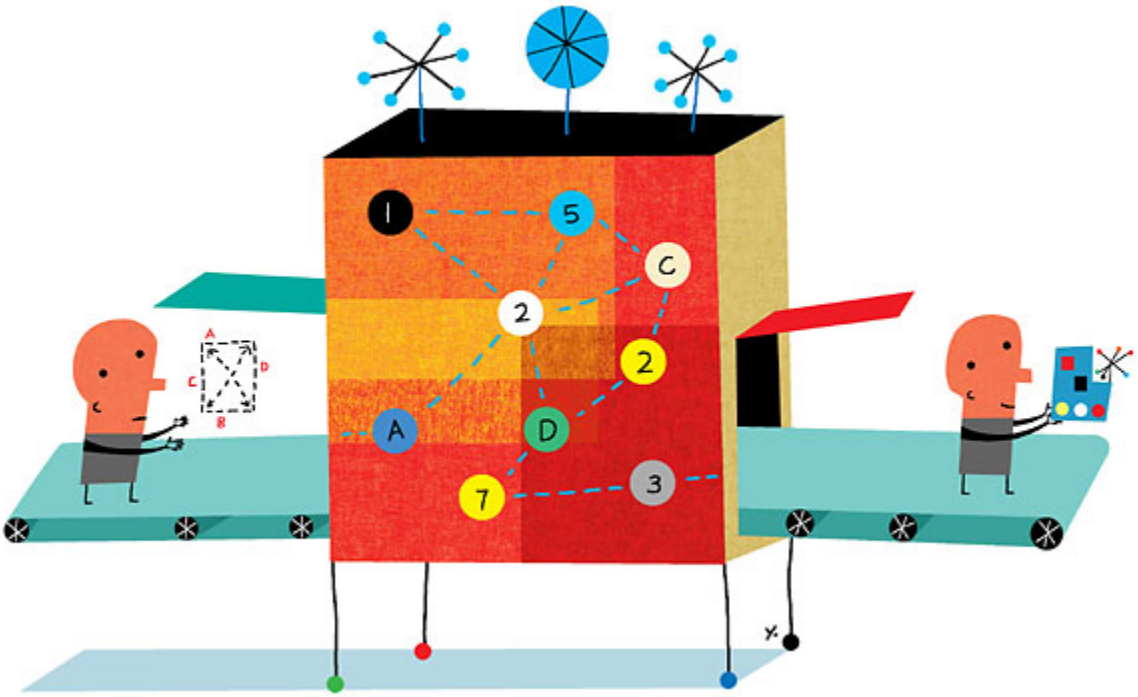
Lab 5. Functions and Prime Numbers



Introduction

A **function** is a named sequence of instructions that performs a specific task and returns the result of its computation. Once defined, it can be then **called** in your program wherever that particular task should be performed.

A function can receive zero or more arguments. For example, consider a function `sum`, which receives three arguments, here named `a`, `b`, and `c`, and returns their sum:

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

To execute (or **call**) a function, you must supply its arguments. For example, if you want to compute the sum of 500, 600, and 700, you can write: `sum(500, 600, 700)`.

A complete program example:

```
#include <iostream>

using namespace std;

/* Defining a function that computes the sum of three integers */
int sum(int a, int b, int c) {
    return a + b + c;
}

int main() {
    // We call it with the actual arguments 1, 20, 300,
    // and save the result in a variable x
    int x = sum(1, 20, 300);
    cout << x << endl;                    // Prints 321
}
```

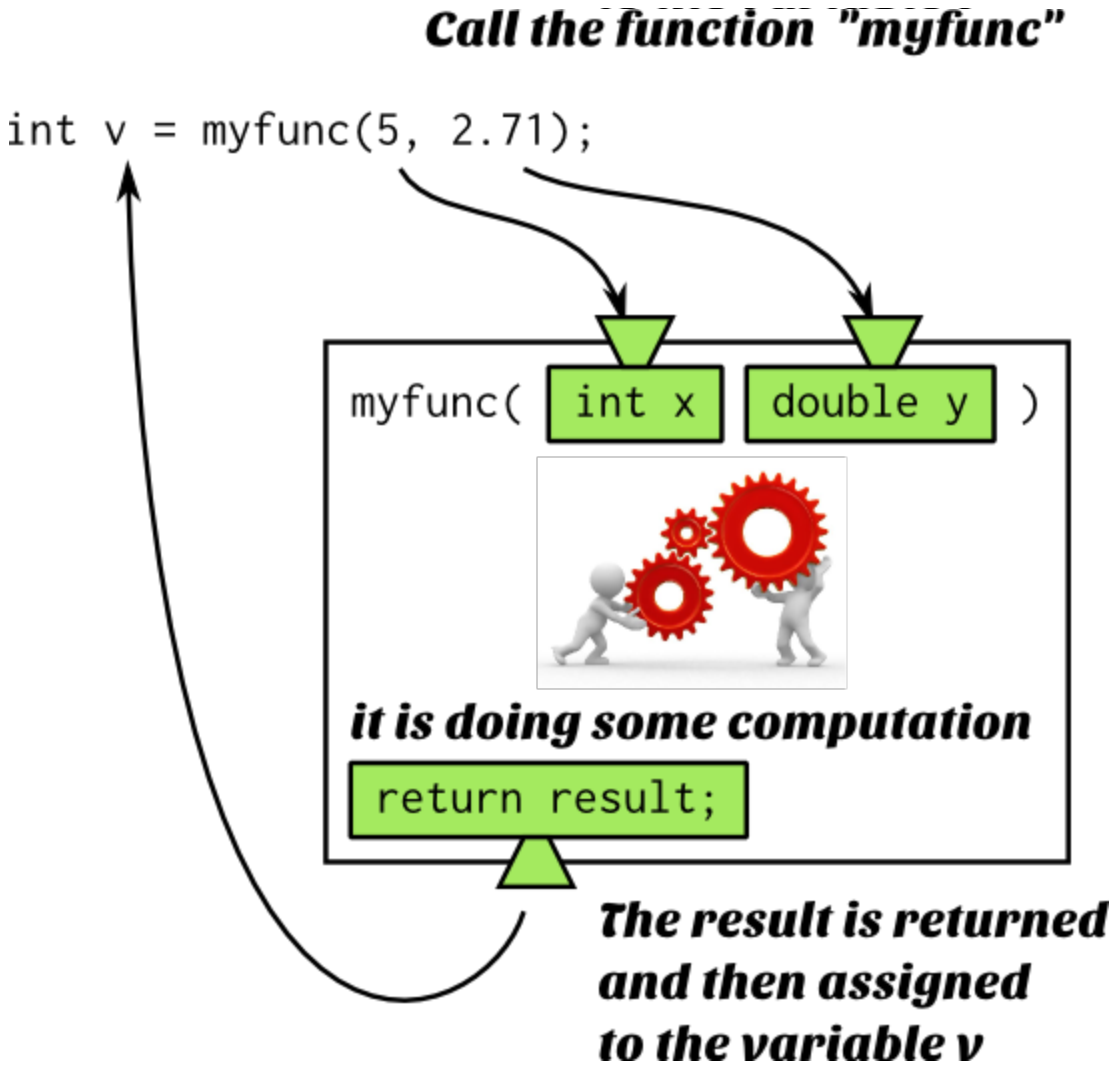
Let's define a function that computes the maximum of two integers:

```
/* Returns the maximum of two arguments */
int max2(int a, int b) {
    if (a > b) {
        return a;
    }
    else {
        return b;
    }
}
```

Then one can find the maximum of thee integers, for example, like this:

```
max2( max2(135, 8763), 500 )   // would return 8763
```

Execution of a function call



Task A. Is divisible?

Write a program `numbers.cpp` that defines a function

```
bool isDivisibleBy(int n, int d);
```

If `n` is divisible by `d`, the function should return `true`, otherwise return `false`.

For example:

```
isDivisibleBy(100, 25) == true

isDivisibleBy(35, 17) == false
```

The program should also have a `main` function that tests your code. For example, it can ask the user to input two integer numbers and print `Yes` if the first number is divisible by the second, otherwise print `No`.

Task B. Is a prime?

A **prime** number is an integer greater or equal to 2 that is only divisible by 1 and by itself. The first few primes are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 …

> ***N* is a prime** if and only if **it is not divisible** evenly by any of the numbers from 2 to *N*−1. Let's implement this decision as a function.

In the same program `numbers.cpp`, add a function

```
bool isPrime(int n);
```

The function should return `true` if `n` is a prime, otherwise return `false`. Change the `main` function to test your new code.

Task C. Next prime

Add a function

```
int nextPrime(int n);
```

that returns the smallest prime greater than `n`.

For example:

```
nextPrime(14) == 17

nextPrime(17) == 19
```

Change the `main` function to test the new code.

Task D. Count primes in range

Add a function

```
int countPrimes(int a, int b);
```

that returns the number of prime numbers in the interval $a \le x \le b$. Change the `main` function to test the new code.

Task E. Is a twin prime?

A prime number *N* is called a **twin prime** if either *N*-2 or *N*+2 (or both of them) is also a prime.
For example, a prime 17 is a twin prime, because 17+2 = 19 is a prime as well.
The first few twin primes are: 3, 5, 7, 11, 13, 17, 19, 29, 31 …

Add a function

```
bool isTwinPrime(int n);
```

that determines whether or not its argument is a twin prime. Change the `main` function to test the new code.

Task F. Next twin prime

Add a function

```
int nextTwinPrime(int n);
```

that returns the smallest twin prime greater than `n`. Change the `main` function to test the new code.

Task G. Largest twin prime in range

Add a function

```
int largestTwinPrime(int a, int b);
```

that returns the largest twin prime in the range $a \le N \le b$.
If there is no twin primes in range, then return `-1`.

For example:

```
largestTwinPrime(5, 18) == 17

largestTwinPrime(1, 31) == 31

largestTwinPrime(14, 16) == -1
```

Change the `main` function to test the new code.

How to submit your programs.

Each program should be submitted through Gradescope.

Write separate programs for each part of the assignment.

Submit only the source code (.cpp) files, not the compiled executables.

Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: your name
Course: CSCI-136
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```