Functions C++

José Mauricio Sevilla[1]
Andrés Camilo Sevilla[1]

[1]Universidad Nacional de Colombia

9 of October of 2015

Functions

The idea of a function in programming is very similar than the defined in mathematics, that needs a set of input and a set of output

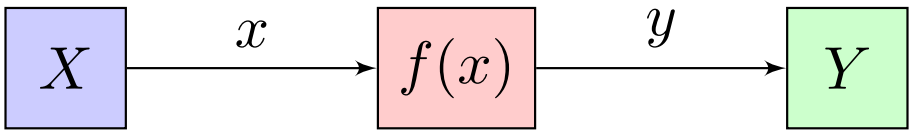

Figure 1: One Variable Function

Where X is the set of input and x belongs to it, and Y is the output set and y one element of it, so, we understand that $f(x)$ has it's domain and range, and that it doesn't work for any set outside of it, the same happens in computing, but the sets of input and output are the kind of variables we defined before.

Scalar

But we can have sometimes more than one set of input

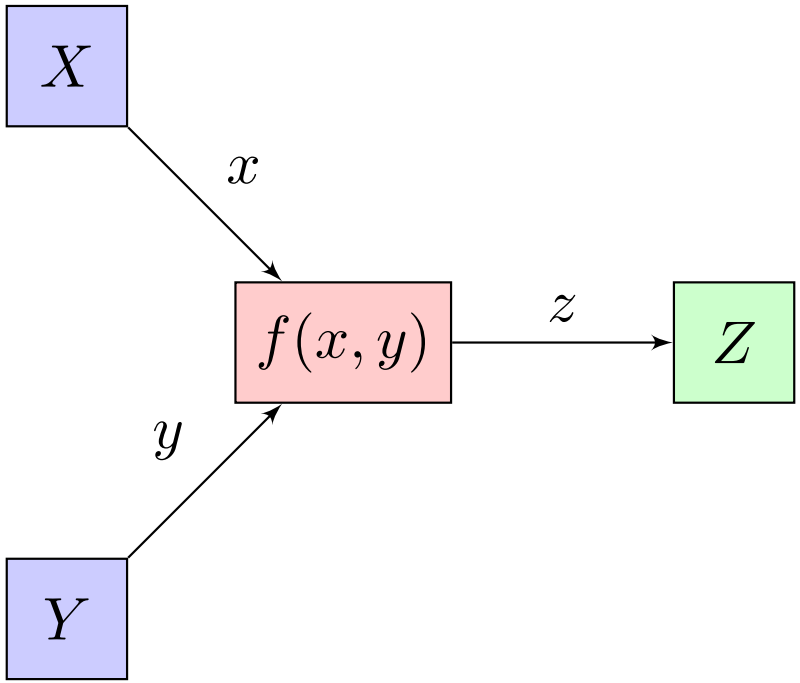

Figure 2: Two Variable Function - a

which is very similar of what we call scalar functions, doesn't matter how, but the function $f(x, y)$ takes two values and return just one, z.

And the input sets can be of the same nature, or a different one

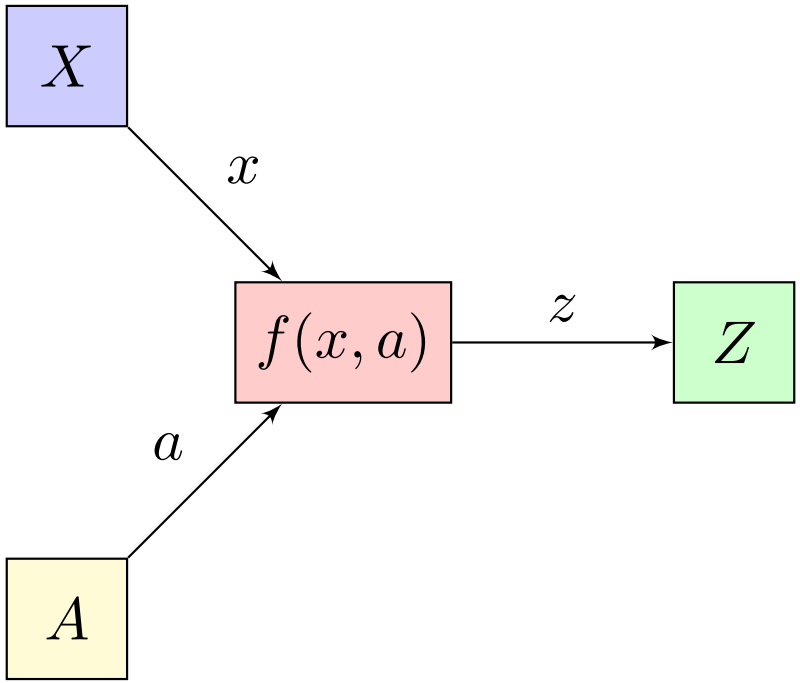

Figure 3: Two Variable Function - b

Where we can think that X and Z are sets of numbers like integers, and A is a set of vowels.

C++ Functions - 1

Definition 1

A Function is a group of statements, which can be called with a name from some part of the program.

The first thing we've got to do is write the output set, (which can be none, for that case, we use `void`), then the name we choose for the functions, then the input parameters and their types

```
type name (type1 parameter1, type2 parameter2,...)
{statements}
```

A program that prints the sum of two integers

```
//integers sum
#include<iostream>
int sum(int a, int b)
{int aux;
aux=a+b;
return aux;}
int main(){
int z;
z=sum(3,2);
std::cout<<z<<std::endl;
int x,y;
x=10;
y=5;
z=sum(x,y)
std::cout<<z<<std::endl;}
```

The important thing is that we can call the function at any time in the

void Functions

a void function is one that doesn't have to return anything, for example, we just want to print message

```
// void function example
#include <iostream>
void printmessage ()
{
  std::cout << "Message";
}

int main ()
{
  printmessage ();
}
```

Arguments passed by value and by reference

In the functions we have write, all the arguments are passed by value, for example in `sum` if we write

```
sum (x,y)
```

where `x` and `y` are integers before defined. And what the function does, es in the temporal variable `a` copies the value of `x`, and in `b` copies the value of `y`, evaluates the function conditions, and then releases the memory occupied by `a` and `b`, so, the values saved in `x` and `y` **Doesn't change**.

But, what if what we want includes a change in any variable passed as an argument?

That's an argument passed by *Reference*, in the lasts cases a copy of the value is made to evaluate the sentences of the function we pass the *value* of the argument, but by *Reference*, we do not pass the value, we pass the *Address* of the variable[1].

[1] Remember that a variable is a physical space in the memory

To use the arguments given by reference, we have to use & in the function definition, so, the syntax goes as follows

```
type name (type1& parameter1,type2 parameter2,...)
{statements}
```

and the parameter1 is taken by reference.

Which advantages does it make?, first, like we do not create any other variable, even if it's temporal, we don't use the memory we don't need, and it's saved for the important variables, so, all of that implies that our code is faster.
Let's test it.

Example

let's use two ways to calculate an operation, so:

```
#include <iostream>
void r(int& a,int& b)
{
  a=*2+b;
}
int main ()
{
int c=10;
int d=11;
r(c,d);
std::cout<<c;
std::cout<<std::endl;
}
```

```
#include <iostream>
int r(int a,int b)
{
        a=*2+b;
        return a;
}
int main ()
{
int c=10;
int d=11;
int aux=r(c,d);
std::cout<<aux;
std::cout<<std::endl;
}
```

Let's see the difference,for doing that we're going to use a terminal function named `time` which says how much time does a command take.

First, in the way we have compiled until now, a file named `a.out` is generated, and it replaces the existing, but, what if we want to have two executable files in the same folder? for doing that, we just say to the compiler we want a different name for a particular executable, that's done by using the `o` flag.

```
g++ FileName.cpp -o ExacutableName
```

for example, if we save the first program as `example1.cpp` and the second one as `example2.cpp`, and we save the executable files as `example1` and `example2` respectively.

And to evaluate the time it takes to run both commands let's type at the prompt

```
time ./example1
```

```
time ./example2
```

Which one is faster?.

If we notice a difference in a code that much simply, do you realize how much fast would it be if we have to use that function a hundred of times?

References


CPLUSPLUS.COM *http://www.cplusplus.com/doc/tutorial/functions/*


CERN *https://twiki.cern.ch/twiki/pub/Main/CollaborativeProjects/Shell.pdf*, 2015


CERN *https://twiki.cern.ch/twiki/pub/Main/CollaborativeProjects/Preprogramming.pdf*, 2015


CERN *https://twiki.cern.ch/twiki/pub/Main/CollaborativeProjects/Basics.pdf*, 2015


CERN *https://twiki.cern.ch/twiki/pub/Main/CollaborativeProjects/Structures.pdf*, 2015