Assignment 4.3:	
Pointers	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic and Design	**Date Performed:** 9/22/2025
Section: CPE11S1	**Date Submitted:** 9/23/2025
Name(s): Ramirez, Angel Mae C.	**Instructor:** Engr. Jimlord M. Quejado

6. Output

1. What is a pointer in C++?

A pointer is a variable that holds another variable's memory address. Instead of containing the data itself, it "points to" the location where it is stored. The * symbol is used to declared pointers.

2. How does a pointer differ from a regular variable?

A regular variable stores a value directly like an int or char. A pointer stores the memory address of another variable so, a pointer indirectly accesses values using that address.

3. What operator is used to get the address of a variable?

The & operator is used to get the address of a variable.It is called the address-of operator. It returns a pointer to the variable's address. This operator allows you to reference variables in memory.

4. What operator is used to access the value stored at a pointer's address?

The * operator is used to access the value stored at a pointer's address. This is called dereferencing the pointer.It allows access to the data pointed to by the pointer. This is how indirect access to values is achieved using pointers.

5. Why are pointers important in C++? Give two uses.

Pointers allow direct access to memory, which makes programs more efficient. They are essential for dynamic memory allocation, enabling flexible use of system resources, pointers help in modifying variables inside functions by passing addresses instead of copies. They are heavily used in building complex data structures like linked lists, trees, and graphs. Overall, pointers give C++ its low-level power and control over system memory.

7. Supplementary Activity

1.

```
int x = 42;
int *ptr = &x;
cout << *ptr;
```

OUTPUT: 42

2.

```
int a = 5, b = 10;
int *p = &a;
p = &b;
cout << *p;
```

OUTPUT: 10

3.

```
int arr[3] = {10, 20, 30};
int *p = arr;
cout << *p;
```
OUTPUT: 10

4.

```
int arr[4] = {2, 4, 6, 8};
int *p = arr;
p++;
cout << *p;
```
OUTPUT: 4

5.

```
int arr[3] = {5, 15, 25};
int *p = arr;
cout << *(p + 2);
```
OUTPUT: 25

Error Spotting
Identify and fix the error(if any) in the codes below.

1.

```
int arr[3] = {1, 2, 3};
int *p = &arr;
```

ERROR: &arr is the error because it gives a pointer to the whole array, not a pointer to an int.

FIXED CODE:
```
int arr[3] = {1, 2, 3};
int *p = arr;
```

2.

```
int arr[5];
int *p;
p = arr[2];
```

ERROR: Here, arr[2] accesses the third element of the array, and its value is an int, not a pointer. So writing p = arr[2]; is trying to assign an integer to a pointer, which is invalid . This could lead to undefined or errors in your code.

FIXED CODE:

```
int arr[5];
int *p;
p = &arr[2];
```

3.

```
int arr[4] = {10, 20, 30, 40};
cout << *arr[2];
```

ERROR: The error in ths code is it tries to dereference arr[2] using *arr[2], but arr[2] is an integer (value 30), not a pointer .Dereferencing an integer as if it were a memory address is invalid and leads to a compile-time error. This happens when you mistakenly use * where it isn't needed.

FIXED CODE:

```
int arr[4] = {10, 20, 30, 40};
cout << arr[2];
```

8. Conclusion

Honestly, this activity was a bit confusing for me at first, especially the output part. Looking at the code and trying to predict the output without actually running it was harder than I thought. I kept mixing up what the pointer was actually pointing to, and sometimes I didn't know if it was showing the value or the address. The error spotting part wasn't easy either, cause the mistakes looked small but they mess up the whole logic. Before doing this, I had to go over my notes and study again how pointers, arrays, and memory addresses actually work in C++. I'm glad I did though, cause now I understand it more than before, but it still feels like something I need to practice more. I've realized that even small things like using * or & wrong can totally change what the code does, so I gotta be extra careful next time.