Activity No. 4.3	
Assignment 4.3: Pointers	
Course Code: CPE007	Program: Computer Engineering
Course Title: Programming Logic and Design	Date Performed: September 21, 2025
Section: CPE11S1	Date Submitted: September 21, 2025
Name(s): Tobias, Lawrence C.	Instructor: Engr. Jimlord M. Quejado

6. Output

1. What is a pointer in C++?

- A pointer is a special variable that doesn't keep the real data. Instead, it keeps the address of where the data is stored. It's like having a map that shows you where the treasure is, instead of holding the treasure itself. We use the * symbol to make a pointer. This way, the pointer can help us reach the real value by following the address.

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

- A regular variable keeps the actual value, like a number or a letter. A pointer keeps the address of another variable, so it reaches the value through that address.

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

- The & symbol is called the address-of operator. It gives the memory address of a variable instead of its value. This way, you can point to where the variable is stored in memory.

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

- The * operator is used to get the value stored at a pointer's address. This is called dereferencing, and it lets us reach the actual data that the pointer is pointing to.

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

- Pointers let us go straight to the computer's memory, which makes programs work faster. They are useful when we want to save and use memory in different ways. Pointers can also change the value of a variable inside a function by using its address. They are important for making things like linked lists, trees, and graphs. In simple words, pointers give C++ more power to control memory.

7. Supplementary Activity

Identify the Output

For each code snippet, predict the output without compiling:

1.

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

The predicted output is 42

2.

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

The predicted output is 10

3.

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

The predicted output is 3

4.

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

The predicted output is 4

5.

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

The predicted output is 25

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

1.

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

The error is: &arr is an error because it points to the entire array instead of pointing to a single integer. To fix it, use either arr or &arr[0] like this :**int *p = arr;**

2.

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

The error is: arr[2] gives the third element of the array, which is an int, not a pointer. So writing p = arr[2]; tries to put an int into a pointer, which is not allowed and can cause errors. To fix it, Use the address of the element like this: **p = &arr[2];**

3.

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

The error is: arr[2] is an integer (30), not a pointer. Writing *arr[2] tries to treat that integer like an address, which is not allowed and causes an error. This mistake comes from using * when it isn't needed. To fix it, just output arr[2] like this: **cout << arr[2];**

8. Conclusion

The first time I did this, I was pretty lost, especially with the output. I kept confusing whether the pointer was pointing to the value or the address, and small mistakes really threw me off. I had to look back at my previous activities about pointers, arrays, and memory to get back on track. After that, things started to make more sense, but I know I still need more practice. I also noticed how even one small mistake with * or & could change the effect of the code.