The Return Value of main( )

This lesson explains the return value of main() and how we can specify the return value.

We have seen that `main()` is a function. Program execution starts with `main()` and branches off to other functions from there. The definition of `main()` that we have used so far has been the following:

```
void main() {
    // ...
}
```

According to that definition `main()` does not take any parameters and does not return a value. In reality, in most systems, every program necessarily returns a value to its environment when it ends, which is called an *exit status* or *return code*. Because of this, although it is possible to specify the return type of `main()` as `void`, it will actually return a value to the operating system or launch environment.

The return value of `main()`

Programs are always started by an entity in a particular environment. The entity that starts the program may be the shell where the user types the name of the program and presses the enter key, a development environment where the programmer clicks the [Run] button and so on.

In D and several other programming languages, the program communicates its exit status to its environment by the return value of `main()`.

The exact meaning of return codes depends on the application and the system. In almost all systems a return value of zero means a successful completion, while other values generally mean some type of failure. There are exceptions to this, though. For instance, in OpenVMS, even values indicate failure while odd values indicate success. Still, in most systems, the values in the range [0, 125] can be used safely as return codes, with values 1 to 125 having a meaning specific to that program.
For example, the common Unix program `ls`, which is used for listing contents of directories, returns 0 for success, 1 for minor errors and 2 for serious ones.

In many environments, the return value of the program that has been executed most recently in the terminal can be seen through the `$?` environment variable. For example, when we ask `ls` to list a file that does not exist, its nonzero return value can be observed with `$?` as seen below.

> **Note:** In the command line interactions below, the lines that start with `#` indicate the lines that the user types. If you want to try the same commands, you must enter the contents of those lines except for the `#` character. Also, the commands later in the lesson start a program named **deneme**; replace that name with the name of your test program. Additionally, although the following examples show interactions in a Linux terminal, they would be similar but not exactly the same in terminals of other operating systems.

```
# ls a_file_that_does_not_exist
ls: cannot access a_file_that_does_not_exist: No such file or directory
# echo $?
2     ← the return value of ls
```

`main()`'s return value

Some of the programs that we have written so far threw exceptions when they could not continue with their tasks. As we have seen, when an exception is thrown, the program ends with an `object.Exception` error message.

When that happens, even if `main()` has been defined as returning `void`, a nonzero status code is automatically returned to the program's environment.

Let's see this in action in the following simple program that terminates with an exception:

```
import std.stdio;

void main() {

    throw new Exception("There has been an error.");

}
```

Although the return type is specified as `void`, the return value is nonzero:

```
# ./deneme
object.Exception: There has been an error.
...
# echo $?
1
```

Similarly, `void main()` functions that terminate successfully also automatically return zero as their return values. Let's see this with the following program that terminates successfully:

```
import std.stdio;

void main() {
    writeln("Done!");
}
```

Function with void return type

The program returns zero:

```
# ./deneme
Done!
# echo $?
0
```

Specifying the return value

To choose a specific return code that should be returned by main, we specify it as a return value from `main()`. This process is done in the same way as any other function. The return type must be specified as `int` and the value must be returned by the return statement:

```
import std.stdio;

int main() {
    int number;
    write("Please enter a number between 3 and 6: ");
    readf(" %s", &number);

    if ((number < 3) || (number > 6)) {
        stderr.writefln("ERROR: %s is not valid!", number);
        return 111;
    }

    writefln("Thank you for %s.", number);

    return 0;
}
```

Return value specified

When the entered number is within the valid range, the return value of the program is zero:

```
# echo $?
0
```

When the number is outside of the valid range, the return value of the program is 111:

```
# ./deneme
Please enter a number between 3 and 6: 10
ERROR: 10 is not valid!
# echo $?
111
```

The value of 111 in the above program is chosen arbitrarily; normally 1 is suitable as the failure code.

Standard error stream `stderr`

The program above uses one of the standard streams namely `stderr`. It is used for writing error messages:

- `stdin`: standard input stream
- `stdout`: standard output stream
- `stderr`: standard error stream

When a program is started in a terminal, normally the messages that are written to `stdout` and `stderr` both appear on the terminal window. When needed, it is possible to redirect these outputs individually, but we will not delve into its details.

In the next lesson, we will learn about the parameters of `main()`.