will clear the error flags on the input and reset it from the failure state and std::cin.ignore(10000, '\n') prevents the same bad input from causing the next std::cin operation to fail immediately (in other words it prevents the program to keep printing/displaying the same error message infinitely over and over again. If the input is finally valid the else part executes calling break; then exits the while-loop. The same process will happen with the width of the rectangle and will also display an error message if the input is not a number.

Now that both length and width are valid, the structure rect is passed by value to the function computeRectangle (this basically means it is sending data to rect into the function. The void computeRectangle(Rectangle rect) receives a Rectangle structure named rect and this has access to rect.length and rect.width. Here is where we will compute the area and perimeter for the rectangle once it gets the access of the length and width.

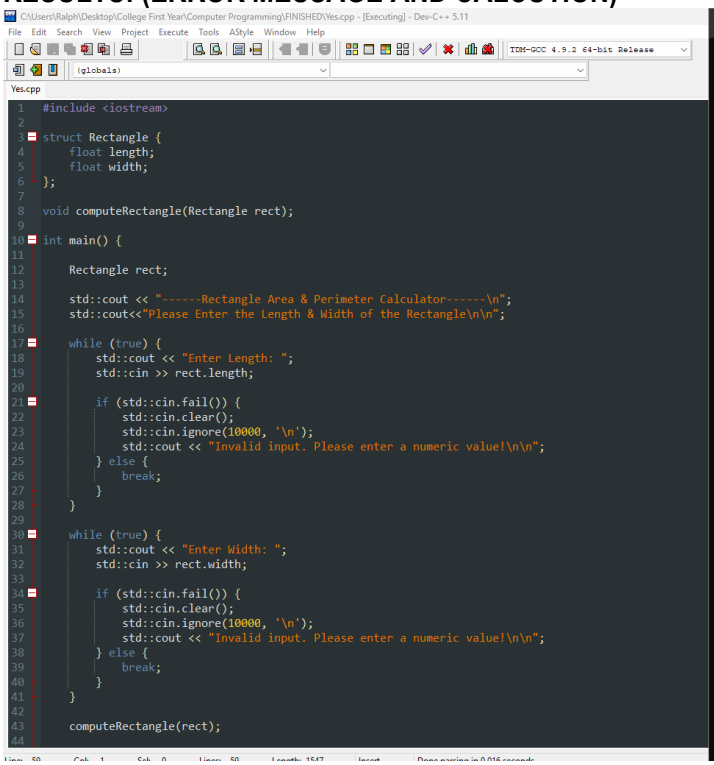
```
void computeRectangle(Rectangle rect) {
    float area = rect.length * rect.width;
    float perimeter = 2 * (rect.length + rect.width);
```

area = length x width
perimeter = 2 x (length + width)
And both are stored as float numbers.

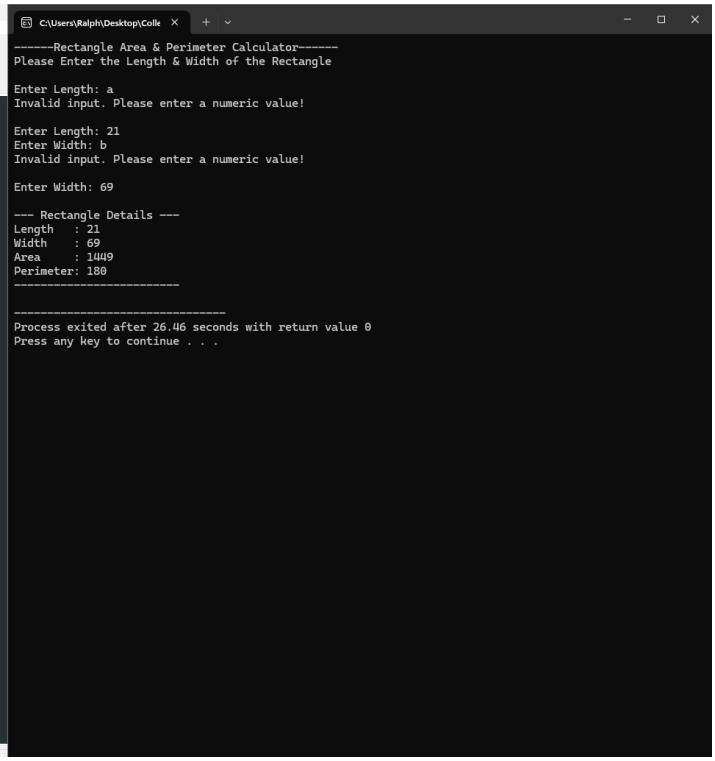
After it does the calculations it will now print the Length, Width, Area and Perimeter

```
    std::cout << "\n--- Rectangle Details ---" << std::endl;
    std::cout << "Length    : " << rect.length << std::endl;
    std::cout << "Width     : " << rect.width << std::endl;
    std::cout << "Area      : " << area << std::endl;
    std::cout << "Perimeter: " << perimeter << std::endl;
    std::cout << "--------------------------" << std::endl;
}
```

RESULTS: (ERROR MESSAGE AND CALCUTION)



```cpp
#include <iostream>

struct Rectangle {
    float length;
    float width;
};

void computeRectangle(Rectangle rect);

int main() {

    Rectangle rect;

    std::cout << "------Rectangle Area & Perimeter Calculator------\n";
    std::cout<<"Please Enter the Length & Width of the Rectangle\n\n";

    while (true) {
        std::cout << "Enter Length: ";
        std::cin >> rect.length;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input. Please enter a numeric value!\n\n";
        } else {
            break;
        }
    }

    while (true) {
        std::cout << "Enter Width: ";
        std::cin >> rect.width;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input. Please enter a numeric value!\n\n";
        } else {
            break;
        }
    }

    computeRectangle(rect);
```



```
------Rectangle Area & Perimeter Calculator------
Please Enter the Length & Width of the Rectangle

Enter Length: a
Invalid input. Please enter a numeric value!

Enter Length: 21
Enter Width: b
Invalid input. Please enter a numeric value!

Enter Width: 69

--- Rectangle Details ---
Length    : 21
Width     : 69
Area      : 1449
Perimeter: 180
--------------------------

--------------------------------
Process exited after 26.46 seconds with return value 0
Press any key to continue . . .
```

2. Write a program that creates a function **multiple** that determines if the integer entered from a keyboard is a multiple of some integer **x.**

CODE:

```cpp
#include <iostream>

bool multiple(int a, int b);

int main() {
   int intValue1;
          int intValue2;

   std::cout << "-------------------------Multiple Checker-------------------------\n";
   std::cout << "Input Two Integers to Determine Whether One is a Multiple of the Other\n\n";

   while (true) {
      std::cout << "Enter the First integer: ";
      std::cin >> intValue1;

      if (std::cin.fail()) {
         std::cin.clear();
         std::cin.ignore(10000, '\n');
         std::cout << "Invalid input! Please enter a numeric value.\n\n";
      } else {
         break;
      }
   }

   while (true) {
```

```
        std::cout << "Enter the Second integer: ";
        std::cin >> intValue2;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input! Please enter a numeric value.\n\n";
        } else if (intValue2 == 0) {
            std::cout << "Cannot check multiples of zero! Please enter a nonzero integer.\n\n";
        } else {
            break;
        }
    }

    if (multiple(intValue1, intValue2))
        std::cout << "\n " << intValue1 << " is a multiple of " << intValue2 << ".\n";
    else
        std::cout << "\n " << intValue1 << " is NOT a multiple of " << intValue2 << ".\n";

    return 0;
}

bool multiple(int a, int b) {
    return (a % b == 0);
}
```

ANALYSIS:

We first have our header <iostream> which gives us access to input/outputs operations (std::cin,std::cout) then we have our function. Instead of void we are using bool where this declares a function that returns a Boolean value (true or false) and it takes two integers a and b as inputs. Now onto our main function, we declare intValue1 and intValue2 as an integer and these variables will store the two numbers entered by the user. The next line of code displays the title and some guidelines of what the program does.

while (true) creates an infinite loop where it keeps asking the user until a valid input is given. It will print a message about asking the user to enter the first integer.

```
while (true) {
    std::cout << "Enter the First integer: ";
    std::cin >> intValue1;
```

std::cin.fail() will check the state of the input returns true if the last input operation failed (meaning if the user entered a text when the program is expecting a number).std::cin.clear() will clear the error flags on the input and reset it from the failure state and std::cin.ignore(10000, '\n') prevents the same bad input from causing the next std::cin operation to fail immediately (basically constantly spamming the error message instead of asking again for another input). If the input is finally valid the else part executes calling break; then exits the while-loop.

```
    if (std::cin.fail()) {
        std::cin.clear();
        std::cin.ignore(10000, '\n');
        std::cout << "Invalid input! Please enter a numeric value.\n\n";
    } else {
        break;
    }
}
```

It is the same process for the second integer but with an extra rule, this prevents division by zero so if the user types 0, it prints a warning and will ask for another input that is not a zero.

```
    } else if (intValue2 == 0) {
        std::cout << "Cannot check multiples of zero! Please enter a nonzero integer.\n\n";
    } else {
        break;
    }
}
```

Here it will call the function multiple(intValue1, intValue2) this returns either true or false. If true, it prints that the first number is a multiple of the second if false, it prints the opposite.

```
    if (multiple(intValue1, intValue2))
        std::cout << "\n " << intValue1 << " is a multiple of " << intValue2 << ".\n";
    else
        std::cout << "\n " << intValue1 << " is NOT a multiple of " << intValue2 << ".\n";

    return 0;
}
```

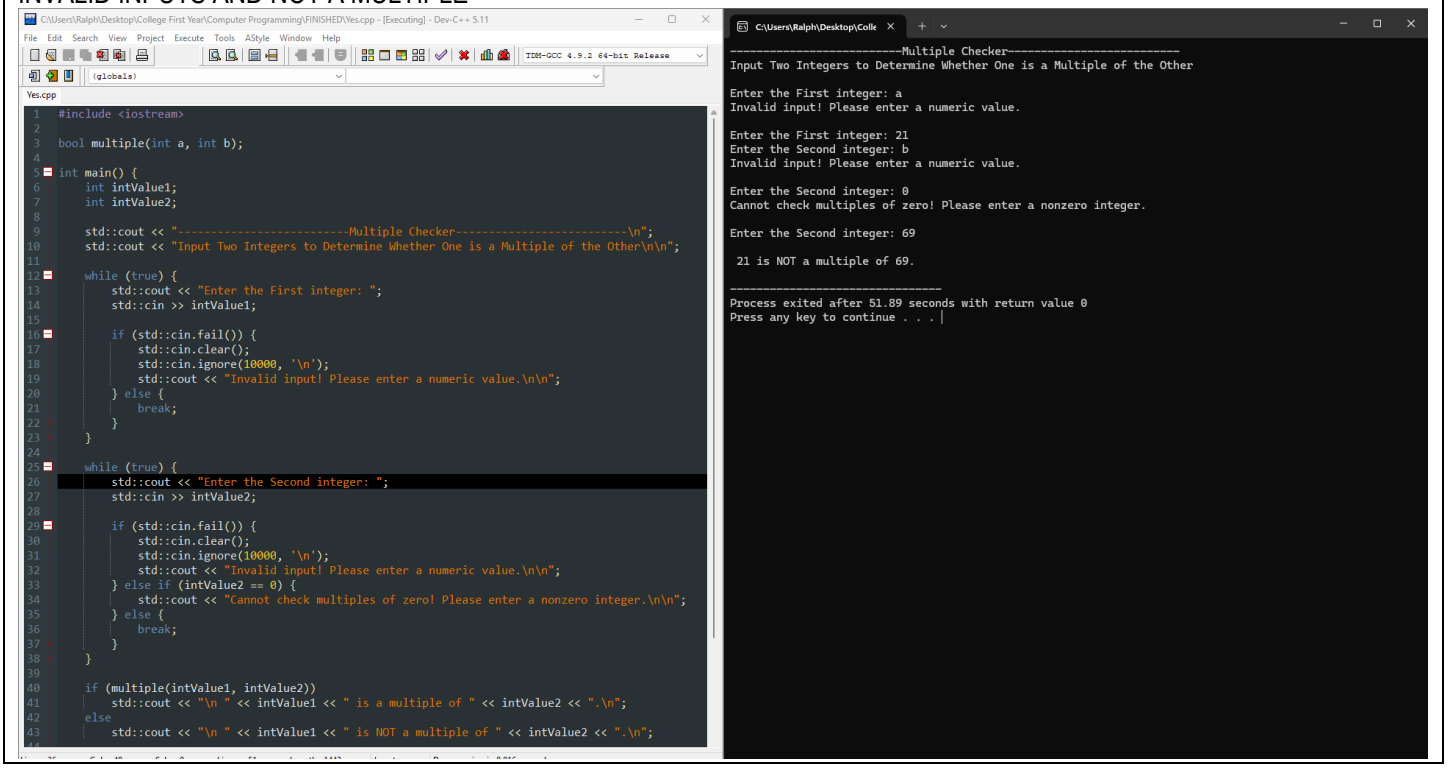
% is the modulus operator, this gives the remainder of a division to check if the values given are multiples of each other.

```
bool multiple(int a, int b) {
    return (a % b == 0);
}
```

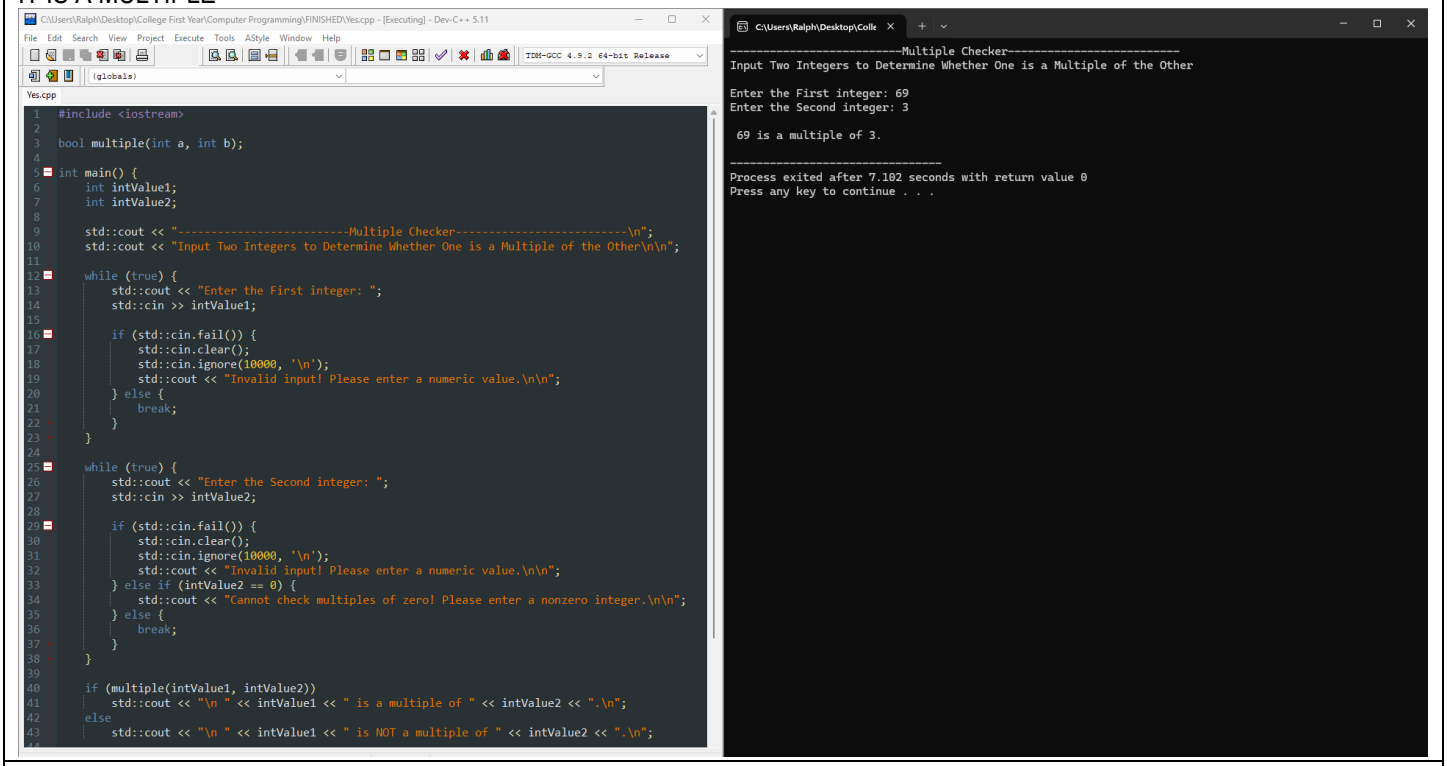
Example:
69 % 3 = 0 is a multiple
69 % 7 = 1 is NOT multiple

The reason why it's outside because it is separate function and not part of the main(). It needs it outside the main() so that it can be called from inside main() whenever needed.

RESULT:
INVALID INPUTS AND NOT A MULTIPLE



IT IS A MULTIPLE



```
#include <iostream>

bool multiple(int a, int b);

int main() {
    int intValue1;
    int intValue2;

    std::cout << "-------------------------Multiple Checker-------------------------\n";
    std::cout << "Input Two Integers to Determine Whether One is a Multiple of the Other\n\n";

    while (true) {
        std::cout << "Enter the First integer: ";
        std::cin >> intValue1;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input! Please enter a numeric value.\n\n";
        } else {
            break;
        }
    }

    while (true) {
        std::cout << "Enter the Second integer: ";
        std::cin >> intValue2;

        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(10000, '\n');
            std::cout << "Invalid input! Please enter a numeric value.\n\n";
        } else if (intValue2 == 0) {
            std::cout << "Cannot check multiples of zero! Please enter a nonzero integer.\n\n";
        } else {
            break;
        }
    }

    if (multiple(intValue1, intValue2))
        std::cout << "\n " << intValue1 << " is a multiple of " << intValue2 << ".\n";
    else
        std::cout << "\n " << intValue1 << " is NOT a multiple of " << intValue2 << ".\n";
```

```
-------------------------Multiple Checker-------------------------
Input Two Integers to Determine Whether One is a Multiple of the Other

Enter the First integer: 69
Enter the Second integer: 3

 69 is a multiple of 3.

--------------------------------
Process exited after 7.102 seconds with return value 0
Press any key to continue . . .
```

Conclusion

This activity gave me more of an understanding about how different parameters outside the main() function work and about analyzing how structs (structures) are used in different ways and how I can use them in different ways as well. It made the supplementary activity easier to do even though this has taken me about 6 hours, but nonetheless, it still gave me a roadmap of where I needed to get started for me to be able to create the code for the two questions that were given to me. I could've just made the code simple and just copied what was on the examples, but the fact that there was no way I was going to pass this on time made me take initiative to go above and beyond by adding some restrictions. I wanted to try and add restrictions to my code so that it will only be able to accept integers or floats. Then I've also added for the second question to check if the value of the second integer is a zero or not so that it can reduce errors while running the program. I've also added design to my code just to make it look cleaner and more readable to understand what is happening with it. I did quite well. I may not have passed the activity on time (I'm sorry, sir), but it has let me immerse myself to learn the code and how exactly it works, reading through multiple articles and listening to tutorials. I will be honest: efficiency is one of the key things I need to work on, but right now I don't think I should really focus on that because what is the point of being fast/efficient if I don't learn anything and my work doesn't even make sense? Eventually, I'll learn to be more efficient, but right now I'm understanding and applying these lessons as much as I can so I'll be able to do more complex tasks in the future, and I pray that I can also improve my efficiency as well along the way.