

maXbox



maXbox Starter

Start with Programming

1.1 First Step

To start programming isn't that difficult. You may know, in the very beginning was nothing which exploded ;). The Starter has been written presuming you don't have any prior knowledge of programming or software engineering.

Coding is sort of a discipline between science and art. In no time we will get deep into maXbox or ObjectPascal programming (OP). It's just a primer to show the first steps in writing a program. Another lesson goes deeper with a second step and shows objects, classes and events.

All you need to know is that in this program, we have a procedure and a function, the procedure is called `TestPrimNumbers` and it's purpose is to check numbers if they are prime or not.

1.2 Get the Code

When you work with maXbox you will have a sight very similar to the image below. Using Windows or Linux will present some differences but the box is pretty much the same no matter what operating system you are using. The tool is split up into the toolbar across the top, the editor or code part in the centre and the output window at the bottom.

 The 4 buttons (speed buttons) in the toolbar provide convenient access to the most commonly used functions within the menus "files" and "program".

Before this starter code will work you will need to download maXbox from the website. This can be done from <http://www.softwareschule.ch/maxbox.htm> (you'll find the download `maxbox2.zip` top left on the site). Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox2.exe` the box opens a default program. Test it with F9 or press **Compile** and you should hear a sound. So far so good now we'll open our example.

```
50_program_starter.txt
```

The **Load** button will present you in `/examples` with a list of programs stored within your directory as well as a list of `/exercises` with defective Pascal code to be fixed for training purposes. Alternatively you can download the file from: http://www.softwareschule.ch/download/50_program_starter.txt Use the `Save Page as...` function of your browser¹ and load it from `examples` (or wherever you stored it). Now let's take a look at the code of this project. Our first line is

```
1 program Primtester_3;
```

We have to name the game, means the program's name is `Primtester_3`.

¹ Or copy & paste

☞ Most of the functions we use like round() or SQRT() are implicit in a library (or unit). A library is a collection of code, which you can include in your program. By storing your commonly used code in a library, you can reuse code for many times in different projects and also hide difficult sections of code from the developer. Once a unit is tested it's stable to use.

```

1 program Printester_3;
2
3 { simple printester between range for performance- and filehandling
4 has a function and procedure, consts, globals and locals, locs = 59
5 shows sequence, selection and iteration (units are implicit)
6 to teach programming for beginners in maXbox!}
7
8 const
9   FROM_RANGE = 1000;
10  TO_RANGE = 5000;
11
12 //globals
13 var
14   myList: TStringList; //is an object from class TStringList!
15   beforeTime, afterTime: string;
16
17 function checkPrim(acter: integer): boolean;
18 var //locals
19   j: integer;
20   isprim: boolean;
21 begin
22   isprim:= true;
23   for j:= 2 to round(SQRT(acter)) do
24     if ((acter mod j) = 0) then begin
25       isprim:= false;
26       break

```

maXbox D:\kleiner2005\TestApp\maXbox2\50_program_starter.txt Compiled done: 29.12.2009 21:23:29

```

Compiling maXbox 59 lines
Code lines in window: 26
D:\kleiner2005\TestApp\maXbox2\50_program_starter.txt file stored
D:\kleiner2005\TestApp\maXbox2\50_program_starter.txt in .ini stored
maXbox D:\kleiner2005\TestApp\maXbox2\50_program_starter.txt Compiled done: 29.12.2009 21:23:29
start: 29.12.2009 21:23:29 from: 1000
stop: 29.12.2009 21:23:30 to: 5000
ruNMax succesfully executed 29.12.2009 21:23:30

```

1: The maXbox Tool

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).

```

03 {simple printester between range for performance- and filehandling
04 has a function and procedure, consts, globals and locals, locs = 59
05 shows sequence, selection and iteration (units are implicit)
06 to teach programming for beginners in maXbox!}

```

Any text entered behind a {or//or(* command will be ignored by the compiler and is simply there for you, or anyone else that reads or extends your code. Comments are essential in your code to help you understand what's going on and how your code works or should work. The tool will automatically turn the colour of any commented text to blue.

Next we learn how a constant works. Constants are fixed numeric or character values represented by a name. Constants can't be changed² while a program runs. A section starts with the word const:

```

08 Const
09   FROM_RANGE = 1000; //start with even numbers
10   TO_RANGE = 5000;

```

The lines 13 to 15 of the program contain the declaration of 3 variables myList, beforeTime and afterTime. Line 14 is declaring a more complex variable, namely an object of type TStringList (as its name suggests a list of strings). Line 15 is declaring two strings and each variable has a type.

² You can only change the declaration

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

```
13 var
14  mylist: TStringList; //is an object of class TStringList!
15  beforeTime, afterTime: string;
```

A variable is a place to store data. In this case you are setting up a variable of type `string` and `TStringList`. Imagine a variable as a small box (in `maxbox`;) where you can keep things. A variable is called a variable because its value can change during the programs execution. But watch out not every name can hold a variable because you can't use any of OP's keywords like `set`, `while`, `case`, `if` then etc as variable names.

 **Keywords** are constants, variables, type and procedure names that are defined as part of the OP language like `var` or `string` above. All keywords within the program will appear in **bold**.



So far we have learned something about library, comments and the difference between a constant and a variable. Now it's time to run your program at first with F9 (if you haven't done yet). The program generates a file called `primetest8.txt` containing all prime numbers in the range 1000 to 5000. But the output is missing in the window on the bottom and we have to change the code. We just activate the code line 56, which is currently a comment:

```
//memo2.lines.loadFromFile('primetest8.txt')
```



After we have removed the `//` in line 56 and pressed F9 (or Compile) we can see the numbers in the output window. Just scroll up to the top of the output window

```
memo2.lines.loadFromFile('primetest8.txt')
```



```
maxbox D:\Heiner2005\TestApp\maxbox2\50_program_starter.txt Compiled done: 29.12.2009 21:48:52
1: 1009
2: 1013
3: 1019
4: 1021
5: 1031
6: 1033
7: 1039
8: 1049
```

2: The Output Window

The **Compile** button is also used to check that your code is correct, by verifying the syntax before the program starts. When you run this code you will see that we catch 501 prime numbers in a set count up from 1009 to 4999. The time consuming depends on your PC and the goal is to get much larger primes. The search for ever larger primes has generated interest outside mathematical circles but there is no known formula yielding all primes!



So let's jump back to line 17. This line is our first **function** called `checkPrim()`.³



Because functions return a value, they can be part of an expression. For example, if you define the function called `checkPrim` that takes one integer argument and returns a Boolean, then the function call `checkPrim(akti)` is a Boolean expression. An expression is a syntactical block that can be the right side of an assignment within a statement and delivers a value; means an expression is resolved into a simple value at runtime. (Also each **begin** must have and **end** ;).

```
17 function checkPrim(akti: integer): boolean;
18 var //locals
19  j: integer;
```

³ We later change the name to `CheckPrime`

```

20  isprim: boolean;
21  begin
22  isprim:= true;
23  for j:= 2 to round(SQRT(acti)) do
24    if ((acti mod j) = 0) then begin
25      isprim:= false
26      break
27    end; //if
28  result:= isprim;
29  end;

```

We have two local variables and `isprim` is initialised with `true` (as long the loop proves false). What's the meaning of initialising? It has the purpose to set them to a known value, to `true` in this case. Then a `for` statement in line 23 implements an iterative loop. After each iteration of the loop, the counter is incremented. Consequently, `j` is called the loop counter.

```

23  for j:= 2 to round(SQRT(acti)) do

```

Then we have our first `if` Statement:

```

24  if ((acti mod j) = 0) then begin

```

If the condition `(acti mod j)` evaluates to 0, then the number can't be a prime and `break` jumps out of the loop. The `mod` operator returns the remainder obtained by dividing with `j`. If the condition is false, means not 0, then we found a prime and the function returns `true` in a result value (line 28).



The `else` keyword of the `if` statement is optional;

Basically a function is a block of code assembled into one convenient block. We create our own function to carry out a whole series of complicated lines of code, we could run that code as many times as we like simply by calling the function name instead of writing out the code again.



Next we go to line 32 and define our first **procedure**.

```

procedure TestPrimNumbers(Vfrom_range, Vto_range: integer);

```

A procedure (call) consists of the name of a procedure (with or without qualifiers), followed by a parameter list (if required). Functions return a value where procedures must not! Most functions and procedures require parameters of specific types. As we already know a type is just a name for a kind of data and parameters must have the types of a language or self defined ones.



The parameters `Vfrom_range`, `Vto_range` of type `integer` declared in the procedure do have automatic scope and only live during the execution of the procedure or function. Next we need to initialise a variable to be the start number of the loop. So we set up variable `count` and assign it to zero:

```

35  count:= 0; //init

```

Then an object variable called `mylist` is constructed⁴. For our example, the `TestPrimNumbers` procedure creates a `TStringList` object and fills it with the found prime numbers. We then set a condition to decide how many times the code in the loop will execute.

```

37  for acti:= Vfrom_range to Vto_range do

```

In this case the code will loop from `acti` is greater than (\geq) 1000 and smaller than (\leq) 5000. The code within a `for` loop normally execute once no matter what the condition is set to. Now to line 39; here we are telling the compiler that our procedure is calling `checkPrim()`, that it passes a number and returns `true` if the number is prime and adds the number to the list `mylist`.

⁴ More on objects in a Second Step

Then we have another if statement.

 You remember that function calls are expressions.

Now remember the `checkPrim` takes one integer value as a parameter and returns a boolean value in turn. Therefore we can call the function directly as the conditional expression:

```
39 if checkPrim(acti) then begin
```

If the expression is true, then the statement is executed, otherwise it's not. By the way, do you know the caller? It's in our case the procedure which calls a function!

  If we call the function without the parameter (`acti`) then we get: Invalid number of parameters, try it (you pass no parameters to it (empty parenthesis)). Therefore each function or procedure has its own **signature**; that is the routine's name, parameters, and return type (for functions).

```
32 procedure TestPrimNumbers(Vfrom_range, Vto_range: integer);
33 var acti, count: integer;
34 begin
35   count:= 0; //init
36   mylist:= TStringList.create;
37   for acti:= Vfrom_range to Vto_range do begin
38     inc(acti) //only odd numbers check
39     if checkPrim(acti) then begin
40       inc(count)
41       mylist.add(intToStr(count) + ': ' + intToStr(acti))
42     end //if
43   end //for
44 end;
```

The `TestPrimNumbers` procedure receives as parameters two integer values (32 bit number), which will be our start and end numbers between 1000 and 5000 for the iteration. Of course you can change them. We have chosen integer for this usage as it's a common used type. If you change it to a byte type (exactly 8 bits in length) you could only pass values from 0 to 255 to the procedure!

 Try to change the consts declaration for example from 10000 to 20000 in line 9 and 10! How many primes you get and how long it'll last?

 A function is called with actual arguments placed in the same sequence as their matching formal parameters. For example, `checkPrim(acti)` is the actual parameter as the caller (or sender) and function `checkPrim(acti: integer): boolean;` is the formal parameter called `acti` as the receiver. Therefore a function can be used in many calls.

 Try it with the procedure `TestPrimNumbers()`: Where is the caller (actual arguments) and in which line is the receiver?

 Yes the caller is `TestPrimNumbers(FROM_RANGE, TO_RANGE)` in line 50 of the main routine and the receiver is line 32 with the signature of the procedure: `TestPrimNumbers(Vfrom_range, Vto_range: integer);`

This is what we call an **interface** in the way software works; the interface declaration of a procedure or function includes only the routine's signature.

 Because `maXbox` can't have a direct access to the C Lib of the API⁵ all functions are wrapped in the interface with the help of the VCL (Visual Component Library) or the CLX (Kylix with CLX runs on Linux). Also the main program is just a wrapper around a main win loop.

⁵ Application Programming Interface

1.3 The Main Routine (PrimeTime)

An OP 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) and is where you will do the general instructions and the main control of the program.

The most important elements that I will present here are: `TestPrimNumbers(FROM_RANGE, TO_RANGE)`; as you already know as the caller and `mylist.saveToFile('primetest8.txt')`; (as the generator of the file).

The Main starts in line 47. In line 49 we are telling the compiler to call a function named `Now()`; It simply returns the current system time and date. We pass no parameters to it (empty parenthesis or no parenthesis).

```
49 beforeTime:= Now();
```

The `beforeTime` variable will be used to store the time difference we are going to display on the output window below. Normally `Now` does have the type like `time` or `Now2` from `TDateTime` but as a simplification it's a string. If you change `Now()` to `Time()` you get this:

PSXCompiler: [Error] D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt(51:19): Type mismatch



```
maxbox D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt Compiled done: 31.12.2009 12:36:53
D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt file stored
D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt in .ini stored
maxbox D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt Compiled done: 31.12.2009 12:36:53
start: 31.12.2009 12:36:53 from: 1000
stop: 31.12.2009 12:36:53 to: 5000

ruNMax succesfully executed31.12.2009 12:36:53
PascalScript maxXbox; (c) 2005 by RemObjects & SynEdit
```

3: Performance in Output

With the procedure `writeln()` in line 52 you write data to the output window. Next I have to say something about file handling. First you have to define a file name for the data to save and load. The actual file name in line 55 is: "primetest8.txt". You can edit this file name directly and you can set an absolute path or a relative one in the procedure⁶ `saveToFile()`:

```
mylist.saveToFile('C:\myPrimeName.txt')
```

But as a developer you are responsible for every aspect of the data access. In short, before any data access can occur, data must be write (save) and read (load) from the same file and stored in memory. So make sure the method `loadFromFile()` has the same file name too:

```
memo2.lines.loadFromFile('C:\myPrimeName.txt')
```



How can you improve the situation that only one file name exists? Yes, you define a constant:

Const

```
PRIMEFILE = 'C:\myPrimeName.txt';
```

And then you alter the two procedures (or methods):

```
mylist.saveToFile(PRIMEFILE)
memo2.lines.loadFromFile(PRIMEFILE)
```



we will do that because these two methods are the corner stone of the Import / Export possibility in this program. By the way: It's a convention to write a constant name in CAPITAL LETTERS.

Next in line 56, if a data file with your const or whatever defined name already exists, it is opened and the previously stored data is read from it. In our case the data is loaded to a **component** called `memo2`, which is a part of the output window (`memo2` of class `TMemo` which is serving both as a memory structure for holding the data and as a visual control for navigating and editing the data).

⁶ We call it a method because it belongs to an object

```

//main program
begin
  //time performance
  beforeTime:= Now;
  TestPrimNumbers(FROM_RANGE, TO_RANGE);
  afterTime:= Now;
  writeln('start: ' + beforeTime + ' from: '+intToStr(FROM_RANGE))
  writeln('stop: ' + afterTime + ' to: '+intToStr(TO_RANGE))
  myList.add(memo2.text)
  myList.saveToFile('primetest8.txt')
  memo2.lines.loadFromFile('primetest8.txt')
  myList.Free;
end.

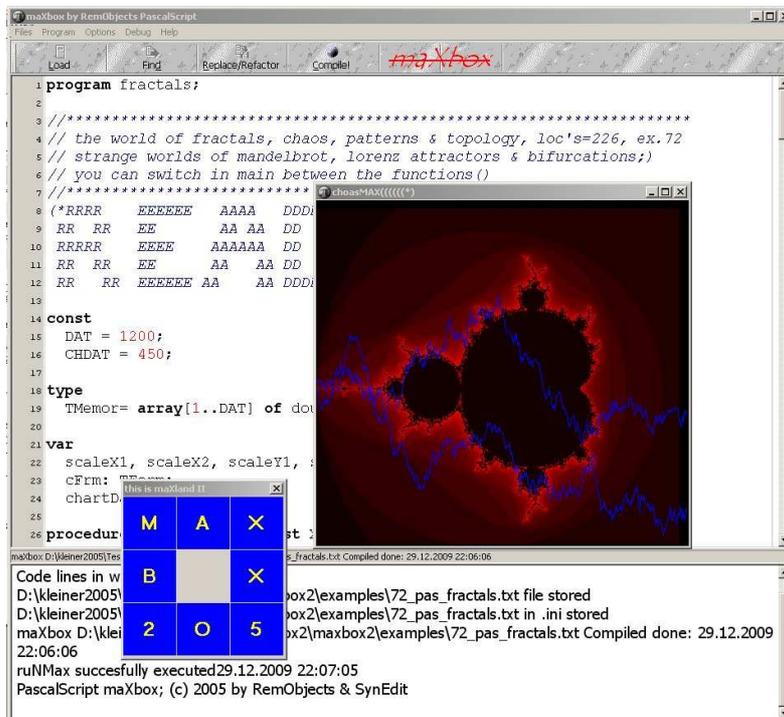
```



The basic concept here is that we have a main routine that executes 4001 (1000 to 5000) times a check function. In that routine we simply take a look at the content of each number (whether prime or not) and display it on output and write it down to a file with the help of a class called TStringList. The data structure is sort of a direct file access. You can always use a memo or a grid to display a result, but if you don't need the overhead of a data aware component or you don't want a database in an embedded system, direct file access with a stringlist can be a choice.

maXbox

One note about this primer; there will be better solutions (executeⁱ a thread or a thiefⁱⁱ) and we still work on it for educational reasonsⁱⁱⁱ, so I'm curious about comments, since I intend to publish improvements of this code in a basic chat on sourceforge.net depending on your feedback ;)



1.3.1 Code Warrior

We can state that for educational reasons its better to begin programming in a procedural way followed by OOP. In procedural programming (which predates OOP), you create constants, variables, functions, and statements. Those have to be learned before OOP. I would say that even OOP is

tougher to learn and much harder to master than procedural coding, you better learn it after having achieved procedural thinking. Truth is that almost anything which can be done procedurally can also be done using objects and vice versa.

Let's go back on track and do a last exercise. The final menu is the **Help** menu where you can find more information on the language or links to the reference pages of the maXbox site and other useful pages. A Code Warrior also knows a lot about debugging. As I promised we won't go any deeper, but as a simple preview try the code once with the magic "step into" F7. It really shows you step by step how the code works!

 With a right mouse click on the pop up menu in the editor you can set so called breakpoints and study the source from which point you want or use different steps to execute and stop from. As a last exercise you may have noticed that the name of prim is different to prime. This call for a change (we talk about refactoring ;)) of names:



Change the name from prim to prime (e.g. from `TestPrimNumbers` to `TestPrimeNumbers()` and from `checkPrim` to `checkPrime()`)



Think about the dependencies of names and rate of changes if the compiler throws an error like "Unknown identifier" but I'm sure you got it.



This is a so called Call Tree: [1]
50_program_starter.main block (50i)
50_program_starter.TestPrimNumbers (32i)
50_program_starter.checkPrim (17i)



That's all folks, may the source be with you for the first step and I hope to see you in second step.

Feedback @

max@kleiner.com

Many thanks to the Reviewer Team (Dr. Silvia Rothen, Beat Strähl, Beat Fankhauser, Dr. Andreas Fuchs)

Literature:

Kleiner et al., Patterns konkret, 2003, Software & Support

Links of maXbox and Prime Numbers:

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

<http://www.softwareschule.ch/>

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

<http://sourceforge.net/projects/maxbox>

The Document itself:

http://www.softwareschule.ch/download/maxbox_starter.pdf

1.4 Appendix

1.4.1 Code in OP

```
program Primetester_3_Plus; //optimized
//based on: http://www.softwareschule.ch/download/50_program_starter.pdf
{ simple primetester between range for performance- and filehandling
has a function and procedure, consts, globals and locals
shows sequence, selection and iteration (units are implicit)}
const
  FROM_RANGE = 1000; //start with even number
  TO_RANGE = 5000;
  PRIMEFILE = 'primetest9.txt';

//globals
var
  mylist: TStringList; //is an object of class TStringList!
  beforeTime, afterTime: string;

function checkPrime(acti: integer): boolean;
var //locals
  j: integer;
  isprim: boolean;
  ..myIdx: integer
begin
  isprim:= true;
  myIdx:= round(SQRT(acti))
  for j:= 2 to myIdx do
    if ((acti mod j) = 0) then begin
      isprim:= false
      break
    end; //if
  result:= isprim;
end;

procedure TestPrimeNumbers(Vfrom_range, Vto_range: integer; vlist: TStringList);
var acti, count: integer;
begin
  count:= 0; //init
  for acti:= Vfrom_range to Vto_range - 1 do begin
    inc(acti)
    if checkPrime(acti) then begin
      inc(count)
      vlist.add(intToStr(count) + ': ' + intToStr(acti))
    end //if
  end //for
end;

//main program
```

```

begin
    //time performance
    beforeTime:= Now;
    mylist:= TStringList.create;
        TestPrimeNumbers(FROM_RANGE, TO_RANGE, mylist);
    afterTime:= Now;
    writeln('start: ' + beforeTime + ' from: '+intToStr(FROM_RANGE))
    writeln('stop: ' + afterTime + ' to: '+intToStr(TO_RANGE))
    mylist.add(memo2.text)
    mylist.saveToFile(PRIMEFILE)
    memo2.lines.loadFromFile(PRIMEFILE)
    mylist.Free;
    //orthogonal and idempotent!
end.

```

1.4.2 Code in C++

```

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <string>
#include <list>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <sstream>
#include <time.h>
#include <math.h>

using namespace std;

#define FROM_RANGE 1000
#define TO_RANGE 5000

static bool checkPrime(int aPotentialPrimeNumber)
{
    bool isprim = true;

    for (int j = 2; j<=round(sqrt(aPotentialPrimeNumber)); j++)
        if (aPotentialPrimeNumber % j == 0)
            {
                isprim = false;
                break;
            }
    return isprim;
}

static void testPrimeNumbers(std::list<string> &primeNumbersFound, int fromRange, int toRange)

```

```

{
int primesFound = 0;

for (int potentialPrimeNumber = fromRange; potentialPrimeNumber<=toRange; potentialPrimeNumber++)
if (checkPrime(potentialPrimeNumber))
{
primesFound += 1;

stringstream str;
str << primesFound << ": " << potentialPrimeNumber << endl;

primeNumbersFound.push_back(str.str());
}
}

```

```

int main(int argc, char *argv[])
{
time_t t;

time(&t);
string beforeTime = ctime(&t);
cout << beforeTime;

list<string> primeNumbers;

testPrimeNumbers(primeNumbers, FROM_RANGE, TO_RANGE);

time(&t);
string afterTime = ctime(&t);
cout << afterTime;

ofstream f("primetest8.txt");
for (list<string>::iterator it = primeNumbers.begin(); it != primeNumbers.end(); it++)
f << *it;

return EXIT_SUCCESS;
}

```

```

-----
-----
//Second Step Notices with: if then else, call by reference, pointer, type, object,
class, try except and events available on April 2010 as maXbox_starter2.pdf

```

```

// if then else example
begin
isprim:= true;
for j:= 2 to round(SQRT(acti)) do begin
if ((acti mod j) = 0) then begin
isprim:= false
break
end else writeln('next check with j');

```

```

    end;
    result:= isprim;
end;

// without isprime only with result
function checkPrim(acti: integer): boolean;
var //locals
    j: integer;
begin
    result:= true;
    for j:= 2 to round(SQRT(acti)) do
        if ((acti mod j) = 0) then begin
            result:= false
            break
        end;
    result:= result;
end;

```



Aim



Introduction to a new topic



Important Information



Code example



Analysis of Code



Lesson



Test



Summary



Congratulation

ⁱ Examples\primetester_14project.exe (compiled version)

ⁱⁱ Examples\50_pas_primetester_thieves.txt

ⁱⁱⁱ Examples\14_pas_primetest.txt (simple)