UNIVERSITY OF ENGINEERING & TECHNOLOGY PESHAWAR

(Jalozai Campus)

NAME: AZKA KARIM

REG NO: 24JZELE0557

DEPARTMENT: ELECTRICAL

SUBJECT: COMPUTER PROGRAMMING LAB

TEACHER NAME: Dr SHERAZ KHAN

LAB REPORTS

LAB 11

Lab 11

Topic: More About Functions in C++

Objective

This lab session focuses on understanding how functions in C++ work under different parameter passing methods. I explored function calls by value and by reference, observed how data behaves under each method, and practiced using default parameter values in functions. The aim was to strengthen control over how data is handled within functions and how outputs can be influenced or preserved based on the type of function call.

Calling Function: Call by Value

Call by value is used when the function does not need to modify the original variable. In this method, a copy of the variable is sent to the function. Any changes made inside the function do not affect the original value.

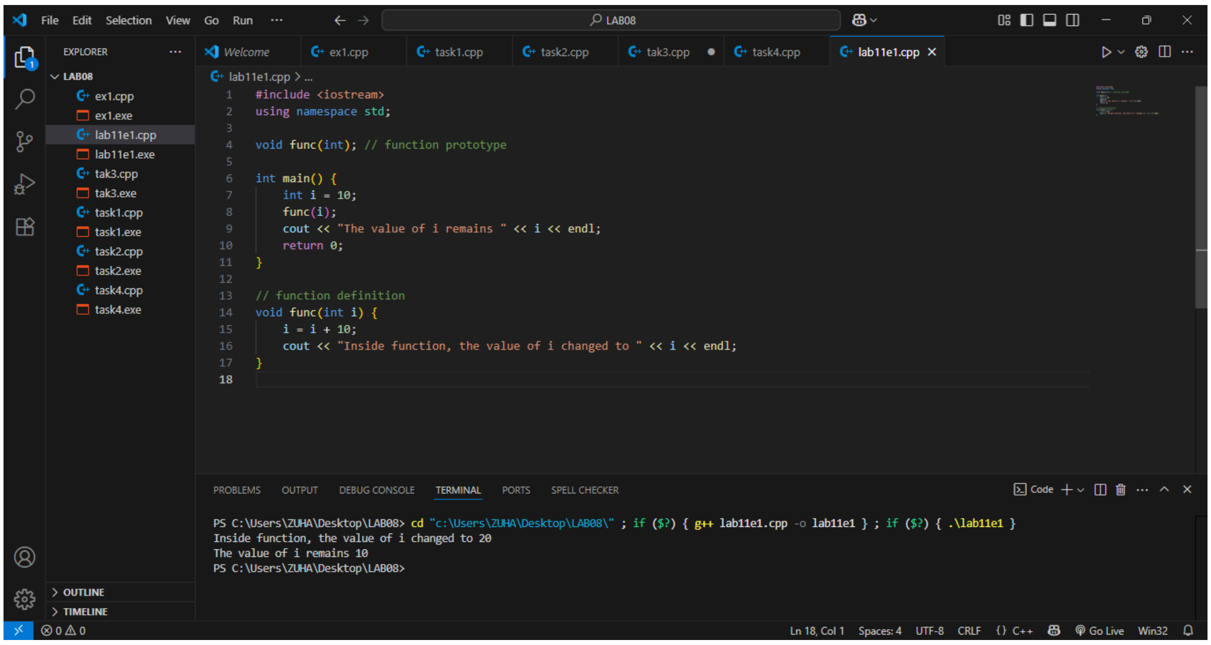
Example Code

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

void func(int); // function prototype

int main() {
   int i = 10;
   func(i);
   cout << "The value of i remains " << i << endl;
   return 0;
}

// function definition
void func(int i) {
   i = i + 10;
   cout << "Inside function, the value of i changed to " << i << endl;
}
```



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

void func(int); // function prototype

int main() {
    int i = 10;
    func(i);
    cout << "The value of i remains " << i << endl;
    return 0;
}

// function definition
void func(int i) {
    i = i + 10;
    cout << "Inside function, the value of i changed to " << i << endl;
}
```

```
PS C:\Users\ZUHA\Desktop\LAB08> cd "c:\Users\ZUHA\Desktop\LAB08\" ; if ($?) { g++ lab11e1.cpp -o lab11e1 } ; if ($?) { .\lab11e1 }
Inside function, the value of i changed to 20
The value of i remains 10
PS C:\Users\ZUHA\Desktop\LAB08>
```

Output

```
Inside function, the value of i changed to 20
The value of i remains 10
```

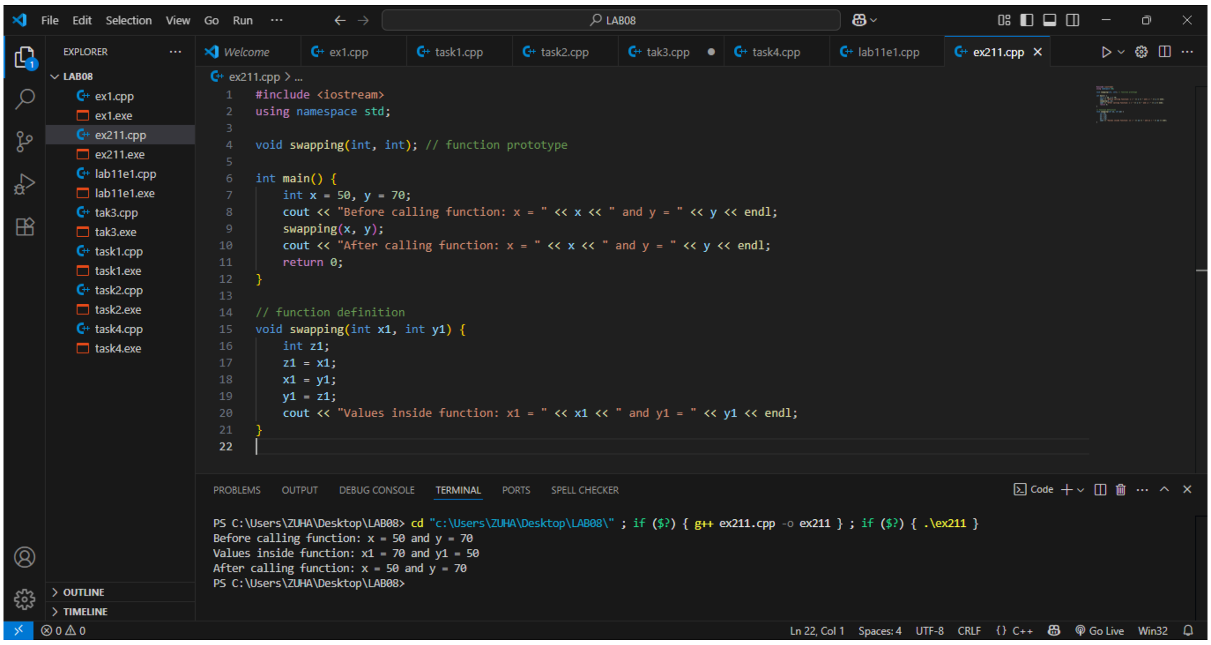
Another Example: Swapping using Call by Value

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

void swapping(int, int); // function prototype

int main() {
    int x = 50, y = 70;
    cout << "Before calling function: x = " << x << " and y = " << y << endl;
    swapping(x, y);
    cout << "After calling function: x = " << x << " and y = " << y << endl;
    return 0;
}

// function definition
void swapping(int x1, int y1) {
    int z1;
    z1 = x1;
    x1 = y1;
    y1 = z1;
    cout << "Values inside function: x1 = " << x1 << " and y1 = " << y1 << endl;
}
```



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

void swapping(int, int); // function prototype

int main() {
    int x = 50, y = 70;
    cout << "Before calling function: x = " << x << " and y = " << y << endl;
    swapping(x, y);
    cout << "After calling function: x = " << x << " and y = " << y << endl;
    return 0;
}

// function definition
void swapping(int x1, int y1) {
    int z1;
    z1 = x1;
    x1 = y1;
    y1 = z1;
    cout << "Values inside function: x1 = " << x1 << " and y1 = " << y1 << endl;
}
```

```
PS C:\Users\ZUHA\Desktop\LAB08> cd "c:\Users\ZUHA\Desktop\LAB08\" ; if ($?) { g++ ex211.cpp -o ex211 } ; if ($?) { .\ex211 }
Before calling function: x = 50 and y = 70
Values inside function: x1 = 70 and y1 = 50
After calling function: x = 50 and y = 70
PS C:\Users\ZUHA\Desktop\LAB08>
```

Output

```
Before calling function: x = 50 and y = 70
Values inside function: x1 = 70 and y1 = 50
After calling function: x = 50 and y = 70
```

Calling Function: Call by Reference

In call by reference, the actual memory address of the variable is passed. This allows the function to modify the original value directly.

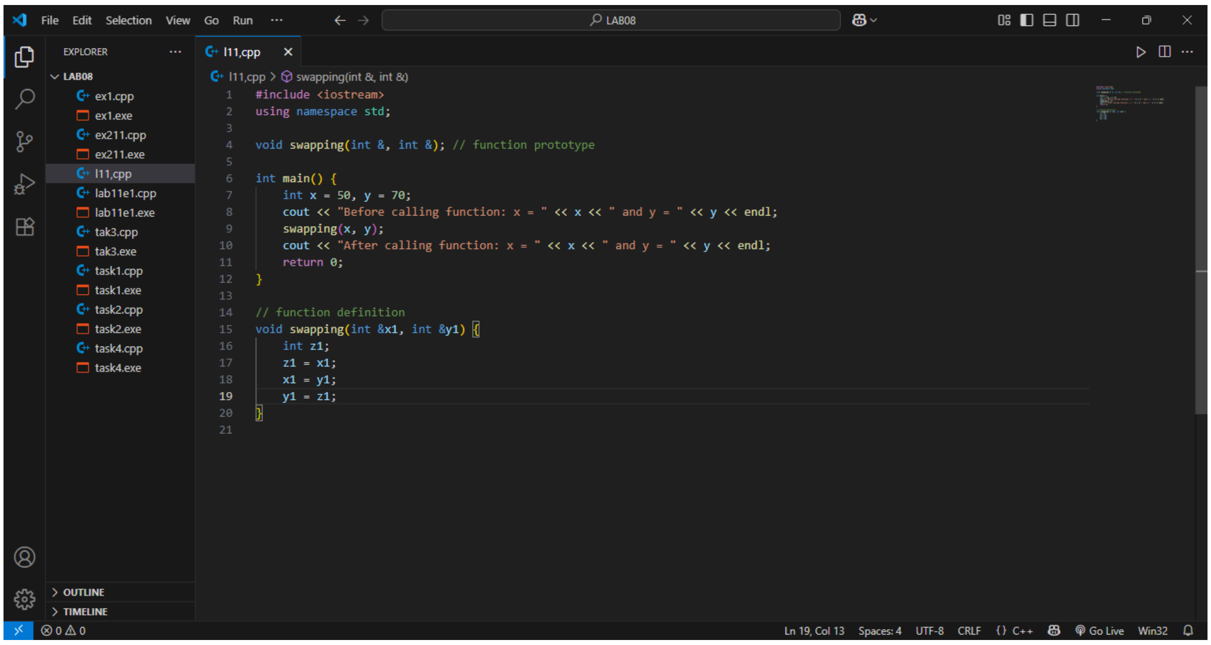
Example Code

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

void swapping(int &, int &); // function prototype

int main() {
    int x = 50, y = 70;
    cout << "Before calling function: x = " << x << " and y = " << y << endl;
    swapping(x, y);
    cout << "After calling function: x = " << x << " and y = " << y << endl;
    return 0;
}

// function definition
void swapping(int &x1, int &y1) {
    int z1;
    z1 = x1;
    x1 = y1;
    y1 = z1;}
```



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

void swapping(int &, int &); // function prototype

int main() {
    int x = 50, y = 70;
    cout << "Before calling function: x = " << x << " and y = " << y << endl;
    swapping(x, y);
    cout << "After calling function: x = " << x << " and y = " << y << endl;
    return 0;
}

// function definition
void swapping(int &x1, int &y1) {
    int z1;
    z1 = x1;
    x1 = y1;
    y1 = z1;
}
```

Output

```
Before calling function: x = 50 and y = 70
After calling function: x = 70 and y = 50
```

Call by reference is useful when you want to return multiple values or modify original data directly.

Default Values in Parameters

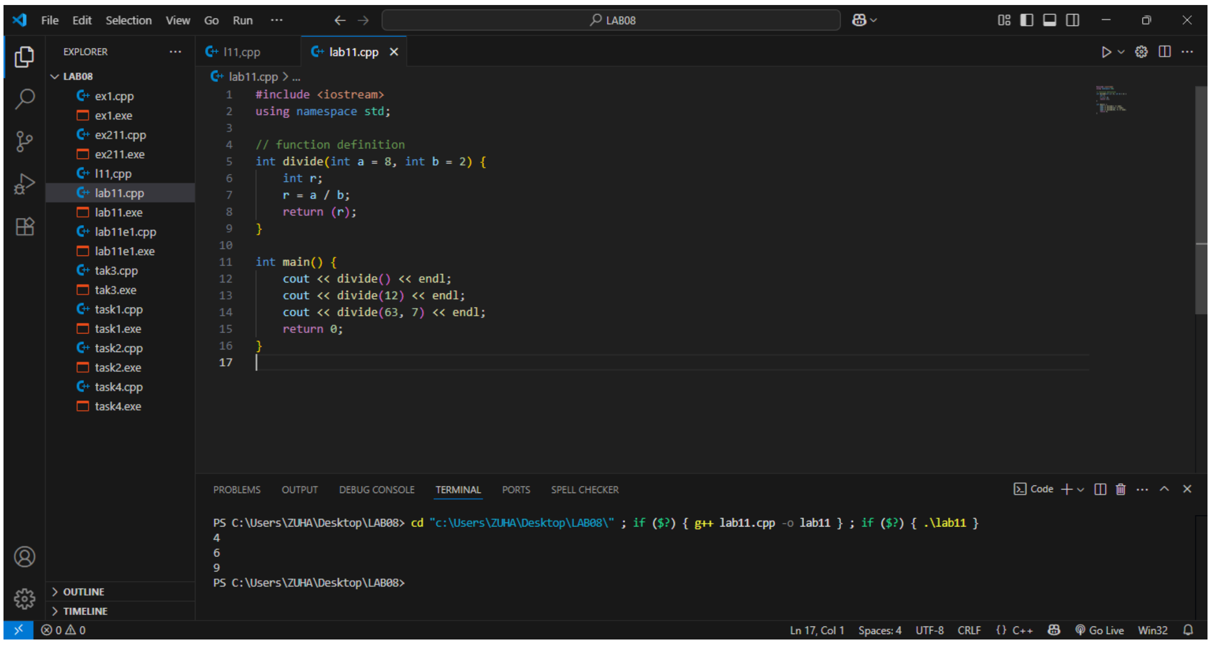
Default parameters allow functions to be called with fewer arguments. If any parameter is not passed during the function call, the default value is used.

Example Code

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

// function definition
int divide(int a = 8, int b = 2) {
    int r;
    r = a / b;
    return (r);
}

int main() {
    cout << divide() << endl;
    cout << divide(12) << endl;
    cout << divide(63, 7) << endl;
    return 0;
}
```



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

// function definition
int divide(int a = 8, int b = 2) {
    int r;
    r = a / b;
    return (r);
}

int main() {
    cout << divide() << endl;
    cout << divide(12) << endl;
    cout << divide(63, 7) << endl;
    return 0;
}
```

```
PS C:\Users\ZUHA\Desktop\LAB08> cd "c:\Users\ZUHA\Desktop\LAB08\" ; if ($?) { g++ lab11.cpp -o lab11 } ; if ($?) { .\lab11 }
4
6
9
PS C:\Users\ZUHA\Desktop\LAB08>
```

Output:

```
4
6
9
```

Lab Tasks

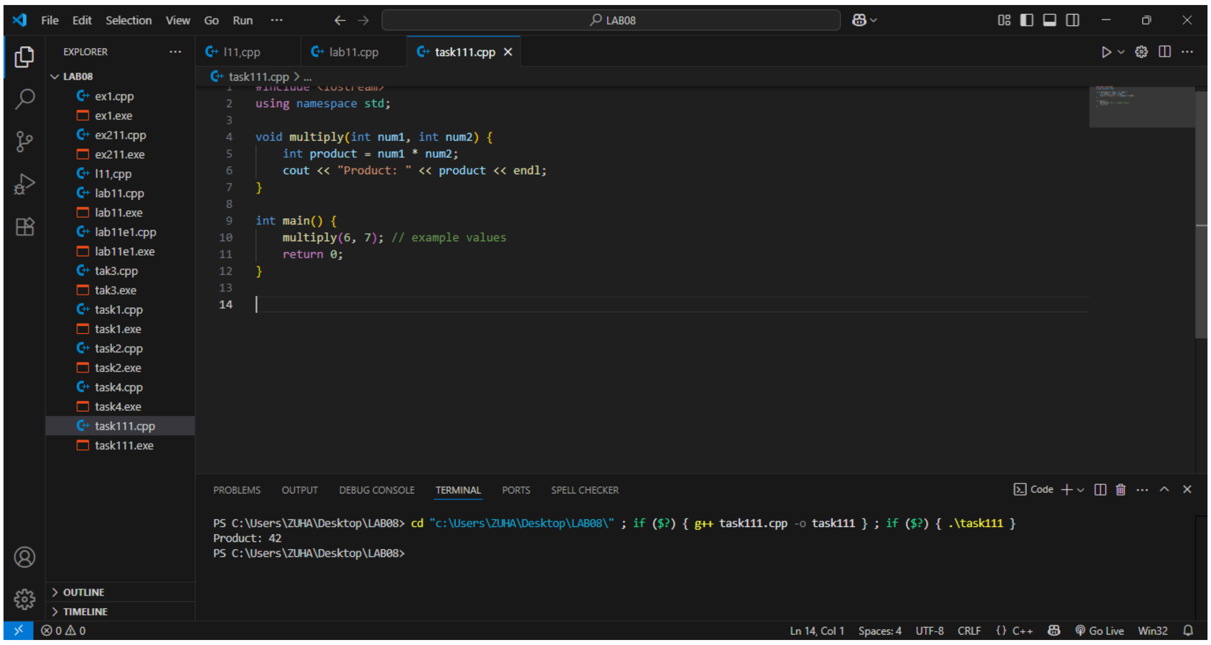
Task 1: Function with Parameters

Code

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

void multiply(int num1, int num2) {
    int product = num1 * num2;
    cout << "Product: " << product << endl;
}

int main() {
    multiply(6, 7); // example values
    return 0;
}
```



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

void multiply(int num1, int num2) {
    int product = num1 * num2;
    cout << "Product: " << product << endl;
}

int main() {
    multiply(6, 7); // example values
    return 0;
}

```

```
PS C:\Users\ZUHA\Desktop\LAB08> cd "c:\Users\ZUHA\Desktop\LAB08\" ; if ($?) { g++ task111.cpp -o task111 } ; if ($?) { .\task111 }
Product: 42
PS C:\Users\ZUHA\Desktop\LAB08>
```

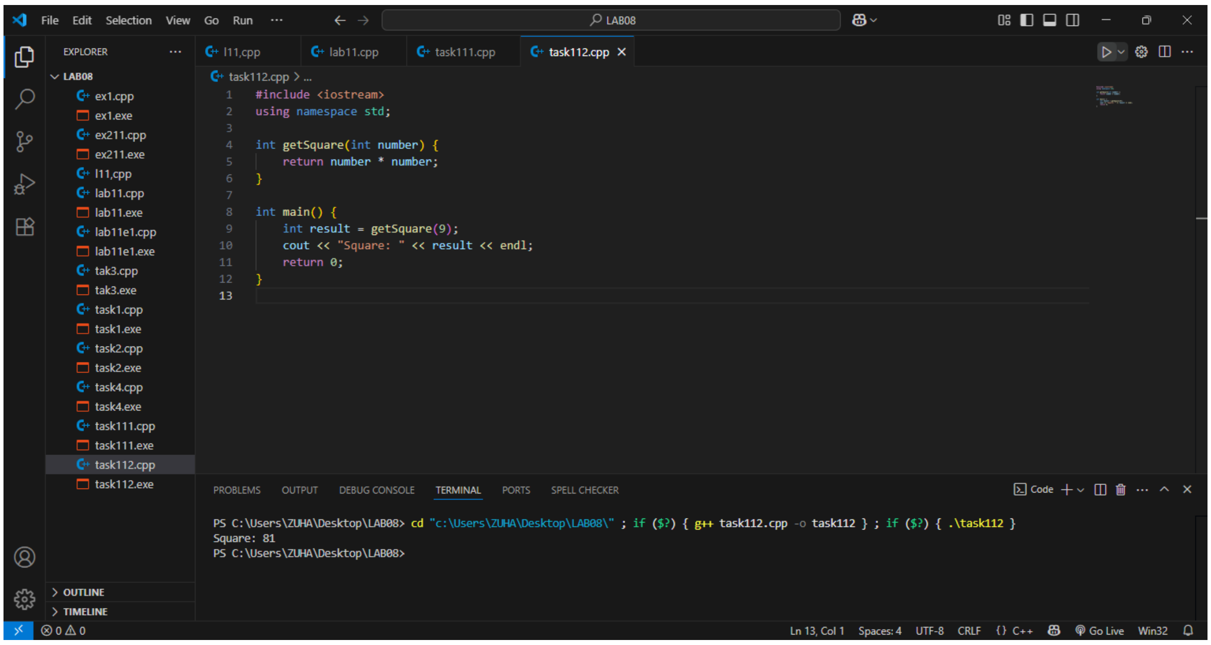
Task 2: Function with Return Value

Code

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

int getSquare(int number) {
    return number * number;
}

int main() {
    int result = getSquare(9);
    cout << "Square: " << result << endl;
    return 0;
}
```



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

int getSquare(int number) {
    return number * number;
}

int main() {
    int result = getSquare(9);
    cout << "Square: " << result << endl;
    return 0;
}
```

```
PS C:\Users\ZUHA\Desktop\LAB08> cd "c:\Users\ZUHA\Desktop\LAB08\" ; if ($?) { g++ task112.cpp -o task112 } ; if ($?) { .\task112 }
Square: 81
PS C:\Users\ZUHA\Desktop\LAB08>
```

Conclusion

This lab provided hands-on understanding of different function call methods in C++. By experimenting with both **call by value** and **call by reference**, I was able to clearly observe how data can be preserved or modified depending on the method used. The implementation of **default parameters** also highlighted how function flexibility can be enhanced. These concepts are crucial for writing modular, reusable, and efficient code in larger C++ programs.