Necklaces

Source code: `necklace.c/necklace.cpp/necklace.pas`
Time limit: 3 s
Memory limit: 128 MB

And Alice said: "I have the most beautiful necklace. From left to right, it consists of two red pearls, two green ones and one more red one." Beatrix was quick to answer: "Mine is even better. It looks almost like yours, but you would need to remove the two rightmost pearls and replace them by two blue ones." No sooner did she stop, when Caroline jumped in: "That is no match for my necklace. I have one more yellow pearl on the left." You probably will not be surprised when I tell you that Dominica also did not stay silent: "That is all boring. To get my necklace, you would have to take the Beatrice's one, remove the leftmost and the rightmost pearls, and add two black pearls to the left." And so it went on, until Zaida asked: "I got a bit confused; what was the color of the leftmost pearl of the Eugenie's necklace?" And silence answered her.

Task

Your task is to write a *library* (unit in Pascal) that is able to help with the conversations of this type. The interface of the library is described in what follows; you can find templates for the libraries in the directories `/mo/public/necklace/c`, `/mo/public/necklace/cpp`, and `/mo/public/necklace/pas` (the subdirectories for C and C++ also contain the header file `necklace.h`). There is no input or output.

Your library will work with a set of *necklaces*. A necklace is a sequence of integers between 0 and 1 000 000, ordered from left to right. Each necklace is identified by a nonnegative integer. Initially, only a necklace 0 that consists of an empty sequence is present.

In the beginning, the evaluation system will call exactly once the procedure **init**. Then, it will repeatedly call in an arbitrary order the functions **create** and **pearl**. In total, your library will be called at most 1 000 000 times in each test run.

The call to the procedure **create** corresponds to creating a new necklace. The number of the necklace is by one higher than the largest number of an existing necklace, i.e., the first call to **create** creates a necklace with number 1, the second one a necklace with number 2, etc. The new necklace is derived from the necklace number **from** by the operation specified by the parameters **operation**, **on_left** and **param**:

- If the parameter **operation** is the letter **R** (as "remove"), a single integer is removed from the end of the necklace. The parameter **param** is ignored in this case.
- If the parameter **operation** is the letter **A** (as "add"), the integer **param** is added to the end of the necklace.

If **on_left** is true (nonzero in C), then the operation is performed at the left end of the necklace, otherwise it is performed at the right end. The **from** parameter of the **create**

call will always be smaller than the number of the necklace created by the call, i.e., it will refer to an existing necklace. You can always assume that no call will attempt to remove a pearl from an empty necklace.

The function **pearl** should return the integer on the left end of the necklace **neck_id** if **on_left** is true (nonzero in C/C++) and the integer on the right end of the necklace **neck_id**, otherwise. The function **pearl** will be only called for non-empty necklaces that were created by the call of the function **create** before. This function does not alter the necklaces.

Description of interface

In C/C++:

```
extern void init (void);
extern void create (int from, char operation, int on_left,
        int param);
extern int pearl (int neck_id, int on_left);
```

In Pascal:

```
unit necklace;

interface

procedure init;
procedure create (from : longint; operation : char; on_left : boolean;
        param : longint);
function pearl (neck_id : longint; on_left : boolean) : longint;
```

Note that you are allowed to create additional procedures, functions, and global variables in the library you prepare.

Example

The following example shows one possible sequence of calls of the functions and the returned values:

In C/C++:

```
init ();
create (0, 'A', 1, 5);
create (1, 'A', 1, 3);
pearl (2, 0);  /* returns 5 */
create (2, 'R', 0, 0);
pearl (3, 0);  /* returns 3 */
pearl (2, 0);  /* returns 5 */
```

In Pascal:

```
init;
create (0, 'A', true, 5);
create (1, 'A', true, 3);
pearl (2, false);  { returns 5 }
create (2, 'R', false, 0);
pearl (3, false);  { returns 3 }
pearl (2, false);  { returns 5 }
```

Testing your library

In addition to the templates for your library, the subdirectories of `/mo/public/necklace` contain the files `neck_main.c`, `neck_main.cpp`, and `neck_main.pas` which are the main source files that can be compiled with your library and will access the library routines in the way shown in the example (the program will inform you whether the return values of the function `pearl` are the correct ones).

The script `compile` with the parameter `necklace` will compile your library with the default main file (`neck_main.c`, `neck_main.cpp` or `neck_main.pas`) and the resulting program `necklace` will allow you to test your library as described further[1]. If the program is run without parameters, it calls your library as described in the example and checks whether the return values of the function `pearl` are the correct ones (if not, a message is written to the standard error output). If the program is called with the parameter `-i`, it switches to the *interactive mode*. It reads requests from the standard input, passes them to the library and writes the responses to the standard output.

In the interactive mode, the program first calls the procedure `init` and after this procedure terminates, it outputs `Initiated`. Then, the following three types of requests are expected (each on a separate line):

- `create` *from operation on_left param*
  This request causes the program to call the procedure `create` with the parameters *from*, *operation*, *on_left* and *param*. After the procedure is finished, the program outputs `Done`.
- `pearl` *neck_id on_left*
  This request causes the program to call the function `pearl` with the parameters *neck_id* and *on_left*. After the function is finished, the program outputs `Returns` X where X is the number returned by the function.
- `quit`
  This request causes the program to terminate.

Note that the program in the interactive mode only provides you with responses of your library and it does not check whether the responses are correct.

[1]Note that you can run the script `compile` with a file name as a parameter which allows you to compile your library with your own main files.

The following shows the requests and responses of the program running in the interactive mode corresponding to the sequence of calls to the library from the example.

In C/C++:

```
Initiated
create 0 A 1 5
Done
create 1 A 1 3
Done
pearl 2 0
Returns 5
create 2 R 0 0
Done
pearl 3 0
Returns 3
pearl 2 0
Returns 5
quit
```

In Pascal:

```
Initiated
create 0 A true 5
Done
create 1 A true 3
Done
pearl 2 false
Returns 5
create 2 R false 0
Done
pearl 3 false
Returns 3
pearl 2 false
Returns 5
quit
```