Wk5 /S2/ Lecture # : DSOOPS-21

Topics Covered:

- Function Overloading
- Call by Value
- Call by Reference
- Call by Pointer
- Return by Reference

Function Overloading

What Is It?

Creating multiple functions with the same name, but different parameter types or numbers. The compiler uses argument types to pick the correct version.

Why Use It?

You can perform actions (like printing or adding numbers) for different types (e.g., int, double) without inventing new function names every time.

Example:

```
#include <iostream>
void print(int x) {
    std::cout << "Int: " << x << std::endl;
}
void print(double x) {
    std::cout << "Double: " << x << std::endl;
}
int main() {
    print(5);      // Calls print(int)
    print(3.14);   // Calls print(double)
    return 0;
}
```

Key Point:

Functions must differ in type or number of parameters—not just the name!

Call by Value

Definition:

Passing a variable's value to a function. The function works on a copy, so changes inside don't affect the original variable.

Example:

```
void change(int x) {
    x = 10;
}
int main() {
    int num = 5;
    change(num);
    std::cout << num << std::endl; // Still 5!
    return 0;
}
```

Key Point:

The original variable stays unchanged.

Call by Reference

Definition:

The function gets access to the *actual variable* (using the `&` symbol in parameter), so any changes in the function change the original variable.

Example:

```
void change(int &x) {
    x = 10;
}
int main() {
    int num = 5;
    change(num);
    std::cout << num << std::endl; // Now 10!
    return 0;
}
```

Key Point:

Changes “inside” the function are reflected “outside,” in the caller.

Call by Pointer

Definition:

Passing the *address* of the variable to the function, and the function uses `*` to access/change the value at that memory location.

Example:

```
void change(int *x) {
    *x = 10;
}
int main() {
    int num = 5;
    change(&num);
    std::cout << num << std::endl; // Now 10!
    return 0;
}
```

Key Point:

Like reference, but uses pointers and addresses.

Return by Reference

Definition:

When a function returns a reference (`&`), it returns the actual variable (not a copy)—so changes to the result can affect the original.

Example:

```
int& getFirst(int arr[], int size) {
    return arr[0]; // Return a reference to the first element
}
int main() {
    int data[3] = {1, 2, 3};
    getFirst(data, 3) = 99; // Sets first element to 99!
    std::cout << data[0] << std::endl; // 99
    return 0;
}
```

Key Point:

Returning a reference lets you modify the variable in the caller through the function's return value.

Practice Problems and Activities

Easy 1

What will be the output? Why?

```
void fun(int x) { x += 5; }
int main() {
    int a = 10;
    fun(a);
    std::cout << a << std::endl;
    return 0;
}
```

Tip: Is the value of a changed inside fun?

Easy 2

Write a function that swaps the values of two integers using call by reference. Call it in main and show the result.

Template:

```
void swapNums(int &a, int &b) {
    // Your code here
}
int main() {
    int x = 3, y = 8;
    swapNums(x, y);
    std::cout << x << " " << y << std::endl;
    return 0;
}
```

Medium

Fix the function so it doubles an integer using call by pointer, and then use it in main. What will be printed?

```
void doubleValue(___) {
    *x = *x * 2;
}
int main() {
    int num = 11;
    doubleValue(&num);
    std::cout << num << std::endl;
    return 0;
}
```

Hint: What should the parameter list of doubleValue be?

Hard

Write a function that returns a reference to the largest element in an integer array. Use it to change the largest value in the array to 0, and print the array before and after.
Template:

```
int& largest(int arr[], int size) {
    // Write logic to return reference to largest element
}
int main() {
    int data[5] = {4, 1, 7, 3, 2};
    std::cout << "Before: ";
    for(int i=0;i<5;i++) std::cout << data[i] << " ";
    std::cout << std::endl;
    largest(data, 5) = 0;
    std::cout << "After: ";
    for(int i=0;i<5;i++) std::cout << data[i] << " ";
    std::cout << std::endl;
    return 0;
}
```

Hint: Use a loop, keep track of the largest element’s index or reference.