maXbox

Blix the Programmer

An Introduction to Programming

1.1 We program time

So you are eager to learn programming, ok let's code a time machine!

Imagine you are a programmer like Blix above. Our goal is to make him look cute nerdy, he looks very busy working with his dual monitor setup and his pile of books. I hope you guys love The Programmer as our guide as much as we do.
You've always wanted to learn how to build software (or just whip up an occasional script) but never knew where to start. Today you do that first step.

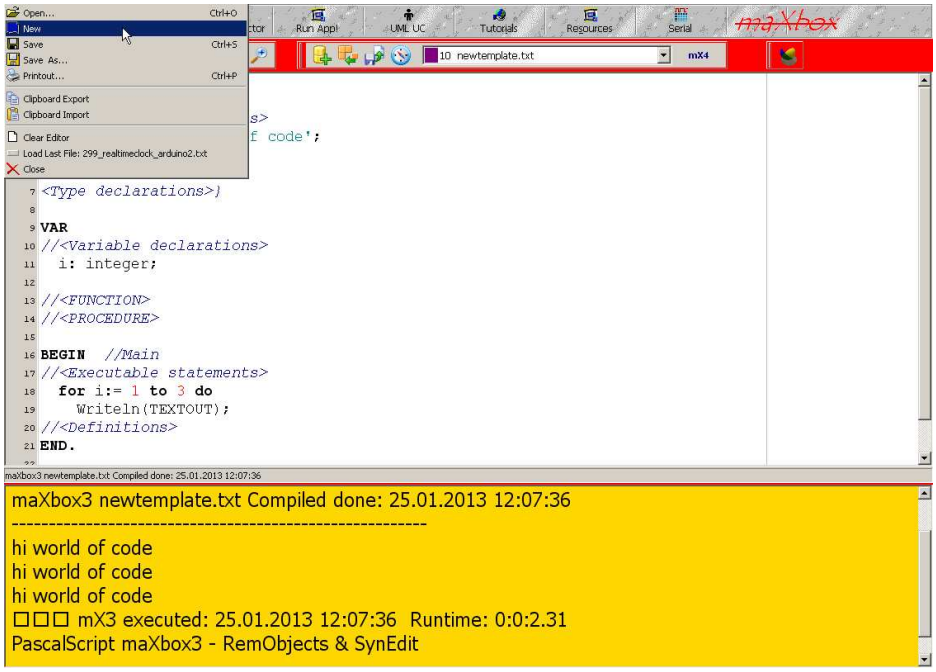
1.2 First we need a tool

A tool is needed to type some code and run it. What you think is coding? Coding is not only writing or typing. After all we don't write software, we build it!
This reminds me of a true "Dilbert moment" a few years ago, when my (obviously non-technical) boss commented that he never understood why it took months to develop software. "After all", he said, "it's just typing." ;-(
Luckily, the web is full of free resources that can turn you into a programmer.
Now we download such a free resource, called maXbox:
http://sourceforge.net/projects/maxbox/files/

The tool you will use is split up into the toolbar at the top, the editor or code part in middle and the output window (console) at the bottom with an interface part on the right. Change that in the menu `/View` at our own style.

☝ In maXbox you will execute just the script, all libraries and units are already built in.

Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default demo program. Test it with F9 / F2 or press **Compile** on the toolbar and you should hear a sound and a browser will open. So far so good now we'll prepare our first program, nowadays called an app:



Click on `File/New` and a template is loaded. You can run this template with `Compile` and the app writes three times "hi world of code" in the console at the bottom of the box.
Keep in mind we want to program a clock in several steps. Its good practise to name and store our script file with `Save As…`, for example to

`myfirstclock.txt`

If you want to look at the whole script you can find the file at:
http://www.softwareschule.ch/examples/341_blix_clock.txt

Now let's take a look at the code of this first part project. Our first line is

```
01 Program myClock;
```

We name it, means the program's name is above. Don't confuse it with the filename stored on your disk `myfirstclock.txt,` the program name is just an internal name defined.

☞ This example requires also objects, but we won't go deeper into object oriented programming, we just use objects. At the end of this doc you'll find links to maXbox tutorials.
Next we learn how a **constant** works. Constants are fixed numeric or character values represented by a name. In line 4 we set a name and the content of a constant:

```
02 Const
03//<Constant declarations>
04  TEXTOUT = ' hi world of code ';
```

A constant is called a const[1] because you can't change it during the program.
The next lines of the program declare a **variable** `i` of type integer. In line 11 we first declare name and type to use. This variable has a numeric type.

☞ A type is essentially a name for a kind of data. When you declare a variable you must specify its type, which determines the range and storage of values the variable can hold and the operations that can be performed on it.

```
09 Var
10//<Variable declarations>
11  i: integer;
```

You see in this primer, we don't deal with own types, function or procedure. The next lines are simply comments in your code and are ignored by the **compiler** (the part that translates your code into instructions a computer can understand before executing it).

```
13 //<FUNCTION>
14 //<PROCEDURE>
```

Line 16 ff is much of interest. A program must have a main routine between **begin** and **end**. The main routine is run once and once only at the start of the program (after you compiled). Here will you place general instructions and the main control of the app.

```
//Main routine
16 Begin
17 //<Executable statements>
18  for i:= 1 to 3 do  //iteration
19    Writeln(TEXTOUT);
20 //<Definitions>
21 End.
```

1.3 Time will come

So we are ready to change our first line. The easiest way to try your hand at coding for your Win, Linux or Mac desktop is to start with a script or macro program and change something.
Our plan is to change the output and add time information in a loop. Right, a loop is similar to iteration. The "for statement" in line 18 implements an iterative loop. After each iteration of the loop, the counter variable `i` is incremented. Therefore, `i` is the loop counter.
In this line we add the actual time:

```
19    Writeln(TEXTOUT + TimeToStr(Time));
```

Time is our first **function** called, you guess it, `Time`. We say we call a function. Because functions return a value, we get the time and send it to the output with `Writeln()`.
But first we have to convert it to a string with another function, `TimeToStr`. Compile it!
But wait a second, as you can see we don't see the difference, because the loop is too fast. What about to delay the loop and get the seconds in between as clockwork. No problem at all, we add a new line 20 with the procedure `Delay`:

```
19    Writeln(TEXTOUT + TimeToStr(Time));
20    Delay(1000); //milliseconds
```

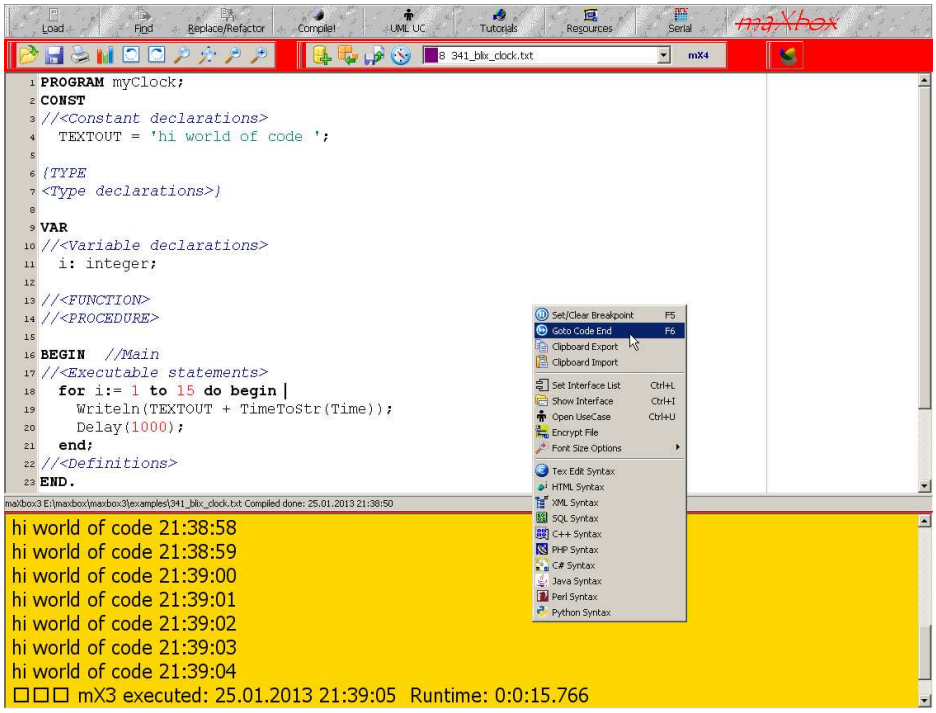
[1] You can only change the declaration for the next execution

Something is still missing in our design. We have to extend the scope of the loop with a building block; means the loop runs between begin and end. Also each begin must have and end ;-).

```
18  for i:= 1 to 3 do begin
19    Writeln(TEXTOUT) + TimeToStr(Time));
20    Delay(1000)
21  end;
```

Now it's up to you: produce a clock output that runs for 15 seconds.
Right you simply change the loop counter from 3 to 15:

```
18  for i:= 1 to 15 do begin
```



As a control, the runtime output (in yellow window) is about 15 seconds more or less, depending on the CPU performance at the moment. A one hour clock has to be set to 3600!

1.4 Time forever

Programmers never die, they just GOSUB without return. You may also know, in the beginning there was nothing which exploded ;). So let the jokes aside. Our last step is an eternal clock that goes back in the time like I said before: we code a time machine.
When we travel in past, present or future we change the logic from digital to trigital ;-).
So this piece of code can then be translated and run on various platforms and frameworks as well. There are several different kinds of software development you can do for various platforms, from the web to a desktop to your smart phone to a command line.
So what's the solution to run this time forever? Answer: a do forever loop or at least one hour.
With the call of another function we set the time one hour back in every second (sign: `-i`).

```
18  for i:= 1 to 3600 do begin
19    Writeln(TEXTOUT + TimeToStr(AddHours(Time,-i)));
20    Delay(1000)
21  end;
```

Be careful, the loop counter 3600 will last long, so change it step by step on your own experience. The clock goes one hour back into the past every second, but the seconds tick forward like a normal clock. In the film “back to the future” they call it the flux-comparator.

 If you want to stop or **break a loop**, just override the loop counter in line 18 and recompile (F9) it during the execution!
When you call the function `AddHours` that takes two argument and returns another time, then we say the function call `TimeToStr(AddHours(Time,-i))` is nested. A nested call contains other functions within a statement. Let me explain:

First we call the time function, the result we pass to the `AddHours` function its result is passed to the `TimeToStr` function and the whole we pass again to `Writeln`!

When we want to travel back to the past, maybe in the year of 1759, further information is missing. Right, there is no date. Easier done than said ;-).

```
18  for i:= 1 to round(Power(2,4)) do begin
19    Writeln(TEXTOUT + DateTimeToStr(AddHours(Now,-i)));
```

Did you see the difference? We replaced the `Time` function with the `Now` function and the string converter to `DateTimetoStr`. And with `Power` big numbers are possible (Power is like 2^4). Now we’re ready to go back to middle age.

  How can you accelerate your time machine? One hour back per second takes to much time, simply the loop must step faster.
You got it; we change the parameter of the delay procedure:

```
20    Delay(10);
```

Remember: The clock rate is still the same, but our time machine can go faster to the past. The time is flushing by with this speed; the calculation of how many lines we get is also interesting:

Suppose we have `Power(2,12)` as the for counter limit, how many lines we get?
Answer: 2^12 = 4096.

And the next by `Power(2,30)` could be also of interest, but its huge and your app runs long!:
Can you imagine where in the past we are landing, middle age or stone age or maybe far out of our history line?
The calculation is simple: (2^30/24)/365 is rounded to 122573 years. This would be Stone Age!
How can you do that in maXbox: 2^30 / (24*365) is another solution.

```
  Writeln(intToStr(round(Power(2,30)/24/365)));
```

We come closer to the end and had to refactor just one thing. Our last release looks like this:

```
  for i:= 1 to round(Power(2,N)) do begin
    Writeln(IntToStr(i)+TEXTOUT + DateTimeToStr(AddHours(Now,-i)));
    Delay(SN);   //speed of time machine
  end;
```

As you can see I introduced two parameters to be more flexible, the counter limit `N` and the speed of time machine `SN`. Means also 2 more variables to add in your code:

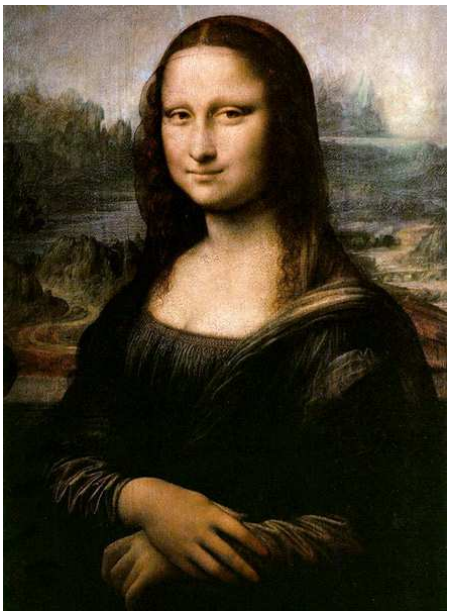
```
11  i, N, SN: integer;

18  N:= 4; SN:= 1000;  //Assign a value to a variable
```

So far we have learned something about functions in a loop, comments and the difference between a constant and a variable. Now its time to reflect over those used functions:

Function	Explanation and Purpose…
Writeln()	Writes one or more lines as string to an output.
IntToStr()	Converts an integer to a string.
DateTimeToStr()	Converts a variable of type TDateTime to a string.
AddHours()	Change the hours of a TDateTime function.
Now	Returns the current date and time.
Power()	Power raises Base to power specified by Exponent.
Round()	Returns a value of a real rounded to the nearest whole number.
Delay() or Sleep()	Pauses execution for a specified number of microseconds.

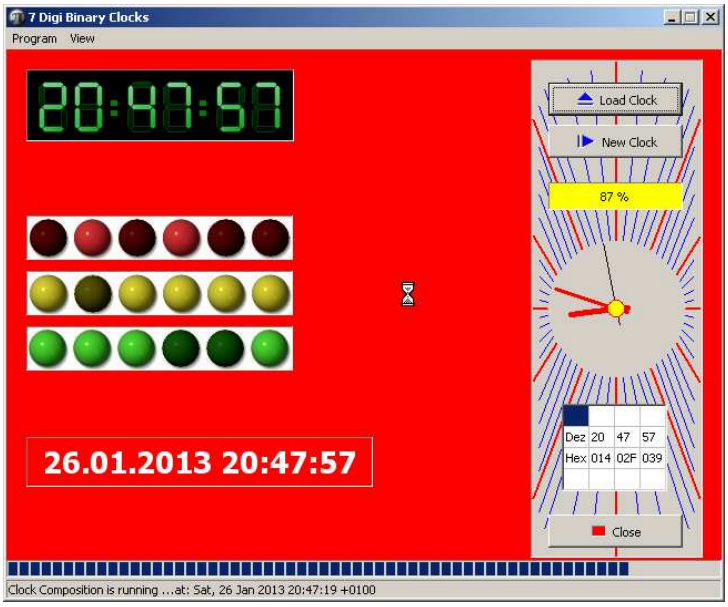
There are plenty more functions, which can be found in `Help/All Functions List`.
This interface information of RTL functions is contained in various unit files that are a standard part of C++, Delphi or Java. This collection of units is referred to as the RTL (run time library). The RTL contains a very large number of functions and procedures for you to use.



Time Travelling from Middle Age to the ancient Greeks till Stone Age ;-)

♏ Conclusion: A common pitfall for beginners is getting stuck figuring out which programming language is best to learn first.

There are a lot of opinions out there, but there's no one “best” language. Here's the thing to consider: In the end, language doesn't matter THAT much. Understanding data and control structures, algorithms, metrics, modelling and design patterns does matter very much. Every language, even a simple scripting language, will have elements that you'll use in other languages as well and will help your understanding. maXbox is build on Object Pascal and is similar to Java or C++ a well known and mighty language.



Script: Examples/336_digiclock2.txt

V1.2, Feedback @
max@kleiner.com

Literature:
Kleiner et al., Patterns konkret, 2003, Software & Support
Proof Reading by Dr. Silvia Rothen
Links of maXbox Tutorials:

http://www.softwareschule.ch/maxbox.htm

http://sourceforge.net/projects/maxbox

http://sourceforge.net/apps/mediawiki/maxbox/

http://en.wikipedia.org/wiki/Time_travel