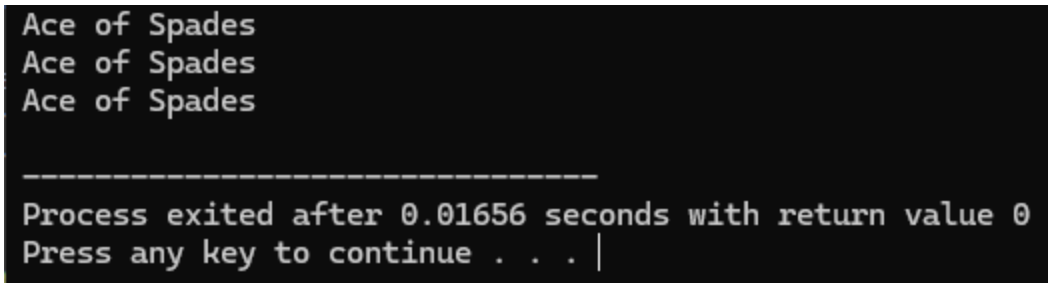
Hands-on Activity 5.2	
Structures	
Course Code: CPE 007	**Program:** Computer Engineering
Course Title: Programming Logic and Design	**Date Performed: 9/30/2025**
Section: CPE11S1	**Date Submitted: 9/30/2025**
Name(s): Juan Paulo C. Lara	**Instructor: Engr. Jimlord M. Quejado**

6. Output

Sample Code of Using the structure member and structure pointer operators:

```cpp
#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;
}
```

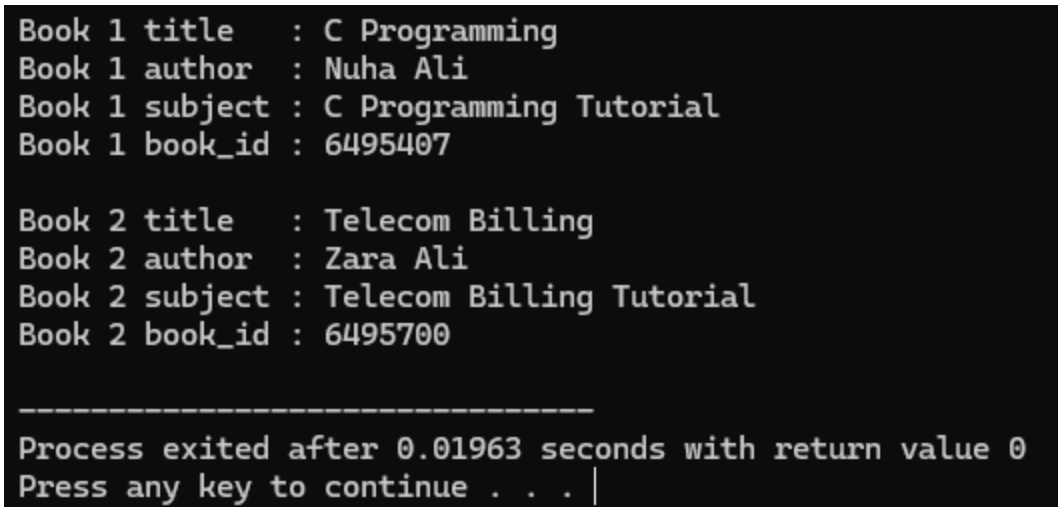


ANALYSIS

During execution of this code, line 5 declares a function called Card as a structure for the two variables face and suit. This forms a structure for the playing cards. Line 10 starts with the int main function. Line 11 creates an instance variable (a) for structure Card using its own face and suit. Line 12 uses Card* aPtr to point to Card by storing the address of variable Card. Line 15 uses a structure variable a.face to assign the values/strings to face and then for a.suit. Line 19 uses a pointer for the card structure by addressing with &a, then storing it to aPtr, resulting to aPtr pointing to variable a. Line 23 prints out the structure variable through a.face and a.suit. An arrow operator at line 25 points aPtr to face directly, and its output is the same as the previous method. Line 29 prints out the dereferenced pointer and dot operator to read structure a and its stored values.

Sample code of Accessing Structure Members:

```cpp
#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;
}
```



ANALYSIS

During execution of this code, line 6 declares a structure for books containing four members consisting of three strings (title, author, subject) and one int value (book_id). This serves as a structure of book information within one variable/name. Line 13 starts an int main function. Line 15 declares variables for the structure by creating two instances of Books with their title, author, subject, book_id. Line 19 to 22 assigns values to Book1 through the dot operator. A similar code is used for Book2. Line 31 to 34 prints out the book info for book 1 using the dot operator. Line 36 adds spacing for the next output block. A similar process is used again for Book2.

Sample Code of Structure Function Arguments:

```cpp
#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;
}
```

```
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

--------------------------------
Process exited after 0.01085 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS

During execution of this code, line 6 declares a data structure called Books containing 3 strings (title, author, subject) and one integer (book_id). This stores the given information about a book. A void function at line 14 declares printBook using the structure from Books as its arguments. By using (Books book), it copies the data and displays them. Since it is a void function, no return value is shown. The main function at line 16 declares Books with Book1 and Book2. Line 22 to 25 assigns the strings to Book1 and its members. A similar process is done at line 28 to 31. Line 34 to 36 calls for copies of printBook for each book. A void function at line 42 calls for printBook and structure (Books) and member (book). Line 43 to 46 prints out the information of each address member by the dot operator.

7. Supplementary Activity

#1

CODE

```cpp
#include <iostream>
#include <string>
using namespace std;
struct rectangle {
    double length;
    double width;
};

void calcRectangle(rectangle rec) {
    double area = rec.length * rec.width;
    double perimeter = 2 * (rec.length + rec.width);

    cout << "Length: " << rec.length << endl;
    cout << "Width: " << rec.width << endl;
    cout << "Area: " << area << endl;
    cout << "Perimeter: " << perimeter << endl;
}

int main() {
    rectangle rec;

    cout << "Enter length of rectangle: ";
    cin >> rec.length;
    cout << "Enter width of rectangle: ";
    cin >> rec.width;
    cout << "\nRectangle info:\n";
    calcRectangle(rec);

    return 0;
}
```

OUTPUT

```
Enter length of rectangle: 11.32
Enter width of rectangle: 24.77

Rectangle info:
Length: 11.32
Width: 24.77
Area: 280.396
Perimeter: 72.18

--------------------------------
Process exited after 4.043 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS

During execution of this code, line 4 declares a structure called rectangle for the double variables length and width. This groups the two separated data into one. A void function at line 9 declares calcRectangle reads the rectangle structure with the rec argument. Line 10 and 11 calculates area and perimeter taking the values using the dot operator. Line 13 to 17 prints out the rectangle information. Since rec is an instance of rectangle, the copied data is contained within this function. Inside the main function at line 19, line 20 initializes the variable rec to the data structure rectangle. Line 22 to 25 waits for user input for length and width of the rectangle. Line 26 outputs rectangle info. Line 27 calls for function calcRectangle(rec) to calculate and output the area and perimeter.

#2

CODE

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   void mult(int num, int x) {
5       if (num % x == 0) {
6           cout << num << " is a multiple of " << x << endl;
7       }
8       else {
9           cout << num << " is not a multiple of " << x << endl;
10      }
11  }
12
13  int main() {
14      int number, x;
15
16      cout << "Input integer: ";
17      cin >> number;
18      cout << "Input multiple of integer (x): ";
19      cin >> x;
20
21      mult(number, x);
22      return 0;
23  }
```

OUTPUT

```
Input integer: 44
Input multiple of integer (x): 8
44 is not a multiple of 8

--------------------------------
Process exited after 8.934 seconds with return value 0
Press any key to continue . . .
```

```
Input integer: 1024
Input multiple of integer (x): 64
1024 is a multiple of 64

--------------------------------
Process exited after 6.713 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS

During execution of this code, a void function at line 4 declares mult reading two integer variables num and x. An if statement at line 5 uses a modulus (%) operator to check whether if num divides by x with no remainder to prove num is a multiple of x. Line 6 outputs the num value with the "is a multiple of" and the x value. An else statement at line 8 outputs similarly to line 6 but this time, it contains the "is not a multiple of" text in the middle. The int main function starting in line 13, int at line 14 declares number for checking and x for the multiple check. Line 16 to 19 waits for user input of the respective values. Line 21 calls for the function mult(number, x) to send the user input values.

8. Conclusion

Concluding this activity, I learned the ways functions can be used to pass data from one function to another, read strings and information from a different location, and call functions to do its command within the function's block. The dot operator can be used for reading values from one function and its arguments to read, compute, and print the information

from the called function. With multiple functions within the same code, the code can do more functions within the same program at runtime, which can allow me to add more solutions in the code to solve problems. The void function also allows for computing values without returning a 0 and ending the program. Overall, this activity allowed me to do more things in one program and one run..