📘 C++ References

🔹 What is a Reference in C++?

A **reference** in C++ is an **alias** for another variable. Once a reference is initialized to a variable, it becomes another name for that variable. Any operation on the reference is actually performed on the original variable.

Syntax:

```
type& referenceName = originalVariable;
```

Example:

```
int a = 10;
int& ref = a;  // ref is a reference to a
```

🔹 Key Properties of References

1. **Must be initialized when declared**.
2. **Cannot be null** (unlike pointers).
3. **Cannot be reseated** to refer to another variable after initialization.
4. **Acts as an alias** — changes to the reference affect the original variable.

🔹 Basic Example

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

int main() {
    int a = 10;
    int& ref = a;  // ref is a reference to a

    ref = 20;      // modifies a
    cout << "a = " << a << endl;  // Output: a = 20

    int b = 15;
    //cannot RESEAT the reference to a different variable
    //ex: the code below does NOT make ref a reference to b
    //    it only COPIES b to ref (and in turn to a)
    ref = b;
    cout << "a = " << a << endl;  // Output: a = 15
}
```

🔹 References vs Pointers

Feature	Reference	Pointer
Syntax	`int& ref = var;`	`int* ptr = &var;`
Nullability	Cannot be null	Can be null
Reassignment	Not allowed	Allowed
Dereferencing	Implicit	Explicit (`*ptr`)

🔹 Use Cases

1. Function Arguments (Pass-by-Reference)

```
//& prevents a copy.
//when called in main, x refers to the 'a' variable in main
void increment(int& x) //pass-by reference. prevents a COPY!
{
    x++; // updates 'a' in main
}

int main() {
    int a = 5;
    increment(a);
    cout << a << endl;  // Output: 6
}
```

When to use pass-by-reference

- **when the parameter is a class** (user-defined classes and system classes).
  - why? to prevent a copy of the object. This is especially important for containers (vectors, maps, etc) that might hold a large number of objects.
- **when the method needs to modify the variable from the other scope**.
  - for example...

```
    int MinMaxSum(const std::vector<int>& numbers, int& min, int& max)
    {
        int sum = 0;
        min = max = numbers[0];
        for(int i=1;i<numbers.size();i++)  {
            if(min > numbers[i]) min = numbers[i];
            if(max < numbers[i]) max = numbers[i];
            sum += numbers[i];
        }
        return sum;
    }

    std::vector<int> nums = {1,2,3,4,5,6,7,8,9};
    int minNumber = 0, maxNumber = 0;
    int sum = MinMaxSum(nums, minNumber, maxNumber);
```

2. Function Return Values

```
int& getElement(int arr[], int index) {
    return arr[index];
}

int main() {
    int arr[3] = {1, 2, 3};
    int& num = getElement(arr, 1); //get a reference to the int in the array
    num = 10; // modifies arr[1]
    cout << arr[1] << endl;   // Output: 10
}
```

3. Range-Based For Loops

```
vector<int> nums = {1, 2, 3};
//the loop variable, n, is a reference to the int in the vector
for (int& n : nums) {
    n *= 2; //modifies the int in the vector
}

//the loop variable, n, is a reference to the int in the vector
//it is NOT a copy of each int
for(int& n : nums) {
    cout << n << " ";
} //outputs 2 4 6
```

🔹 Const References

Used to avoid copying large objects and to ensure the referenced value is not modified.

```
//we don't know how big the string is
//therefore, we should NOT copy it (pass-by-reference) AND
//make it const so that we do not modify it
void print(const string& str) {
    cout << str << endl;
}
```

✅ Summary

- References provide a powerful way to alias variables.
- They are safer and more intuitive than pointers in many cases.
- Essential for efficient function parameter passing and object manipulation.

🎯 Quiz!

Here's a short quiz on the topic: quiz

🎨 Markdown Viewer

How to view the markdown files in a browser...

- Markdown Viewer

🧠 Lecture Practices

Here are the lecture Practices...

- Day 1
- Day 2
- Day 3

🔍 Lecture Quizzes

Here are the lecture quizzes...

- Day 1
- Day 2
- Day 3

⚙️ Weekly Topics

Here are the topics for the week...

- Calling Methods
- Calling Methods 2
- Creating Methods
- Iterators
- Vectors
- References
- Const
- Erasing in a Loop
- Default Parameters