Activity No. 4.3	
Pointers	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic and Design	**Date Performed:9/18/25**
Section:CPE11S1	**Date Submitted:9/18/25**
Name(s): Will Stuart D. Ponce Jr.	**Instructor:** Engr. Jimlord M. Quejado

6. Output

What is a pointer in C++?

How does a pointer differ from a regular variable?

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

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

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

1.In C++, a pointer is a special type of variable that stores the memory address of another variable. Instead of holding a direct value like an integer or a character, it holds the location where a value can be found

2.A regular variable stores a direct value, while a pointer stores a memory address where a value is located.

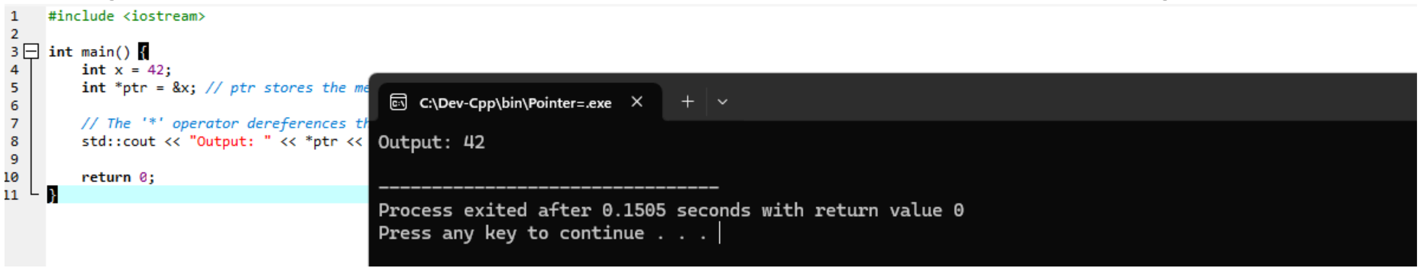
3.The operator used to access the value stored at a pointer's address is the dereference operator, which is the asterisk

4.The operator used to access the value stored at a pointer's address is the dereference operator, which is the asterisk (*)

5.Pointers are important in C++ because they allow for direct memory management, which is crucial for creating high-performance, dynamic, and flexible applications.

7. Supplementary Activity

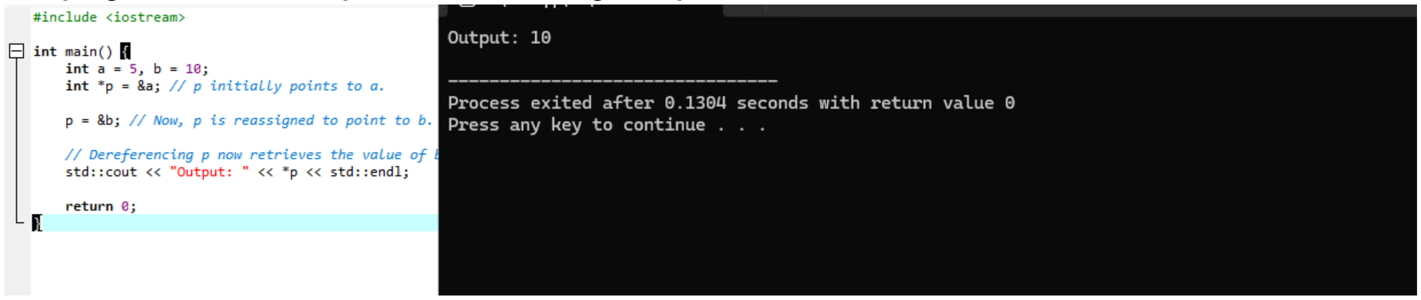
1.
int x = 42;
int *ptr = &x;
cout << *ptr;
This program initializes a pointer to a variable and then prints the value the pointer is pointing to



```
#include <iostream>

int main() {
    int x = 42;
    int *ptr = &x; // ptr stores the me
    
    // The '*' operator dereferences th
    std::cout << "Output: " << *ptr <<
    
    return 0;
}
```

```
Output: 42

--------------------------------
Process exited after 0.1505 seconds with return value 0
Press any key to continue . . .
```

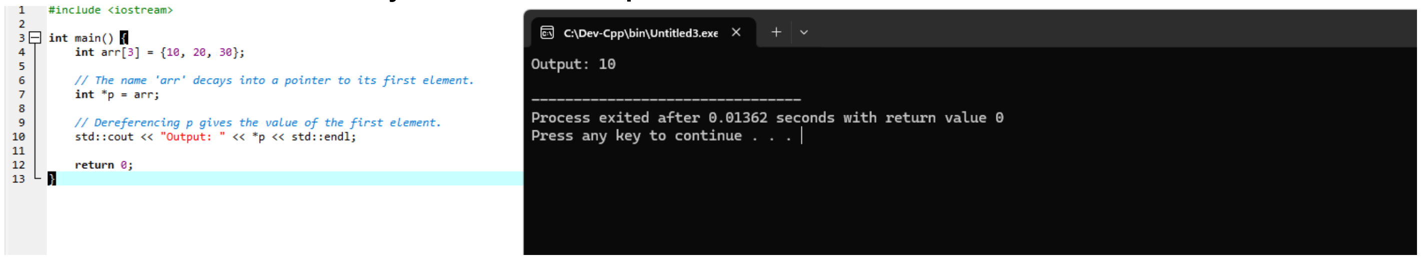
2.int a = 5, b = 10;
int *p = &a;
p = &b;
cout << *p;
This program shows that a pointer can be changed to point to a different variable.



```
#include <iostream>

int main() {
    int a = 5, b = 10;
    int *p = &a; // p initially points to a.
    
    p = &b; // Now, p is reassigned to point to b.
    
    // Dereferencing p now retrieves the value of b
    std::cout << "Output: " << *p << std::endl;
    
    return 0;
}
```

```
Output: 10

--------------------------------
Process exited after 0.1304 seconds with return value 0
Press any key to continue . . .
```

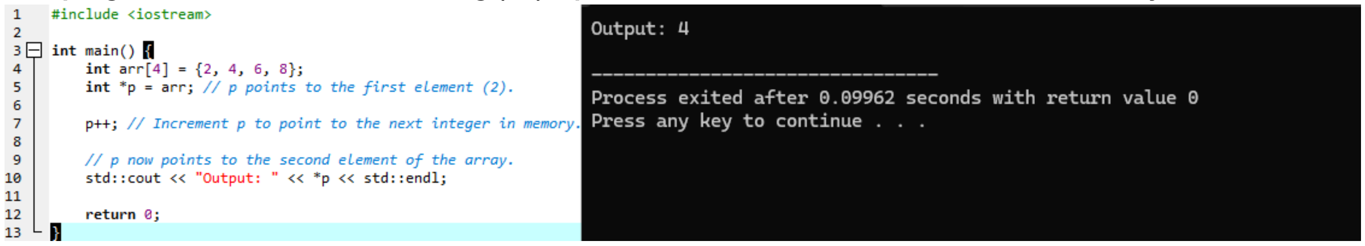
3.int arr[3] = {10, 20, 30};
int *p = arr;
cout << *p;
This demonstrates how an array's name acts as a pointer to its first element.



```
#include <iostream>

int main() {
    int arr[3] = {10, 20, 30};
    
    // The name 'arr' decays into a pointer to its first element.
    int *p = arr;
    
    // Dereferencing p gives the value of the first element.
    std::cout << "Output: " << *p << std::endl;
    
    return 0;
}
```

```
Output: 10

--------------------------------
Process exited after 0.01362 seconds with return value 0
Press any key to continue . . .
```

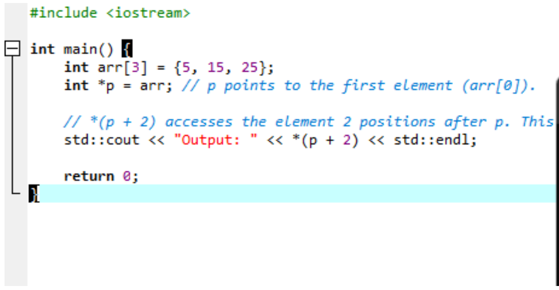
4.int arr[4] = {2, 4, 6, 8};
int *p = arr;
p++;
cout << *p;
This program shows how incrementing (++) a pointer moves it to the next element in an array.



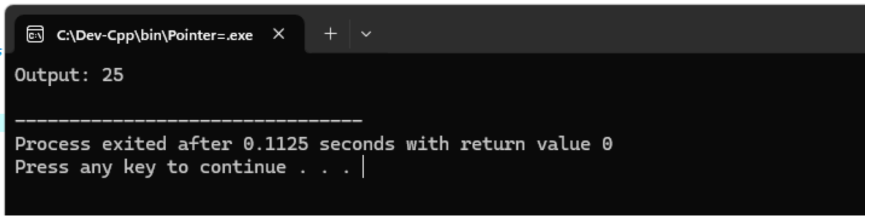
```
#include <iostream>

int main() {
    int arr[4] = {2, 4, 6, 8};
    int *p = arr; // p points to the first element (2).
    
    p++; // Increment p to point to the next integer in memory.
    
    // p now points to the second element of the array.
    std::cout << "Output: " << *p << std::endl;
    
    return 0;
}
```

```
Output: 4

--------------------------------
Process exited after 0.09962 seconds with return value 0
Press any key to continue . . .
```

5.int arr[3] = {5, 15, 25};
int *p = arr;
cout << *(p + 2);
This program uses pointer arithmetic to access an element at a specific offset from the beginning of an array.



```
#include <iostream>

int main() {
    int arr[3] = {5, 15, 25};
    int *p = arr; // p points to the first element (arr[0]).

    // *(p + 2) accesses the element 2 positions after p. This
    std::cout << "Output: " << *(p + 2) << std::endl;

    return 0;
}
```

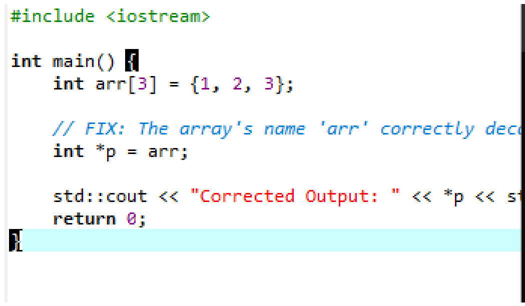


```
Output: 25

--------------------------------
Process exited after 0.1125 seconds with return value 0
Press any key to continue . . .
```

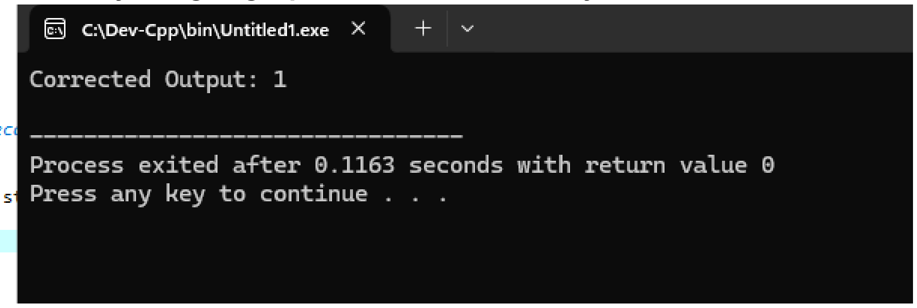
Error Spotting

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

1.int arr[3] = {1, 2, 3};
int *p = &arr;
This section shows the error and fix for incorrectly assigning a pointer to an entire array.

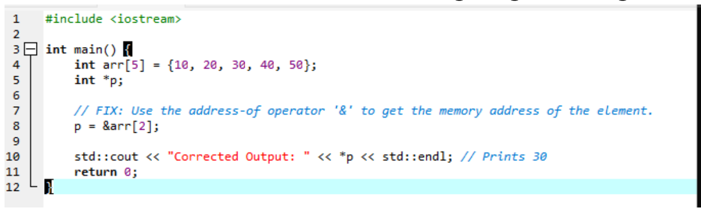


```
#include <iostream>

int main() {
    int arr[3] = {1, 2, 3};

    // FIX: The array's name 'arr' correctly dec
    int *p = arr;

    std::cout << "Corrected Output: " << *p << s
    return 0;
}
```

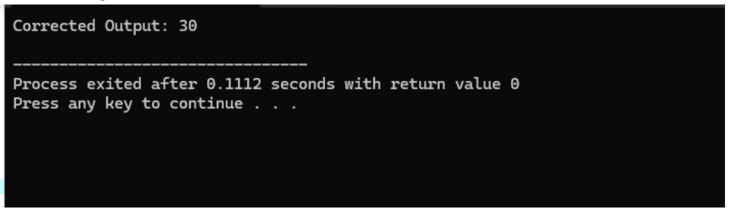


```
Corrected Output: 1

--------------------------------
Process exited after 0.1163 seconds with return value 0
Press any key to continue . . .
```

2.int arr[5];
int *p;
p = arr[2];
This shows the error and fix for assigning an integer value to a pointer variable.

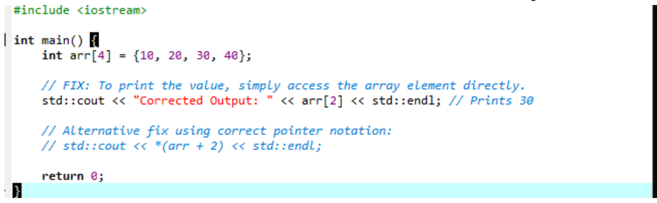


```
#include <iostream>

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *p;

    // FIX: Use the address-of operator '&' to get the memory address of the element.
    p = &arr[2];

    std::cout << "Corrected Output: " << *p << std::endl; // Prints 30
    return 0;
}
```

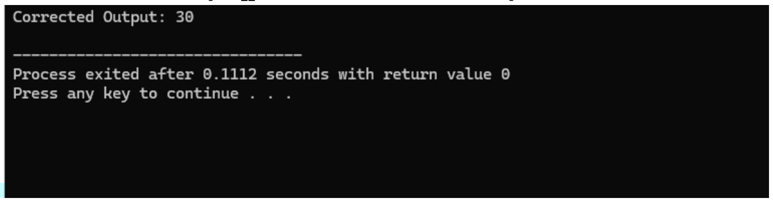


```
Corrected Output: 30

--------------------------------
Process exited after 0.1112 seconds with return value 0
Press any key to continue . . .
```

3.int arr[4] = {10, 20, 30, 40};
cout << *arr[2];
This shows the error and fix related to the precedence of the subscript [] and dereference * operators.



```
#include <iostream>

int main() {
    int arr[4] = {10, 20, 30, 40};

    // FIX: To print the value, simply access the array element directly.
    std::cout << "Corrected Output: " << arr[2] << std::endl; // Prints 30

    // Alternative fix using correct pointer notation:
    // std::cout << *(arr + 2) << std::endl;

    return 0;
}
```



```
Corrected Output: 30

--------------------------------
Process exited after 0.1112 seconds with return value 0
Press any key to continue . . .
```

8.Conclusion:Pointers are C++ fundamental features that allow the user to have direct access to the memory. Although powerful, they require the user's maximum accuracy. The majority of errors occur as a result of type mismatches—mixing a value with an location in memory—and incorrectly identifying the separate functions of the address-of (&) and dereference (*) operators. A strong understanding of the pointers and arrays relationship, as well as pointer arithmetic, is necessary to successfully utilize their indeterminate and efficient applications.