**Official Support Documentation**

Function Range\_me()

**Made by Rui Gomes**

Content

[Purpose 3](#_Toc514842932)

[Pre-requisites 4](#_Toc514842933)

[Input 5](#_Toc514842934)

[Output 6](#_Toc514842935)

[Help from DaBot 8](#_Toc514842936)

# Purpose

The purpose of this document is to explain how the function range\_me() works. As it was presented to us, we were asked to implement an API that would be capable of doing the intersection and union between several intervals and to be able to find specific numbers (integer or decimal) inside those intervals. Here’s the official proposal:

*“You need to provide an API that will allow other developers to create and manipulate ranges of numbers. A range is a pair of numbers that represent the interval between them.*

*So, a range defined by the numbers 3 and 7 will represent an interval that includes all numbers between 3 and 7 (3, 4, 5 and 6 and even non-integer numbers).*

*A range can also be more than one interval. You can also create a range that is both the interval (3, 7) and the interval (10, 12). This range will represent any number between 3 and 7 and any number between 10 and 12 but not numbers between 7 and 10. You must create an API that will represent this concept and allow for the creation of these ranges.*

*It will also allow for the Union operation between two ranges (simple or complex). This will result in a range that includes all the values between the two ranges. For example: union( (3, 7), (6, 10) ) = (3, 10).*

*It will also allow for the Intersection operation between them. This will result in a range that includes only the values that are in both the ranges. For example: intersection( (3, 7), (6, 10) ) = (6, 7).*

*It will also allow the developer to check if a number is contained in a range.”*

In the next pages, we will show you how you can apply this function inside your code by mentioning:

* What are the dependencies to make this function to be working as expected (“Dependencies”);
* What type of variables you will need implement with our function (“Input”);
* What type of result you will get from the function (“Output”);
* How the function is working, with the assistance of DaBot (“Help from DaBot”).

# Pre-requisites

Before you can use this function in your C++ code, you will have to:

- Install and build the library (follow the document named as “HOW\_TO\_BUILD\_LIBS”);

- Add the default libraries, if you still don’t have them:

#include <vector>

#include <string>

- Make sure you have the std namespace called for string and vector;

using std::string;

using std::vector;

- Define the function in your C++ code, as the default procedure:

string range\_me(vector<int> startingVector, string op);

# Input

**Function:** string range\_me(vector <int> vectorInput, string stringInput);

**Input variables**: vector <int> vectorInput, string stringInput;

* **Vector <int> vectorInput:**

For this function to work with the intervals, you will have to present it with a vector. With this, you will be able to tells us:

1. **How many intervals the function needs to work with;**
2. **The minimum and the maximum values for each interval.**

Your vector needs to be built sequentially, so that the function understands the beginning and the end of each interval. For example:

**Intervals**: [1 4] and [2 5];

**Vector**: {1,4,2,5} with size of 4.

* **String stringInput:**

As we want to make this function as dynamic as possible, in the **stringInput** variable, you’re allowed to choose what actions you want to apply to your intervals:

1. **Intersection (“i”);**
2. **Union (“u”);**
3. **Finding a number (“f<number>”).**

This means that you can specify only the actions that you actually need. The **stringInput** is organized in the following way: **“i u f<number>”.** It’s important that each action is divided by a space and the **“f<number>”** has the number that you want to find attached to the **“f”** letter.

To better explain the functionality of this string, let me share some examples:

* **If you want to apply the intersection to your intervals:** “i”;
* **If you want to apply intersection and union to your intervals:** “i u”;
* **If you want to apply union to your intervals and find the number 3.2:** “u f3.2”;
* **If you want to apply all actions:** “i u f3.2”.

# Output

**Function:** string range\_me(vector <int> vectorInput, string stringInput);

**Output variable**: string result;

* **String result:**

This function will present its result based on a string. This string will be based on the following parameters: **“<interval after intersection>,<interval after union>,<find in initial range, in interval after intersection, in interval after union>”.**

If we go through this string parameter by parameter:

1. **<interval after intersection>** = interval that was resulted after applying the intersection. If you have more than 2 intervals and you want to apply this functionality, then all of the intervals will be intersected in sequential way.

For example, here are the possible results you can get:

[1 2] [1 4] => **Result**: [1 2]

[1 2] [2 4] => **Result**: [2 2]

[1 5] [2 6] [1 4] => [2 5] [1 4] => **Result**: [2 4]

[1 2] [3 4] => **Result**: [0 0] **(intersection failed)**

1. **<interval after union>** = interval that was resulted after applying the union. If you have more than 2 intervals and you want to apply this functionality, then all of the intervals will be suffered by union in sequential way.

For example, here are the possible results you can get:

[1 2] [1 4] => **Result**: [1 4]

[1 2] [2 4] => **Result**: [1 4]

[1 5] [2 6] [1 4] => [1 6] [1 4] => **Result**: [1 6]

[1 2] [3 4] => **Result**: [0 0] **(Union failed)**

1. **<find in initial range, in interval after intersection, in interval after union>** = result for finding the input number on each of the ranges (the initial range, the interval after intersection and the interval after union). The result is implemented as a character, which can be **“t”** as for successful finding, **“f”** as for unsuccessful and, finally, **“n”** meaning it’s not applicable because the intersection/union weren’t initially called.

For example, if you want to find the number 2.1, here are the possible results you can get:

**vectorInput** = [1 2] [1 4]

**i** = [1 2] [1 4] => Result: [1 2]

**u** = [1 2] [1 4] => Result: [1 4]

**f2.1** => 2.1 is inside the range of **vectorInput**;

2.1 is not inside the range of **i;**

2.1 is inside the range of **u**;

**Final result**: “t,f,t”.

**vectorInput** = [1 2] [2 4]

**u** = [1 2] [2 4] => Result: [1 4]

**f2.1** => 2.1 is inside the range of **vectorInput**;

2.1 is not to be found because there’s no **i**;

2.1 is inside the range of **u;**

**Final result**: “t,n,t”.

**vectorInput** = [1 2] [2 4]

**u** = [1 2] [2 4] => Result: [1 4]

**With no finding**

**Final result**: “”.

To summarize the output of the **range\_me()** function, your final result will be based, accordingly to the explanation done above, on the following example:

**vectorInput** = {1,2,1,4};

**stringInput** = “i u f2.1”;

**String result** = “1 2,1 4,t,f,t”

In this way, you will be able to select the output that you need to work with in your C++ code, by selecting the **string’s position**.

# Help from DaBot

Optionally, it was decided to implement another static library named “**Project\_MiniclipDaBot\_RuiG**” which calls for the same function, but with a simple assistance through a user interface (based on the console application) to better explain how the intersection, union and finding a number works. With that, the library installation is the same as what’s defined in the “**HOW\_TO\_BUILD\_LIBS**” document file with a minor change **(Note: we highly suggest building this library into a new solution for testing purposes):**

1. In step **4.2.1)**, instead of copying the file **"Project\_Miniclip.lib",** you will need to copy the library called **“Project\_MiniclipDaBot\_RuiG.lib”** to your new solution’s project directory under the Libraries folder, as we advised;
2. Follow the rest of the steps as they are defined;
3. In step **4.2.4.3)**, instead of adding the library name **“Project\_Miniclip.lib”**, you will have to add this library **“Project\_MiniclipDaBot\_RuiG.lib”;**
4. Finally, add this function in your cpp code file as, to properly call it in your **main()** function:

int range\_me\_withDaBot();

Note that the function uses a different name **“range\_me\_withDaBot()”** and it doesn’t need any input variables. This means that you will be prompted to define your range (or even better, your **vectorInput** variable’s data) and the actions you want to apply to it (the **stringInput** variable’s data) when you execute this function.

Basically, simply call **“range\_me\_withDaBot()”** and the “DaBot” will assist you on how to move forward.

**WARNING:** This library should **NOT** be called to act as a replacement for the official **range\_me()** function. This is simply a project to help the developers understand how this is working.