1.7 Types of Functions

- **User-defined Functions:**

  Functions created by the programmer to perform specific tasks and improve program modularity and reusability.

- **Standard Library Functions:**

  Predefined functions provided by C++ libraries to perform common operations (e.g., sqrt(), pow(), cin, cout).

1.7.1 Call by Reference

Call by Reference is a parameter-passing technique in which the reference (or alias) of the actual argument is passed to a function. Any modification made inside the function **directly affects the original variable**.

Need for Call by Reference

- **Memory Location**: Both the formal parameter and the actual argument refer/point to the same memory location.
- **Data Modification**: Any changes to the parameter inside the function directly affects the original argument's value.
- **Efficiency**: Call by reference is more efficient for large data structures as it avoids copying the data.
- **Implementation**: This method is implemented differently across languages, for example, using pointers in C, the `&` operator in C++, while Java and Python use variations of pass-by-value

1.7.1 Reference Variables

A reference variable is an **alias for another variable**. Any change made to the reference variable **directly affects the original variable**.

Syntax

```
void function_name(int &x, int &y)
{
   // statements
}
```

Using Pointers Syntax

```
void function_name(int *x, int *y)
{
   // statements
}
function_name(&a, &b);                    // Function Call
```

Reference Version Function Call

```
function_name(a, b);
```

1.7.3 Passing Arrays and Objects by Reference

Large arrays or objects can be passed by reference to **avoid copying and improve performance**.

Any modifications inside the function will **directly affect the original array or object**.

This approach is particularly useful in **firmware design and system modeling**, where memory efficiency is important.

Example Program (Swap Using Call by Reference)

```
#include <iostream.h>
class Addition
{
public:
   void add(int &a, int &b, int &sum)  // Call by Reference
   {
      sum = a + b;
   }
};
void main()
```

```
{
    int x = 10, y = 20, result;
   Addition A1;  // Object name starts with 'A' → A1
   A1.add(x, y, result);
   cout << "Addition = " << result;
}
```

Advantages

1. Original variables are **directly changed** without returning them.
2. No need to explicitly return values from the function.
3. Saves memory and execution time since data is **not copied**.
4. Particularly useful for **large arrays, objects, and structures**.

Disadvantages

1. Changes may occur **unintentionally**, leading to logical errors.
2. Debugging becomes harder because the original data is **modified indirectly**.
3. Using invalid references or pointers may cause errors.
4. Overuse of call by reference can reduce **program clarity and safety**.

1.7.4 Return by Reference

Return by Reference allows a function to **return a reference to a variable** instead of returning a copy. This provides **direct access to the original variable** outside the function, enabling modifications to the variable without creating a new copy.

Need for Return by Reference

Return by Reference is used in the following situations:

1. To provide **high efficiency** when returning large objects.
2. To allow **direct modification** of the returned variable.
3. Useful in **operator overloading** and **chaining operations**.
4. To **avoid the overhead** of creating temporary copies of data.

To return by reference, an ampersand (&) is placed after the return type in the

function declaration.

Syntax

Function Definition

```
datatype& function_name()
{
   return variable;   // must not be a local variable
}
```

Function Call

```
int &ref = function_name();
```

Example Program

```
#include <iostream.h>
class Addition
{
public:
   int& add(int &a, int &b, int &sum)  // Return by Reference
   {
      sum = a + b;
      return sum;
   }
};
void main()
{
      int x = 10, y = 20, result;
   Addition A1;                          // Object name starts with 'A' → A1
   int &res = A1.add(x, y, result);      // Call add() and get reference
   cout << "Addition = " << res;
```

```
}
```

Output:

```
Addition = 30
```

- ✓ add() takes two integers and a reference to sum.
- ✓ The function **returns a reference** to sum.
- ✓ Using int &res = A1.add(x, y, result);, we access the sum via reference.

Key Characteristics and Uses:

- **Avoiding Copies:**

Returning a reference can be more efficient than returning by value, especially for large objects, as it avoids the overhead of creating and copying a temporary object.

- **Modifying Original Data:**

When a function returns a reference, the returned reference can be used to directly modify the original variable to which it refers. This allows the function call itself to be used on the left-hand side of an assignment, effectively changing the original data.

- **L-value Return Types:** Returning by reference is crucial when a function needs to evaluate to an l-value (something that can appear on the left side of an assignment). This is common in overloaded operators, particularly the assignment operator (=).

Advantages

1. Improves performance by **avoiding unnecessary data copying**.
2. Useful for returning **large structures or objects** efficiently.
3. Enables **direct modification** of the returned data.
4. Supports **operator overloading** and other advanced programming techniques.

Disadvantages

1. Returning a reference to a **local variable is dangerous**, because it is destroyed when the function ends.
2. Increases the risk of **unintentional side effects**.
3. The returned reference must remain **valid**, requiring careful memory management.
4. Harder to **debug** if misuse occurs.