SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

23ITT101 – PROGRAMMING IN C & DATA STRUCTURES

I YEAR - II SEM

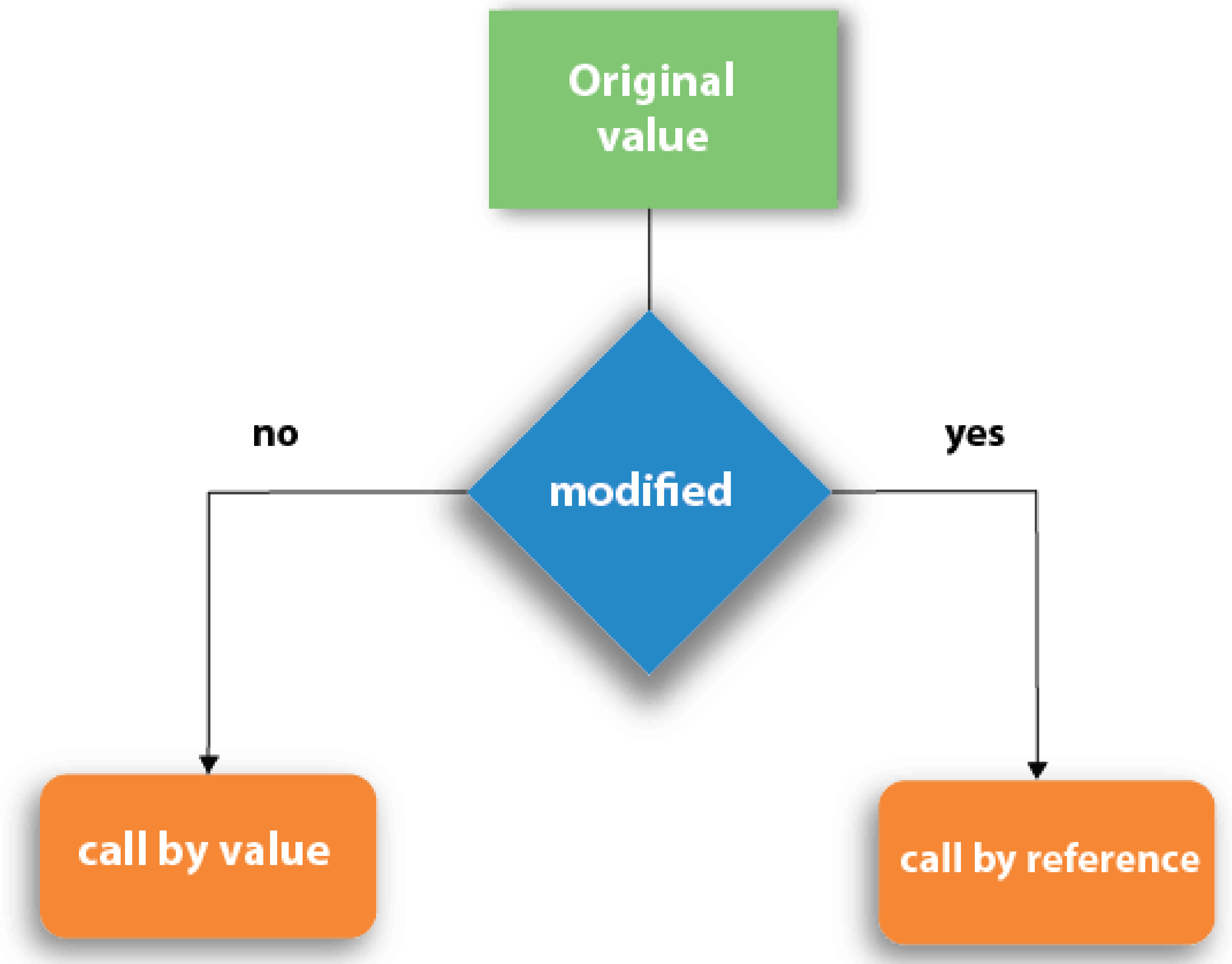
UNIT II – DECISION STATEMENTS & FUNCTIONS

CALL BY VALUE & CALL BY REFERENCE

Introduction

- ✓ In C, a function specifies the modes of parameter passing to it.
- ✓ There are two ways to specify function calls:
  - **CALL BY VALUE**
  - **CALL BY REFERENCE**
- ✓ In call by value, the function parameters gets the copy of actual parameters which means changes made in function parameters did not reflect in actual parameters.
- ✓ In call by reference, the function parameter gets reference of actual parameter which means they point to similar storage space and changes made in function parameters will reflect in actual parameters.

Introduction



Call by Value

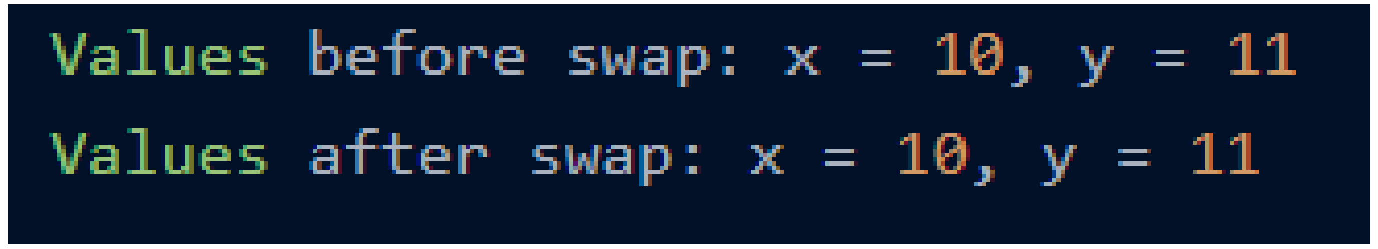
- ✍In call by value method, **the value of the actual parameters is copied into the formal parameters**.
- ✍In other words, we can say that the value of the variable is used in the function call in the call by value method.
- ✍In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- ✍In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- ✍The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Example

```c
#include <stdio.h>

void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

int main()
{
    int x = 10;
    int y = 11;
    printf("Values before swap: x = %d, y = %d\n", x,y);
    swap(x,y);
    printf("Values after swap: x = %d, y = %d", x,y);
}
```



```
Values before swap: x = 10, y = 11
Values after swap: x = 10, y = 11
```

We can observe that even when we change the content of x and y in the scope of the swap function, these changes do not reflect on x and y variables defined in the scope of main. This is because we call swap() by value and it will get separate memory for x and y so the changes made in swap() will not reflect in main().

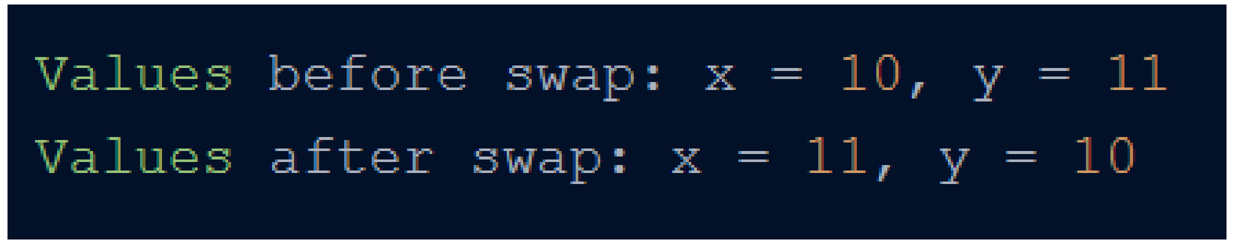
Call by Reference

- Calling a function by reference will give **function parameter the address of original parameter** due to which they will point to same memory location and any changes made in the function parameter will also reflect in original parameters.
- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Example

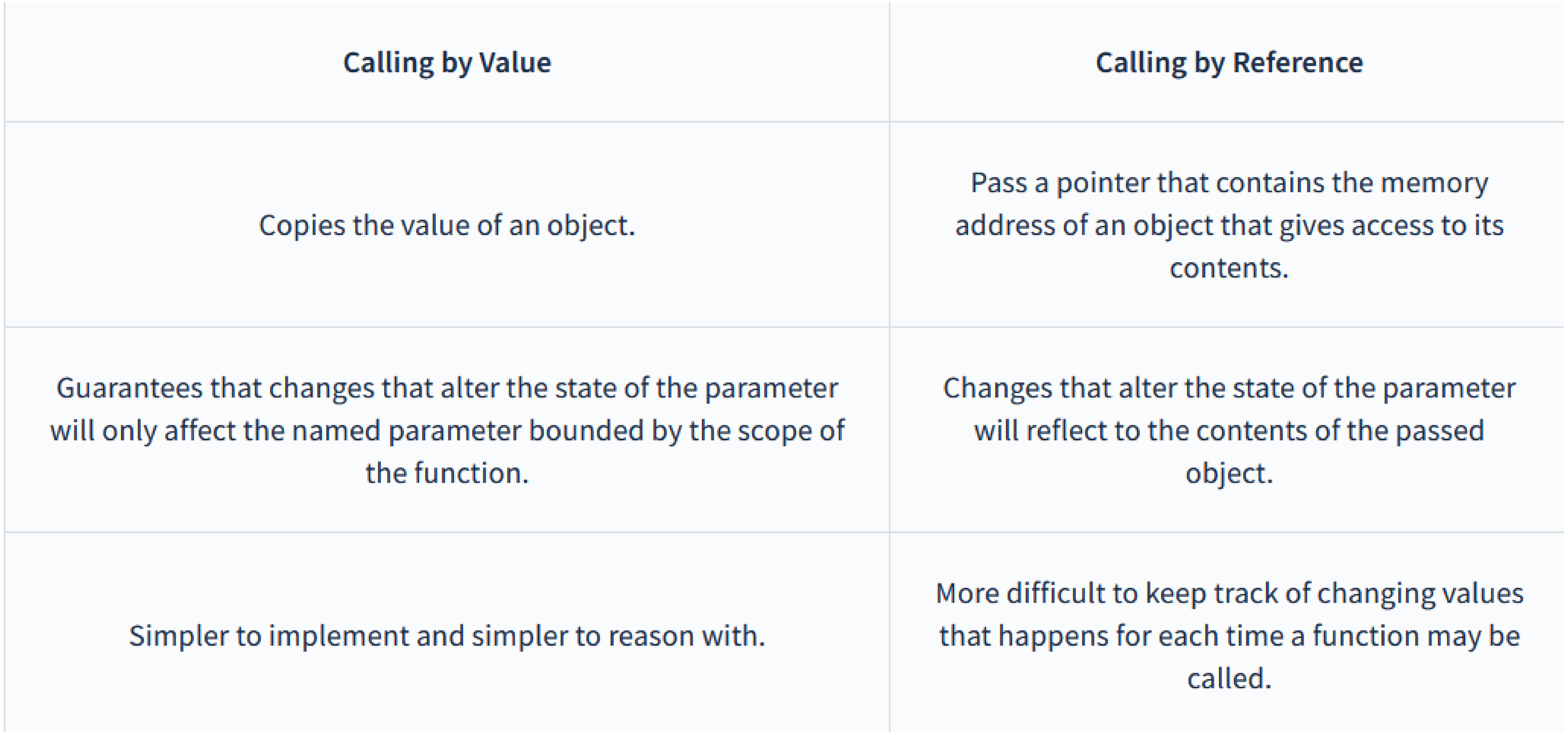
```
#include <stdio.h>
void swap(int *x, int *y)
{
int temp = *x;
*x = *y;
*y = temp;
}
int main()
{
int x = 10; int y = 11;
printf("Values before swap: x = %d, y = %d\n", x,y);
swap(&x,&y);
printf("Values after swap: x = %d, y = %d", x,y);
}
```

We can observe in function parameters instead of using int x,int y we used int *x,int *y and in function call instead of giving x,y, we give &x,&y this methodology is call by reference as we used pointers as function parameter which will get original parameters' address instead of their value. & operator is used to give address of the variables and * is used to access the memory location that pointer is pointing. As the function variable is pointing to the same memory location as the original parameters, the changes made in swap() reflect in main()



```
Values before swap: x = 10, y = 11
Values after swap: x = 11, y = 10
```

Difference between Call by value & Call by Reference



Calling by Value	Calling by Reference
Copies the value of an object.	Pass a pointer that contains the memory address of an object that gives access to its contents.
Guarantees that changes that alter the state of the parameter will only affect the named parameter bounded by the scope of the function.	Changes that alter the state of the parameter will reflect to the contents of the passed object.
Simpler to implement and simpler to reason with.	More difficult to keep track of changing values that happens for each time a function may be called.

When to use Call by value & Call by Reference

Copying is expensive, and we have to use our resources wisely. Imagine copying a large object like an array with over a million elements only to enumerate the values inside the array, doing so will result in a waste of time and memory. Time is valuable and we can omit to copy when:

We intend to read state information about an object,

or

Allow a function to modify the state of our object.

However, when we do not intend our function to alter the state of our object outside of our function, copying prevents us from making unintentional mistakes and introduce bugs.

Advantages of Call by Value

- ✓ Guarantees that changes that alter the behaviour of a parameter stay within its scope and do not affect the value of an object passed into the function.
- ✓ Reduce the chance of introducing subtle bugs which can be difficult to monitor.
- ✓ Passing by value removes the possible side effects of a function which makes your program easier to maintain and reason with.

Disadvantages of Call by Value

- ❑ Incurs performance penalty when copying large objects.
- ❑ Requires to reallocate memory with the same size as the object passed into the function.

Advantages of Call by Reference

- ✓ Calling a function by reference does not incur performance penalties that copying would require. Likewise, it does not duplicate the memory necessary to access the content of an object that resides in our program.
- ✓ Allows a function to update the value of an object that is passed into it.
- ✓ Allows you to pass functions as references through a technique called function pointers which may alter the behavior of a function. Likewise, lambda expressions may also be passed inside a function. Both enable function composition which has neat theoretical properties.

Disadvantages of Call by Reference

- ❑ For every function that shares with the same object, your responsibility of tracking each change also expands.
- ❑ Making sure that the object does not die out abruptly is a serious issue about calling a function by reference. This is especially true in the context of a multithreaded application.

Conclusion

- ✓ There are two ways to pass an argument in C: passing by value and passing by reference. Also known as call by value and call by reference in C.
- ✓ Passing by value copies the content of an object that is being passed into the function. This results in an independent object that exists within the scope of that function. This provides a simpler way to think and reason with our program as we do not allow the function to modify the contents of an object.
- ✓ Passing by reference elides copying and instead passes the memory address of an object. A function may be granted with the privileges to modify the values of an object that is passed onto it.
- ✓ Passing by reference enables a function to accept functions and lambda expressions.
- ✓ Managing object references within the program can be difficult to maintain, which may incur a less maintainable codebase.