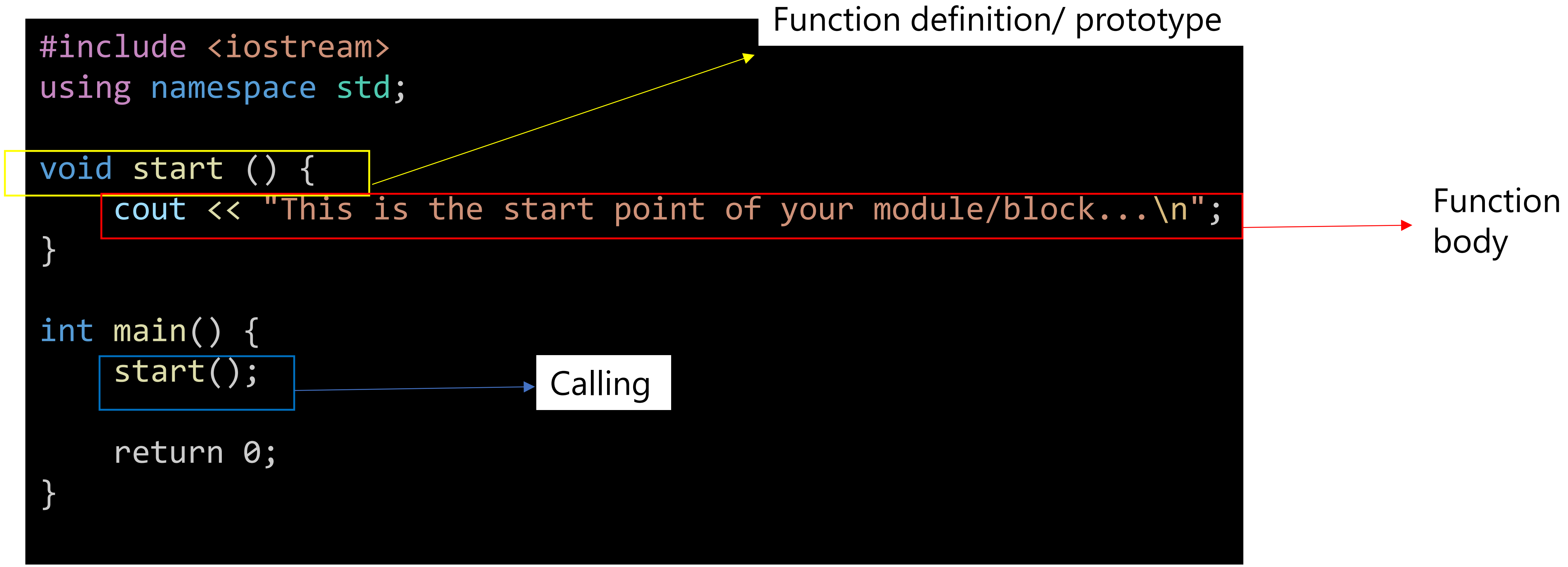
Intermediate programming(C++)- *Lab 4 - Functions*

CSCI112
Nile University

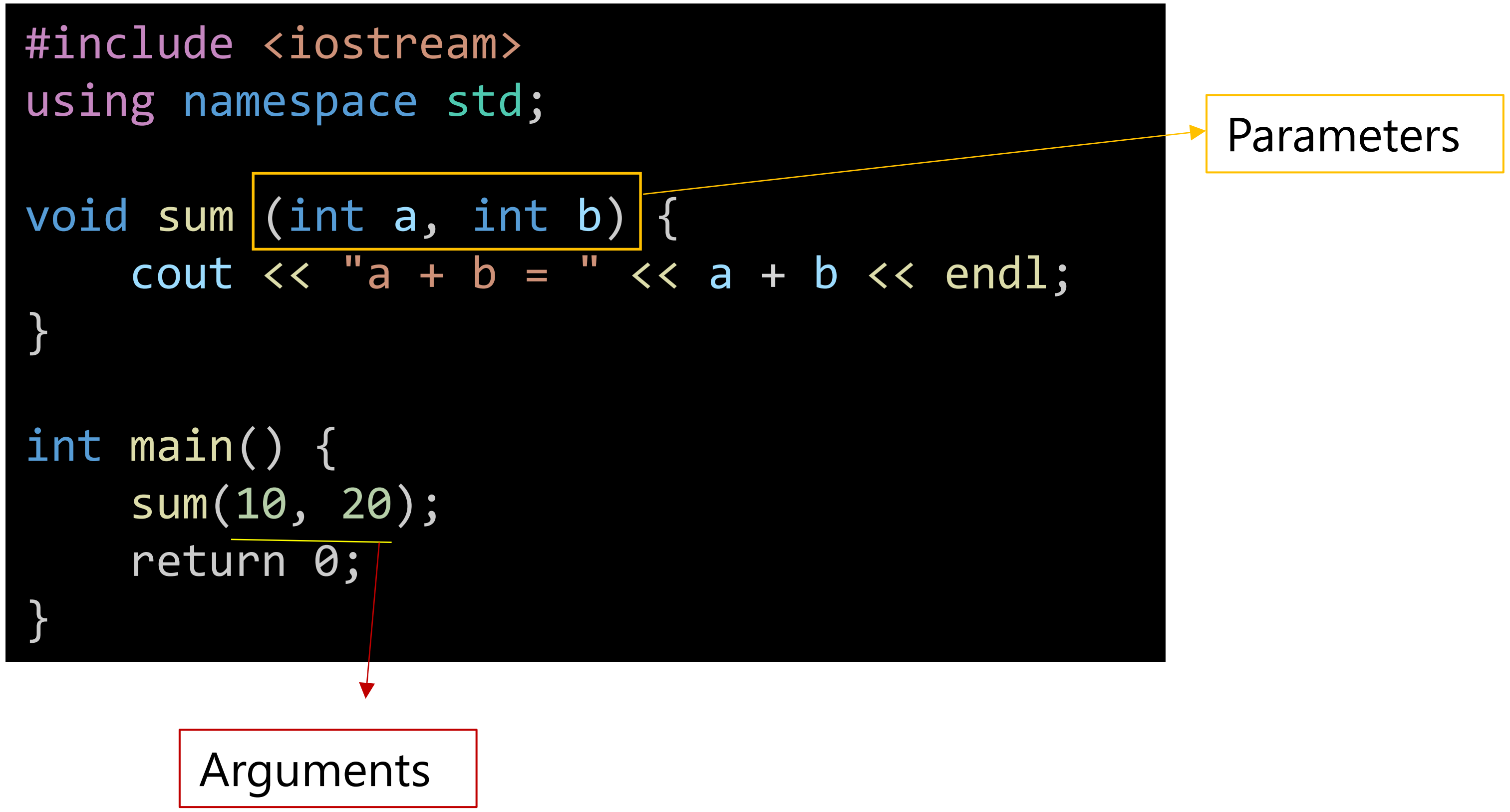
Content

Functions...declaration, definition, and calling



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

void start () {
    cout << "This is the start point of your module/block...\n";
}

int main() {
    start();

    return 0;
}
```

Functions with parameters



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

void sum (int a, int b) {
     cout << "a + b = " << a + b << endl;
}

int main() {
     sum(10, 20);
     return 0;
}
```

main() function

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

void sum (int a, int b) {
    cout << "a + b = " << a + b << endl;
}

int main() {
    sum(10, 20);
    return 0;
}
```

```cpp
void main() {
    sum(10, 20);
    return 0;
}
```

```cpp
int main() {
    sum(10, 20);
}
```

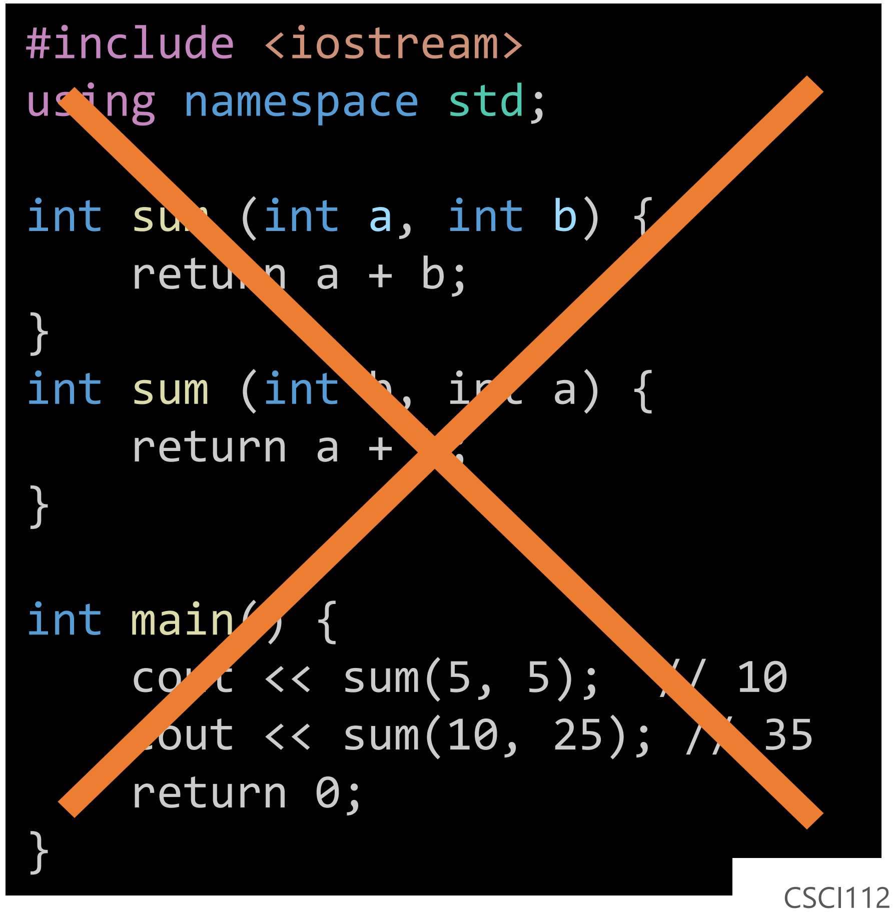
Return value

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

int sum (int a, int b) {
    return a + b;
}

int main() {
    cout << sum(5, 5);  // 10

    int res = sum(10, 20);
    cout << res * 4;    // 120
    return 0;
}
```

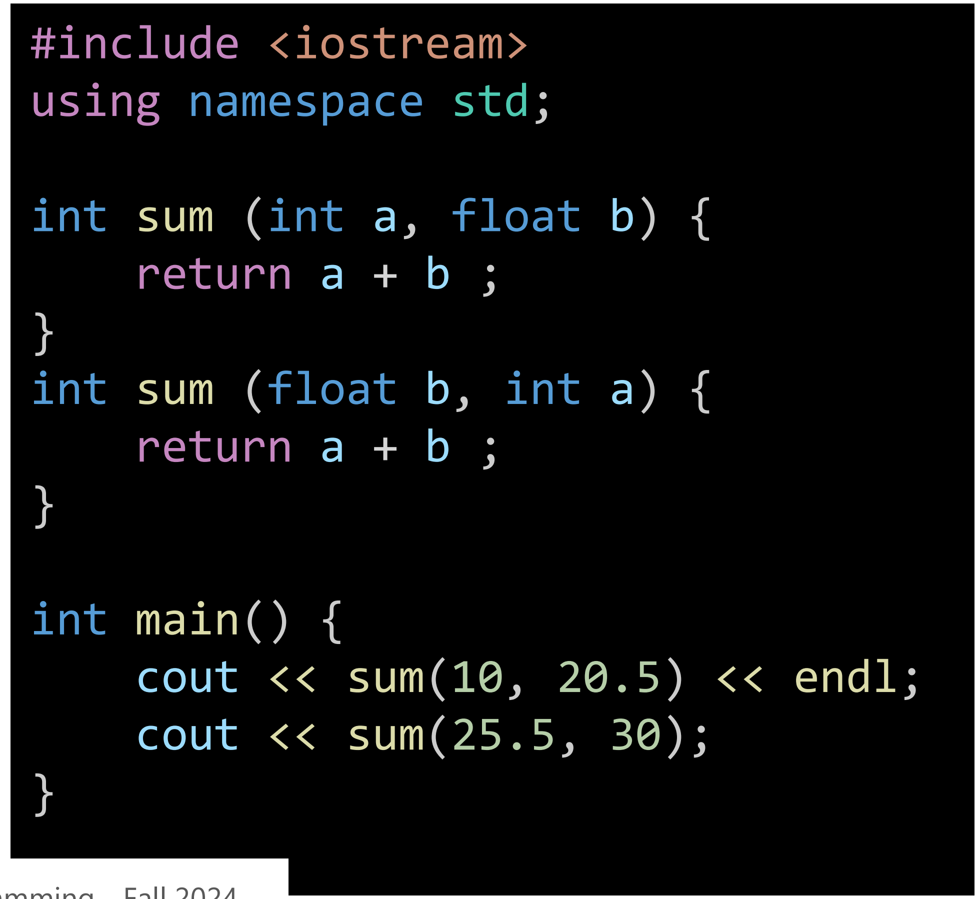
Functions overloading

Overloading means creating a second function with the same name, but in change in parameters number, or parameters data types **ONLY**.

It is not allowed to overload by changing the return type of the function



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

int sum (int a, int b) {
    return a + b;
}
int sum (int b, int a) {
    return a + b;
}

int main() {
    cout << sum(5, 5);   // 10
    cout << sum(10, 25); // 35
    return 0;
}
```



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

int sum (int a, float b) {
    return a + b ;
}
int sum (float b, int a) {
    return a + b ;
}

int main() {
    cout << sum(10, 20.5) << endl;
    cout << sum(25.5, 30);
}
```

Call by value and call by reference

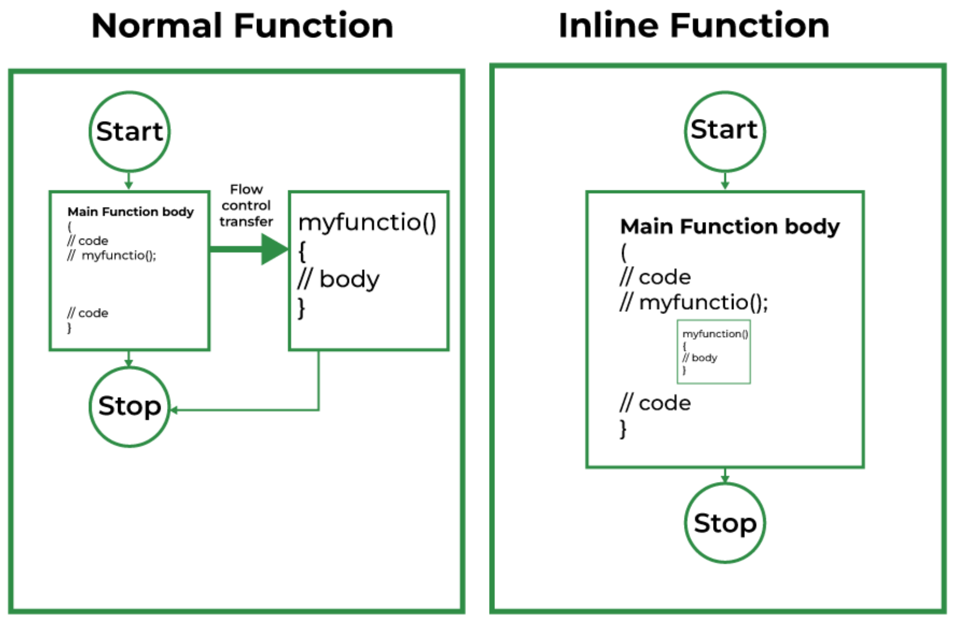
```
#include <iostream>
using namespace std;

void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int a = 5, b = 7;
    swap(a, b);
    cout << a << " " << b;
    // Output: 5 7
}
```

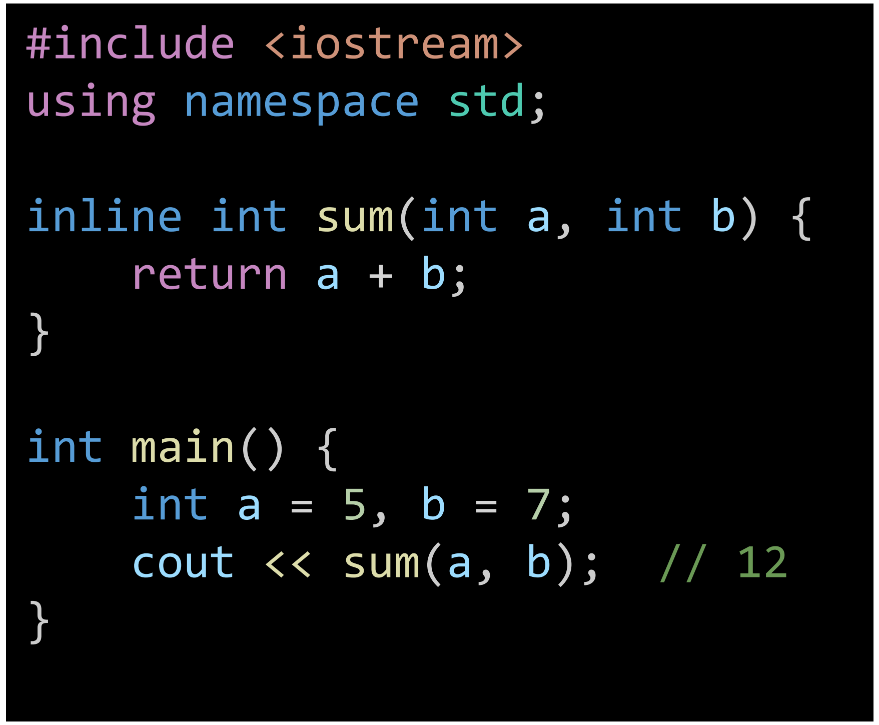
Call by value

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

void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int a = 5, b = 7;
    swap(a, b);
    cout << a << " " << b;
    Output: 7 5
}
```

Call by reference

Inline function





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

inline int sum(int a, int b) {
    return a + b;
}

int main() {
    int a = 5, b = 7;
    cout << sum(a, b);  // 12
}
```

Inline function

When doesn't inline function work?

1) If a function contains a loop. (for, while and do-while)

2) If a function is recursive.

3) If a function return type is other than void, and the return statement doesn’t exist in a function body.

4) If a function contains a switch or goto statement.

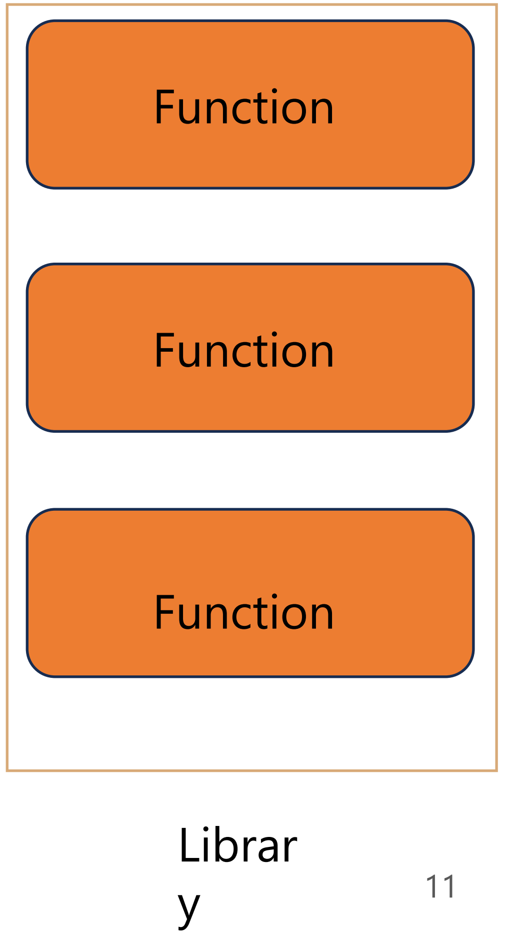
Reference

Built-in Functions

There are many built-in functions supported in libraries in C++.

Examples:

- sqrt() ==> inside cmath library ==> to find the square root of a positive number.
- swap() ==> to swap two numbers (in reference).
- max() ==> to find the maximum of two numbers.
- min() ==> to find the minimum of two numbers.
- Etc….



Task::Implement a Calculator Using Function Overloading

Write a simple calculator program that can perform addition, subtraction, multiplication, and division using function overloading. The calculator should support operations for both integers and floating-point numbers.

Requirements ==> Four Functions:

- Implement overloaded functions for add, subtract, multiply, and divide.
- Each operation should have two versions: one that works with integers and one that works with floating-point numbers.

Input:

The user should be able to input two numbers (either integers or floating-point numbers) and choose the operation they want to perform.

Output:

The program should display the result of the operation.

Division Edge Case:

Handle division by zero appropriately, by displaying an error message when attempting to divide by zero.