

This way, we have direct access to the input/output functions we'll use.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure left_stars is
begin

end left_stars;
```

We need a constant that will contain the character used to create the triangle, let's name it `Symbol`, and another constant that define the number of lines of our triangle, let's name it `N_Lines`.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure left_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

end left_stars;
```

We need two loops. One for printing each line, and the other for printing the correct number of stars according to the index of the line.

The first loop displays the lines of stars. Hence, we can use a For-Loop that goes from 1 to the number of lines, i.e. from 1 to `N_Lines`.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure left_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

    for I in 1..N_Lines loop

    end loop;

end left_stars;
```

Next, the second loop should display the stars. We will have `N_Lines` of stars. First line has $(N_Lines + 1 - Index_of_the_line)$ times the `Symbol` character. Where *Index_of_the_line* is the `I` (that goes from 1 to `N_Lines`) of the outer loop. During the first loop, it's equal to 1, then 2, 3, ..., 10.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure left_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

    for I in 1..N_Lines loop
        for J in 1..(N_Lines + 1 - I) loop

            end loop;
        end loop;

    end left_stars;
```

And what should the inner loop do? Only displaying **one** `Symbol` character on the screen and let the cursor on the same line to get the next `Symbol` character close to the previous one. Be careful, we want our lines of symbol characters displayed each time on a new line. That's why we add a command to go to a new line after the inner loop.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure left_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

    for I in 1..N_Lines loop
        for J in 1..(N_Lines + 1 - I) loop
            Put(Symbol);
        end loop;
        New_Line;
    end loop;

    end left_stars;
```

Now it is easy to adapt your procedure in order to align your triangle on the right. Start by renaming your procedure.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure right_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

    for I in 1..N_Lines loop
        for J in 1..(N_Lines + 1 - I) loop
            Put(Symbol);
        end loop;
        New_Line;
    end loop;

end right_stars;
```

The only stuff to add is some blank spaces before starting to display the stars. How many blank spaces per line? 0 for the first line, 1 for the second, ... (*Index_of_the_line* - 1) for the *Index_of_the_line*-st line of stars. Thus, create another For-Loop that goes from 1 to (I - 1) before displaying the stars.

```
with Ada.Text_IO; use Ada.Text_IO;

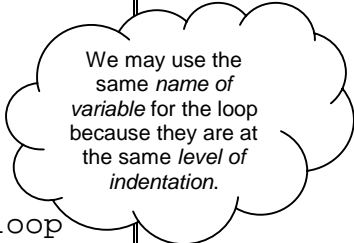
procedure right_stars is

    Symbol : constant character := '*';
    N_Lines : constant := 7;

begin

    for I in 1..N_Lines loop
        for J in 1..(I - 1) loop
            Put(' ');
        end loop;
        for J in 1..(N_Lines + 1 - I) loop
            Put(Symbol);
        end loop;
        New_Line;
    end loop;

end right_stars;
```



For the center alignment, you can easily determine the number of stars for the I -st line: $(I - 1) * 2 + 1$

The rest of the code is easy to find.

```
with Ada.Text_IO; use Ada.Text_IO;

procedure center_stars is

  Symbol : constant character := '*';
  N_Lines : constant := 4;

begin

  for I in 1..N_Lines loop
    for J in 1..(N_Lines - I) loop
      Put(' ');
    end loop;
    for J in 1..((I - 1) * 2 + 1) loop
      Put(Symbol);
    end loop;
    New_Line;
  end loop;

end center_stars;
```

Warning: This is not the same number of blank spaces as above.

You can now try to adapt the previous procedures to get these outputs...

Left	Right	Center
*	*	*****
**	**	*****
***	***	***
****	****	*
*****	*****	
*****	*****	
*****	*****	