PASS ARGUMENTS IN CPP

1. Pass by Value

Definition:
In **Pass by Value**, the actual value of the argument is passed to the function. The function works on a **copy** of the data, meaning any changes made to the parameter inside the function do not affect the original argument in the calling function.

- **How it works**:
  - When an argument is passed by value, a **new copy** of the argument is created inside the function.
  - The changes made inside the function affect only the local copy of the argument, not the original one.

Syntax:

```
void functionName(type argument) {
    // Function body
}
```

Example:

```
#include <iostream>
using namespace std;

void modifyValue(int a) {
    a = 100;  // Modify the value of a inside the function
    cout << "Inside function, a = " << a << endl;
}

int main() {
    int x = 10;
    cout << "Before function call, x = " << x << endl;
    modifyValue(x);  // Pass by value
    cout << "After function call, x = " << x << endl;  // x remains unchanged
    return 0;
}
```

Output:

```
Before function call, x = 10
Inside function, a = 100
```

```
After function call, x = 10
```

- **Explanation**:
  - `x` is passed to `modifyValue()` by value, so a copy of `x` is created in the function. Changing the copy (`a`) inside the function does not affect `x` in `main()` .

---

2. Pass by Reference

Definition:
In **Pass by Reference**, the function receives a reference to the actual argument, not a copy. Any changes made to the parameter inside the function affect the original argument in the calling function.

- **How it works**:
  - Instead of passing a copy of the argument, a reference (or address) of the variable is passed.
  - Changes made to the parameter inside the function directly modify the original argument.

Syntax:

```
void functionName(type& argument) {
    // Function body
}
```

Example:

```
#include <iostream>
using namespace std;

void modifyValue(int& a) {
    a = 100;  // Modify the value of a directly (it affects the original variable)
    cout << "Inside function, a = " << a << endl;
}

int main() {
    int x = 10;
    cout << "Before function call, x = " << x << endl;
    modifyValue(x);  // Pass by reference
    cout << "After function call, x = " << x << endl;  // x is modified
    return 0;
}
```

Output:

```
Before function call, x = 10
Inside function, a = 100
After function call, x = 100
```

- **Explanation**:
  - `x` is passed by reference to the function `modifyValue()`. Any changes made to `a` (the reference to `x`) inside the function directly modify `x` in `main()`.

---

3. Pass by Pointer (a form of Pass by Reference)

Definition:
Another form of passing arguments by reference is through **pointers**. In this method, a pointer to the argument is passed, allowing the function to access and modify the value stored at the pointer's address.

- **How it works**:
  - A pointer to the argument is passed to the function.
  - The function dereferences the pointer to access and modify the value of the original argument.

Syntax:

```
void functionName(type* argument) {
    // Function body
}
```

Example:

```
#include <iostream>
using namespace std;

void modifyValue(int* a) {
    *a = 100;  // Dereference the pointer and modify the value it points to
    cout << "Inside function, *a = " << *a << endl;
}

int main() {
    int x = 10;
    cout << "Before function call, x = " << x << endl;
    modifyValue(&x);  // Pass by pointer (address of x)
    cout << "After function call, x = " << x << endl;  // x is modified
    return 0;
}
```

Output:

```
Before function call, x = 10
Inside function, *a = 100
After function call, x = 100
```

- **Explanation**:
  - `x` is passed by pointer (`&x`), meaning the function receives the address of `x` .
  - The function then dereferences the pointer (`*a`) to modify the value of `x` .

---

4. Key Differences Between Pass by Value and Pass by Reference

Aspect	Pass by Value	Pass by Reference
What is passed	A **copy** of the argument.	A **reference** to the actual argument.
Effect on Original	Changes in the function do not affect the original argument.	Changes in the function affect the original argument.
Memory Usage	Requires additional memory for the copy.	Does not require extra memory (only the reference/pointer).
Use cases	When you do not want to modify the original data.	When you want to modify the original data or avoid copying large structures (e.g., large arrays or objects).
Efficiency	Less efficient for large objects (like arrays, structs) due to copying.	More efficient for large objects because no copy is made.

---

5. When to Use Pass by Value vs Pass by Reference

- **Pass by Value**:

  - **Use it when** you want the function to work on a copy of the data and do not need to modify the original argument.
  - It's suitable for small data types like `int` , `float` , `char` , or small structs.

- **Pass by Reference**:

  - **Use it when** you want the function to modify the original data, or if the argument is large (like large arrays, structs, or classes) and copying would be inefficient.

- It's also useful when you need to pass large structures or objects and want to avoid the overhead of copying them.

---

6. Const References

If you want to pass by reference but do not want the function to modify the argument, you can use a **const reference**. This allows the function to access the argument efficiently without modifying it.

Syntax:

```
void functionName(const type& argument) {
    // Function body
}
```

Example:

```
#include <iostream>
using namespace std;

void printValue(const int& a) {
    cout << "The value is: " << a << endl;
    // a = 10;  // Error: Cannot modify a because it is a const reference
}

int main() {
    int x = 10;
    printValue(x);  // Pass by reference but cannot modify x
    return 0;
}
```

- **Explanation**:
  - The `const int&` ensures that `a` cannot be modified inside the function, but it avoids making a copy of `x`, making it efficient.

Summary of Pass by Value vs Pass by Reference:

Concept	Pass by Value	Pass by Reference
Function Argument	A **copy** of the argument is passed.	A **reference** (or pointer) to the argument is passed.
Original Data	Original data is **not modified** by the function.	Original data is **modified** by the function.

Concept	Pass by Value	Pass by Reference
Memory	Requires more memory (because of copying).	More memory-efficient (no copy).
Efficiency	Less efficient for large objects due to copying.	More efficient for large objects (no copy).
Use Case	When you don’t want to modify the original data.	When you want to modify the original data or avoid copying large structures.

Choosing between **Pass by Value** and **Pass by Reference** depends on the needs of your program, such as whether you want to modify the original data or minimize memory usage.