Programming challenges in C++

7. Using functions and improving sorting algorithms

Nacho Iborra

IES San Vicente

Table of Contents

Programming challenges in C++

1. Function definition in C++

If we want to create a function in C++, we have to specify its return type, function name and parameters. For instance:

```
int addNumbers (int n1, int n2)
{
    return n1 + n2;
}

int main()
{
    cout << addNumbers(2, 6) << endl;
}
```

1.1. Passing values by reference

If we need to define a reference parameter, we just add '&' as a prefix to the parameter name when we declare the function. For example:

```
int swapValues(int &value1, int &value2)
{
    int aux = value1;
    value1 = value2;
    value2 = aux;
}

int main()
{
    int n1 = 5, n2 = 10;
    swapValues(n1, n2);
    cout << "n1 = " << n1 << ", n2 = " << n2 << endl;
}
```

Regarding this, remember that arrays are *always* passed by reference, so you don't need to use this symbol with them.

2. Quicksort algorithm

Let's use functions to define a really useful, recursive function. The quicksort algorithm is a sorting algorithm which is faster than the "traditional" sorting algorithms that we learn when we start programming, such as "bubble" algorithm. It relies on dividing the array in two halves according to a given value called *pivot*. All the values lower than the pivot are placed at his left, and all the values greater than the pivot are placed at his right. Then, we make two recursive calls to quicksort algorithm to sort these two halves.

In C++ the code could be more or less like this:

```
void quicksort(int array[], int start, int end)
{
    int pivot, left, right;

    if (start < end)
    {
        pivot = array[(start + end) / 2];
        left = start;
        right = end;
        do
        {
            while (array[left] < pivot && left <= end)
                left++;
            while (array[right] > pivot && right >= start)
                right--;
            if (left <= right)
            {
                int aux = array[left];
                array[left] = array[right];
                array[right] = aux;
                left++;
                right--;
            }
        } while (left <= right);
        if (start < right)
            quicksort(array, start, right);
        if (left < end)
            quicksort(array, left, end);
    }
}
```

As you can see, at the beginning of the algorithm we take a reference called pivot. Typically this pivot is the number in the middle of the array, but assuming that the array does not need to be ordered, we could choose any value as the pivot.

Then, the `do..while` loop puts all the numbers greater than pivot to its right, and all the numbers lower than pivot to its left, and after that we call recursively the same function to sort both sides.

2.1. *An example: Los niños buenos*

To see how this algorithm works, and why it is so important in some challenges, let's face this one. Santa has to deliver all the gifts, and he wants to start by the best children. To do this, he has information about how good each child has been (a number between 1 and 100) and how heavy the gift for that child is (a number between 1 and 1000). So Santa will deliver first the gifts to the best children, and if two children are just as good, then he will deliver first the gifts which are lighter.

The input for each test case will consist of:

- A number indicating the number of children to manage
- For each children, a line containing his goodness level (from 1 to 100), and the weight of his gifts (from 1 to 1000)
- The program ends when the number of children to manage is 0

For each test case, we must output in which order will the gifts be delivered, according to the requirements specified before. We must print a **blank line** at the end of each case (except for the last one with 0 children). For instance, for this test case:

```
3

80 2

100 12

100 1
```

The output should be:

```
100 1

100 12

80 2
```

2.2. How to solve the problem

Basically, what we have to do for each test case is:

1. Define a structure to store both the children goodness level and the weight of their gifts. For instance:

```
struct child
{
    int goodnessLevel;
    int giftWeight;
};
```

2. Read the line containing the number of children, and create an array of `child` elements of that size
3. Store each data (goodness and weight) in the corresponding field of each position of the array
4. Sort the array so that better children are first, and if goodness level is the same, then sort it by weights, from lighter to heavier.
5. Explore the array and show the data for each children

This is the complete code for this challenge, applying these steps:

```
#include <iostream>

using namespace std;

struct child
```

```
{
    int goodnessLevel;
    int giftWeight;
};

void quicksort(child array[], int start, int end)
{
    child pivot;
    int left, right;

    if (start < end)
    {
        pivot = array[(start + end) / 2];
        left = start;
        right = end;
        do
        {
            while ((array[left].goodnessLevel > pivot.goodnessLevel ||
            array[left].goodnessLevel == pivot.goodnessLevel &&
            array[left].giftWeight < pivot.giftWeight)
            && left <= end)
                left++;
            while ((array[right].goodnessLevel < pivot.goodnessLevel ||
            array[right].goodnessLevel == pivot.goodnessLevel &&
            array[right].giftWeight > pivot.giftWeight)
            && right >= start)
                right--;
            if (left <= right)
            {
                child aux = array[left];
                array[left] = array[right];
                array[right] = aux;
                left++;
                right--;
            }
        } while (left <= right);
        if (start < right)
            quicksort(array, start, right);
        if (left < end)
            quicksort(array, left, end);
    }
}

int main()
{
    int numberOfChildren;
    do
    {
        cin >> numberOfChildren;
        if (numberOfChildren > 0)
        {
            child children[numberOfChildren];

            for (int i = 0; i < numberOfChildren; i++)
            {
                cin >> children[i].goodnessLevel;
                cin >> children[i].giftWeight;
            }

            quicksort(children, 0, numberOfChildren-1);
            for (int i = 0; i < numberOfChildren; i++)
            {
                cout << children[i].goodnessLevel << " "
                     << children[i].giftWeight << endl;
            }
            cout << endl;
        }
    } while (numberOfChildren > 0);
    return 0;
}
```