Assignment 4.3	
Pointers	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic and Design	**Date Performed: 09/22/25**
Section: CPE11S1	**Date Submitted: 09/23/25**
Name(s): James Daniel M. Verano	**Instructor: Engr. Jimlord M. Quejado**

6. Output

- **What is a pointer in C++?**
  - *A pointer variable is a type of a variable where it stores the memory address of another variable. For example, In array values, In order to get the exact address of the variable, using a pointer directly takes its address. That is the main purpose of the pointer.*
- **How does a pointer differ from a regular variable?**
  - *As for the definition above, a pointer variable differs from a regular variable in accordance to what they can hold. For example, a regular variable holds a value, which defines the variable as the declared value, while the pointer holds the address of the variable.*
- **What operator is used to get the address of a variable?**
  - *The "&" is the operator that is used to get the address of a variable.*
- **What operator is used to access the value stored at a pointer's address?**
  - *The operator that is used to access the value stored at a pointer's address is " * "; A dereference pointer.*
- **Why are pointers important in C++? Give two uses.**
  - *In my point of view, according to the said points above, a pointer variable stores the memory address of another variable, thus, it can help in transferring variables into different locations without using too much redundancy. The term that I found close to it, is "**Dynamic Memory Allocation**".*
  - *With the mention of dynamic memory allocation, this provides a support that enhances the efficiency of handling or arrays and functions in terms of using pointers.*

7. Supplementary Activity

1.
```
int x = 42;
int *ptr = &x;
cout << *ptr;
```

   ⇒ *Based on my observation of the code, The output of the code will be the address's value of the X value. we can see the "***ptr**" variable is defining the dereference value of the address of X. Thus, using the dereference pointer, it will be outputting, the value of the address defined in the variable "***ptr**".*

2.
```
int a = 5, b = 10;
int *p = &a;
p = &b;
cout << *p;
```

⇒ *Based on my analysis of the code, The output of the snippet will be the value of the B variable. As you can see, "**int *p = &a**" were indeed defined earlier than "**p = &b**", but the readings of code will always be the latest declaration that was declared / processed. Thus, the dereference pointer "***p**" is reading the latest declaration, which was "**p = &b**", which means, we are getting the dereference value of the address variable of B. In conclusion, the output will be the value of the B variable.*

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

⇒ *Based on my analysis of the code, and also, through research, the output of the code will be the dereference value of the first element of the array, which is 10. According to my research, does dereference output multiple values of an array? It said no. Which means, it can only output 1 value in the array but not randomly. Which means, it's between from the index value of 0 - 2. Given that, getting a dereference value from 1 and 2 in a way that there is a 0 value, which is the first value of the array, doesn't make sense in terms of getting a value. Thus, I identified that the first value of the variable will be the output of the code snippet.*

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

⇒ *With the research that I have in the previous code analysis, knowing that in terms of no index value beside the array variable, the dereference value that will be taken will be the first element of the array variable. But for this example, we received an increment, which means p value was increased by 1, for visuals, "**p = arr[0 + 1]**" 0 is the first value of the index, +1 is the increment. Thus, the new value of p is "**p = arr[1]**". The output of the code snippet will be the element value of the index "**p = arr[1]**", which is the element value " **4** ".*

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

⇒ *This example has correlation to examples 3 and 4. Knowing that we can see here, that the pointer is declared to the arr value without any index value beside the array. Which means, "***p = arr[0]**" as a starting point. Why 0? Recalling on example 3 and 4, when an array is declared without any index value beside it, the first element of the array will be the one taken, which has an index value of 0. So, it's defined as, "***p = arr[0]**". Now, proceeding to the next part, where in "**cout << *(p + 2)**", Which I'm unfamiliar with if it is illegal, but if yes, then I'll interpret it in a point where, the value of p, is increased by 2. Which means, in terms of declaration, p has an index value of 0, which means, adding 2, will be converting the index value into 2 **(0 + 2 = 2)**. The next declaration on the variable is, "***p = arr[2]**". In conclusion, based on the analysis I have provided, the output of the code will be the 3 element value, which is " **25** ".*

----------ERROR SPOTTING----------

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

   ⇒ *Based on my analysis, this example is trying to get the address of an array, which the error occurred in the part of declaring the "***p**". The declaration "***p = &arr**", Is the line code error given that getting the address of an array value without the index, leads to the unidentified address since the array consists of 3 elements. Which means, "**&arr**" is trying to get the address of the whole array, which is impossible. In order for it to work, you need to put an index beside the array. It should be like this: "***p = &arr[0]**". This means that you are trying to getting the address of the array element [0], also known as the first element value.*

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

   ⇒ *In this example code, this example code is quite easy to identify. It is one of the most basic errors in C++. As you can see that, we declared 2 variables, the "**arr[5]**" and the "***p**". However, in the declaration of the processing of assigning variables, the variable "**p**" was never declared. The "**p**" that was declared had a dereference pointer, where in the "**p**" that was assigned in the assigning of variables, was a simple int variable "**p**" that was not declared, which only holds values. To fix the code snippet, you'll just have to either convert it into dereference pointer too, by adding the operator " * " beside the "**p**" or create another point where you'll declare "**p**" however the declaration of the *p will be removed if you declare another variable to declare "**arr[2]**" since it will conflict one another.*

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

   ⇒ *In this example, we can see that the code is trying the dereference value of the "**arr[2]**" directly in the array which is an invalid argument. In order for us to fix the code, we have to declare a pointer variable, link it into the array, and declare the dereference pointer variable to the specific array that we want to get. Just like in the example of 2. For example 5 on the code analysis, and example 2 in the error spotting reference.*

   For Visuals:

   *int *p = arr;*
   **p = arr[2];*
   *cout << *p;*

   ⇒ *This defines that, we defining *p to the arr values, and we are specifying that inside that array, we are defining a specific element of "**arr[2]**". Thus, That will get the element value with the index 2 in the array variable.*

8. Conclusion

⇒ *With all the provided examples and definitions, The purpose of this activity is for you to furtherly assess the main function and purpose of the pointers and their operators. Given that this activity has challenged us into*

analyzing the code snippet without the use of the c++ compilers, we should be able to identify how pointers hold element values and how dereferences refer to the address of the array value. Within the examples above, I have expected a new different mechanism on how pointers take different approaches in taking the address and the dereference value of an array. In addition, I was able to find some of my abilities lacking in analyzing invalid arguments and expressions. With all honesty, I did fair enough in this activity. I lack critical analysis in spotting the errors regarding identifying invalid arguments and expressions. For further improvement, I have to furtherly examine my ability to analyze errors and identify if the argument or the expression is defined as invalid.

note: answer this in a paragraph form not bullet or per question.