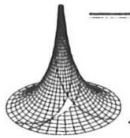

INFODIDACT (2008), 111–141
tmcs@inf.unideb.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Expressiveness of programming languages and environments: a comparative study

GÁBOR TÖRLEY

Abstract. In written and oral communication tools, the support of the understanding of our message have an important role: we can increase the expressiveness and the level of understanding of our topic by approaching it in several ways, i.e. in written methods by highlighting the important parts: in oral by changing tone and other elements of non-verbal communication. In this paper programming languages and developing environments are compared with each other in terms of their methods and their level of support to the solution of programming tasks.

There is a need to have these tools in programming and, of course, in teaching programming. What are the factors that define the distinctness and the legibility of a program? What are the basic principles which give an instrument in programmers' and students' hands in order to create a properly working program from already existing algorithms in the most efficient way? We search for the answers to these questions in this paper.

Key words and phrases: programming languages, teaching programming, programming environments, expressiveness, secondary school education.

ZDM Subject Classification: P43, P44, P53, P54.

1. Introduction

We want to add new knowledge to the ways programming is taught. We are not interested in the educational problems of the first steps. Instead, we are concerned with the relation between the style of programming and the language, which is to be dealt with in the early stages of learning. We assume that

the learner already knows the basic algorithm elements (sequence, branch, loop, procedure, function, use of variables) and he/she has previous experience in a programming language (supposedly Comenius Logo or Imagine).

According to our definition, "*expressiveness*" is the feature of programming languages which (with its structure, syntax, etc.) supports the solution of a programming task. The more expressive the language is, the less mental effort and time is needed by the programmer during coding, i.e. an appropriate question can be whether the language hinders or supports the composition of the program if the algorithm is already known. This problem is similar to the well-known question in arithmetic: What kind of impact has/can have the calculation on paper on the skill of calculus?[1] [1] Nowadays, we work not only with a programming language, but with a complementary programming environment while we create a program. During the study we manage the language and the environment as a complex unity, because for example a good debugging function, which does not belong to the language, can help to get the solution with less thinking.

This paper presents the practice of the first four steps of methodical programming [2]—coding, testing, debugging, error correcting—and compares the languages and environments from our point of view. It is expedient to separate the features of the language and the environment according to the followings [3]:

- Linguistic aspects
  - Understandable keywords
  - Simple and consequent program-structure
  - Simple coding, easy learning
- Developing environments' aspects
  - Highlighting keywords
  - Coding supporting services (i.e. intellisense)
  - Compiling, error messages
  - Debugging system

Our study is restricted to the family of imperative languages.

Let us have a thought experiment: let us imagine ourselves in our learners' shoes! They have the (not absolutely formal) specification and the algorithm ready. The next step is coding.

[1] Arabic/Indian numerals vs. Roman numerals, ..., Chinese numerals.

Instead of the regular "Hello world" example, let us take the following program-item as the basis of comparison which performs our task: a sequence of integers is given. Let us decide if there is any even number in the sequence.

During the programming, the first step is to specify the program, than create the algorithm.

The specification follows the conventions which are applied at the Eötvös Loránd University. (Its formal or informal feature is not important; the point is to provide exact information for the solution method.)

Specification:

Input: $N \in \mathbb{N}$, $Numbers \in \mathbb{Z}^*$, Even: $\mathbb{Z} \to B$
$[B = \{ true, false \}$ — Set of Boolean values]

Output: $Even? \in B$

Precondition: $\text{Length}(Numbers) = N$

Postcondition: $Even? \equiv \exists i \in [1..N]$: $\text{Even}(Numbers_i)$,

where $\text{Even}(x) = \begin{cases} true, & \text{if } x \equiv 0 \pmod 2 \\ false & \text{else.} \end{cases}$

We only present and study the main part of the algorithm. We discuss the read and the write procedures with relation to the programming language and environment, because those will be different at that point.

Algorithm:

Procedure Is_There_Any_Even(**Constant** N: Integer; *Numbers*: tArray;
Variable *Even?*: Boolean):

```
    Variable
        I: Integer
    I := 1
    While I ≤ N and Not Even(Numbers[I])
        I := I + 1
    End While
    Even? = (I ≤ N)
End Procedure
```

At first let us examine what are the features which support the legibility of the *algorithm*. The example presents them well: "talkative" variable, type and procedure names; highlighted (in this case bold) keywords, marking the beginning

and the end of the program blocks, and organising them in one unit (every unit in a different paragraph). We can realize that one part of them is a stylistic question, it is up to the programmer to choose, but the other part depends on the language and the environment.

2. Interesting programming steps

We are examining four programming languages/language-family which are in use in primary/secondary education: (1) Pascal/Delphi, (2) (Visual) C++, C#, (3) Java, (4) Visual Basic. We add to this collection two younger script languages: (5) Ruby and (6) Python, which can have a bigger role in teaching programming in the future.

We investigate the impact of the knowledge, and the existence or the non-existence of the coding rules on the process of coding; or rather the services with which the program editor, the developing environment supports the process of coding. We pay special attention to the solutions of I/O.

In the next chapter we will explore the information content and the rigidity of the error messages, i.e. when the program is declared to be ready by the compiler and how much hidden error can remain for the next phase.

In the testing phase we compare the different environments in terms of the existence of the debugging system.

2.1. Program editing

The fact that programming languages are quite similar to the English language can remove some of the difficulties of Hungarian learners. So if they have at least basic English, this can help the understanding and facilitate the process of coding. We will not review the description of environments and languages.

2.1.1. Pascal/Delphi

In Hungary Pascal is the most widely used language in educational environment for teaching programming. We present two environments: the "classical" Borland Pascal 7.0 (BP) and Borland Turbo Delphi Explorer[2]. The latter one is available for educational purposes for free. There is an other free environment,

[2] `http://www.turboexplorer.com/`

called Free Pascal[3], which is similar to BP's look and its operation. We will not discuss this in this paper. The Pascal language [4] is the programming language of professor Nikolaus Wirth *for educational purposes.*

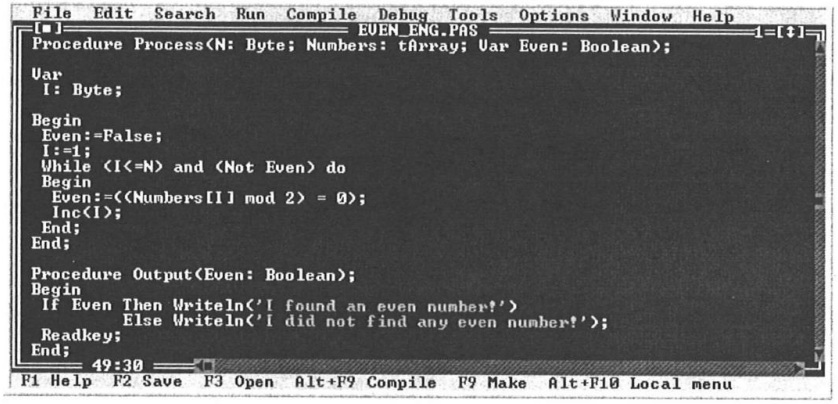

Figure 1. Borland Pascal 7.0

The *program-structure* can be easily followed and memorized. There are separate sections for constants, type definitions, variable declarations, and program body in every procedure and function. [5] According to the coding rules, one of the main features of Pascal language is that it is built up from the bottom to the top i.e. declaration should precede application. Thus, in a procedure, a variable can be used only if it has already been declared; and a procedure can be called only if it has already been declared (at least its head has been declared by `forward`). Arrays are static (although in Delphi there is a special dynamic array as well), that is why the length of the array should be known at compiling (more exactly: the type of the index should be fixed), i.e. at coding. [6]

Pascal is a good example of *understandable keywords* as well. The structure of this language can be easily manageable with basic English. Using of I/O is simple, commands relate unambiguously to read/write (`read`, `readln`, `write`, `writeln`). It is beneficial that the language does not make a distinction between the lowercase and uppercase letters, so there is no need to spend extra time to memorize them.

[3] `http://www.freepascal.org`

There can be errors because of the fact that not every type identifier belong to the protected keywords of language (i.e. `byte`, `word`, `integer`). In the type declaration section the following type definition can occur: `byte = string`, which the compiler will not sign as an error. Obviously, the programmer will find type conflict, if he/she wants to use `byte` as positive integer afterwards.

In Pascal, the parameter passing without access permission implies serious danger. Also, it is hard to understand how to match the parameter passing with `Const`.

The language is not always consistent. It separates the actual parameters with comma, the formal ones with semicolon; such as in the case of semicolons that should be theoretically put to every end of line; this rule is not true; see the case of `if-then-else`.

The designation of the beginning and the end of *complex structures* is not uniform either: i.e. `begin-end`, `record-end`, `while-end`, `repeat-until`.

Branches can be nested, so seemingly ambiguous structures can be created:

```
if a=5 then
if a=4 then
else writeln(a);
```

The above code-part illustrates the "dangling else" problem (it is true for C++, C#, and Java as well). From the code, it is not clear to which condition the `else` branch belongs. In this case, the language does not require the use of `begin-end`, which would make the dilemma clear.

Let us look at BP first. Figure 1 shows well that the language's keywords are highlighted (in white) and digits and strings are presented in different colours (blue and lilac).[4]

Similarly, in Delphi we can create console programs the same way (like above) if we use the `APPTYPE CONSOLE` directive. Because of its likeness we will not discuss this service of the Delphi environment.

The component-oriented "face" of the environment differs a lot from the above. This system thinks in an object-oriented way, it offers this paradigm. Our program will behave as an object; the components manipulate the object's variables through the object's methods. How does it affect clarity? Because of the paradigm-change and the component-oriented way, the implementation of the write/read greatly differs from the way discussed above. The main part (*Process* procedure) is the same in terms of coding (see Figure 2).

[4] The user can set freely the colours, the figure shows an example.


```
procedure TEven.b_NumbersClick(Sender: TObject);

Var
 I: Byte;
 IsEven: Boolean;

begin
 I:=0;
 While (I<=(StrToInt(N.Text)-1)) and ((StrToInt(Numbers.Cells[I,0]) mod 2) <> 0) do
    Begin
      Inc(I);
    End;
 IsEven :=(I<=(StrToInt(N.Text)-1));
 If IsEven then ShowMessage('I found an even number!')
           else ShowMessage('I did not find any even number!');
end;
```

Figure 2. Delphi — Main part

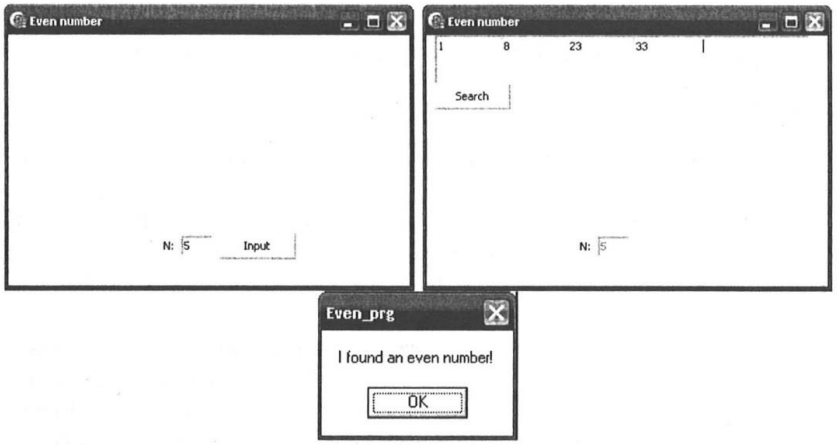

Figure 3. Delphi — "input-output"

The difference between input and output (see Figure 3) raises the question whether we can bridge it on algorithm level? Should we bridge it at all? In case of a visual developing environment, where the representation of the input and the output is done with the help of components, the preparation should be "visual" as well. The role of the I/O part in the algorithm (which we did not discuss in detail previously) is hardly more than to specify this information. Thus, in an environment like this, when we organise I/O, a lot of time is needed for coding.

The highlighting role of colours are the same like at BP.

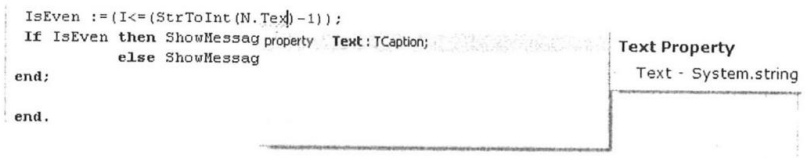

Figure 4. Delphi — "intellisense"

During program editing—like in other 4GL environments—the editor helps the programmer to choose which method or property of the component he/she can reach. So he/she does not need to know exactly the identifier and the parameters' type by heart (see Figure 4). There is additional online help as well: the corresponding brackets are marked with the same colour when we pass them or stay on them. This function can help to prevent and explore bracket-errors.

2.1.2. (Visual) C++, C#

For writing non visual C++ programs, there is a widely used, free editor: Dev-C++.[5]

The C language and its descendants are more permissive than Pascal, which we have discussed above. There are no sections for type-definition and declaration. We can declare variables wherever we feel it is needed. The language is built up bottom-up, like the previous language.

Many people commend the C language's descendants for the reason of their compactness. For example, program blocks can be easily framed (by `{` and `}`); however, they still are segregated clearly from their environment. (See Figure 5) For this compact way of drawing, we can mention some negative examples as well, the "`++`" and "`--`" operators [3]. The difference between "`i++`" and "`++i`", or rather "`i--`" and "`--i`", is not clear from the code, thus they do not support the code's legibility and lucidity. The side-effect of these two expressions is the same; the value of the variable `i` will be incremented (and most of the time it is used for this function), whereas their major effects are different; when evaluating, the expression "`i++`" contains the original value, whereas "`++i`" contains the incremented (by 1) value. The mechanism of "`i--`" and "`--i`" is the same, except for the fact that they decrement the value of variable `i`. Therefore, in the case of

[5] `http://www.bloodshed.net`

a complex expression, the programmer has to be careful which operator he/she uses from these two ones, which differ only slightly syntactically.

The keywords are understandable, but in the case of read and write "`cin`" and "`cout`" do not refer unambiguously to the function of read and write; more "etymology" is needed to understand the keywords better. However, the language is expressive in the way it signals the direction of the datastream: (i.e. `cin >> n`, the value will "go" from console *in*put to `n`).

```
void in(int n, int numbers[])
{
     for (int i=0; i<n; ++i)
     {
         cout << "Please type the " << (i+1) << ". number: ";
         cin >> numbers[i];
     }
}

void process(int n, int numbers[], bool& IsEven)
{
     int i = 0;
     while ((i < n) &&((numbers[i] % 2) != 0))
     {
           i++;
     }
     IsEven = (i < n);
}

void out(bool IsEven)
{
     if (IsEven) cout << "I found an even number!\n";
     else cout << "I did not find any even number!\n";
}

int main(int argc, char *argv[])
{
    int n;
    cout << "This program will print out if there is any even number in the array.\n";
    cout << "How many numbers are in the array: ";
    cin >> n;
    int numbers[n];
    in(n,numbers);
    bool IsEven = false;
    process(n,numbers,IsEven);
```

Figure 5. Dev-C++

Error source can be an accidentally written blank instruction (`;`) instead of the body of complex instructions. Like in C++ and Java, if the programmer does not write the `break` statement at the end of a multiple branch (`switch`), not only the branch where the condition is true will be chosen, but every other

branches below it as well. Thus, its mechanism totally differs from the "classical" `if-then-else`. The compiler does not require the `break` statement, so because of this deficiency, there can be errors. However, the C# compiler requires the `break` statement to be put at the end of branches.

C++ is strongly typed language, though during the check of the type accuracy, the compiler does not mark any error at a type conflict, just gives a warning and it converts the type automatically. There are times when it does this properly, but other times not, thus an extra error source is introduced into the code, and as a result, its distinctness decreases as well.

The C-like languages' common feature is that they differentiate the lower- and uppercase letters at identifiers. The disadvantage of this is that the methods (as well) should be retained letter-correctly. On the other hand it is true that when creating one's own identifiers it is possible to build on this feature: when developing the own name conventions it can be used with benefit.

The visual brother of C++ [8] can be downloaded freely. Visual Studio 2008 Express Edition[6] (VS) contains C++, C# and Visual Basic languages. The Express edition covers just the basic functions, but it is enough for secondary school education.

In Figure 6. we can see the main part of the solution of our example task. The language differs the access to the object level and class level methods. It marks the former with "::", the latter with "->". But is it possible to explain this difference for a beginner in programming? There is a lot of "unnecessary" information in the source code i.e. component initialization, which suggests a "complicated image".

```
private: System::Void b_Search_Click(System::Object^  sender, System::EventArgs^  e) {
             int i = 0;
             int n = System::Convert::ToInt32(N->Text);
             while ((i < n) && ((System::Convert::ToInt32(Numbers[i,0]->Value) % 2)
                {
                    i++;
                }
             bool isEven = (i < n);
             if (isEven) MessageBox::Show("I found an even number!");
             else MessageBox::Show("I did not find any even number!");
         }
```

Figure 6. Visual C++ — a typical code-part

[6] http://www.microsoft.com/express/

The environment has the same architecture as Delphi. The titles of the methods and the properties, which are important for us, are more united than in the case of Delphi, so it is easier to notice them. For example, in Delphi, the button caption is called `Caption`, but the edit box is called `Text`, while VS calls both by the latter name.

Like Delphi, this environment also helps with drop-down lists to find the required component.

```
private void b_Search_Click(object sender, EventArgs e)
{
        int i = 0;
        int n = System.Convert.ToInt32(N.Text);
        while ((i < n) && ((System.Convert.ToInt32(Numbers[i, 0].Value) % 2) != 0))
           {
               i++;
           }
        bool isEven = (i < n);
        if (isEven) MessageBox.Show("I found an even number!");
        else MessageBox.Show("I did not find any even nunmber!");
```

Figure 7. C#

In Figure 7 we can see the solution on C#. In comparison with C++, it can be seen that this language [9] is clearer, more understandable. There is a possibility to write console application, like with Delphi. The operator of the class and object level method is united (.), and in parameter handling the parameters segregate clearly i.e. the value of a parameter: `int a`, the reference parameter: `ref int a`, and the output parameter: `out int a`. In that case if we need a function with more than one output, we need to create output parameters. The commands of I/O refer clearly to their roles (`Read`, `ReadLine`, `Write`, `WriteLine`).

The developing environment highlights more keywords, the heads of procedures are more readable and the source file contains only the source code; the details of components are in an other file.

2.1.3. Java

For Java [10] programming there are free developing tools as well. One of them is Eclipse[7]. The language's coding rules are similar to C++. Nevertheless some specific features have to be mentioned.

[7] http://www.eclipse.org/downloads

Java is an object-oriented language as well, so before starting on something, we need at least a class definition and a main method. Even if we would create a "Hello World"-like program, we should type at least six, maybe, meaningless lines beside "main part's" one line. Is it helpful for performing the task? Should we be familiar with object orientation when writing these kind of programs? (See Figure 8.)

```
import java.util.*;

public class Even {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("This program decides if there is any even number in the array.");
        int n = 0;
        System.out.println("Please type the amount of the numbers!");
        Scanner read = new Scanner(System.in);
        n = read.nextInt();
        int[] numbers = new int[n];
        in(n, numbers);
        boolean isEven = process(n, numbers);
        out(isEven);
    }
```

Figure 8. Java — main method

The parameter handling in Java differs from the languages discussed before. It knows only value parameters, so a copy is made of the value at the parameter's address. This copy is used by the method, instead of the original one. That is why the value of "isEven" should be defined by a function (see Figure 8). This is true only for simple data types. E.g. in the case of an array, a copy is made about the reference, so the changes in array will remain. It is a pity that in this issue, the roles of the language are not united. Just declared values can be delivered as parameters.

The philosophy of the language is not uniform in terms of I/O. The method of write (`System.out.println`) shows well its goal and function; however the read method is more complicated. In the example above we used the `Scanner` class' method. It is evident that the read method looks like an assignment so this takes the code further from algorithmic language.

The developing environment supports the programmer in a lot of ways. Beyond the tools recognized so far, in the case of parenthesizing, the environment inserts the closing parenthesis during programming instead of the programmer,


```
public static boolean process(int n, int[] numbers)
{
    int i = 0;
    while ((i < n) && ((numbers[i] % 2) != 0))
    {
        i++;
    }
    return (i < n);
}
```

Figure 9. Java — process function

or, rather, if the number of the two types is not equivalent, it underlines the incorrect unnecessary parenthesis with a red line. When the cursor is on a variable's name, the editor highlights all the variables with the same name (see Figure 9).

2.1.4. Visual Basic

Visual Basic (VB) [11] is part of the above mentioned VS environment. In fact, VB is an implementation of the Basic language, which intends to extend the basis language's raw architecture with structured and object-oriented elements, but those are linked strongly to the developing environment. It is not simple to proceed because of the obsolete roots. [12]

```
Private Sub b_search_Click(ByVal sender As System.Object, ByVal e As
    Dim i As Integer
    i = 0
    Dim n As Integer
    n = Convert.ToInt32(Me.N.Text)
    Numbers(n, 0).Value = 0
    While i < n And Convert.ToInt32(Numbers(i, 0).Value) Mod 2 <> 0
        i += 1
    End While
    Dim isEven As Boolean = (i < n)
    If isEven Then
        MessageBox.Show("I found an even number!")
    Else
        MessageBox.Show("I did not find any even number!")
    End If
End Sub
```

Figure 10. Visual Basic

From a lingual viewpoint, the VB differs from the first three ones. One statement should be written in one line, so it is unnecessary to separate them by

semicolon. Although the language is typed, it is not so strict as the Pascal-based ones. "Dim", which is used for declaring variables (see Figure 10) defines the size and the type of the elements of the variable. It is a positive example that a special keyword (ByVal) differentiates the parameters passing by value from the parameters passing by address. Also, there is a specific keyword for functions (Function) and procedures (Sub), so it resembles more to algorithmic language.

It is very helpful to understand that each loop and branch has an easily recognizable and learnable keyword for beginning and ending (i.e. While ... End While), so it is simple to write a readable code.

The environment does its best to "find out" which keyword or method the programmer wants to use. As the programmer begins to write something, the alternatives immediately appear on the screen. However, in this system, it is objectionable that the cursor feeds a line when we select the name of a component by pressing Enter. But the writer of the program has not yet gotten the correspondent property. By pressing Ctrl-Enter, the problem can be solved, and until the selection of the right property, the cursor stays in the same line. This feature is unique in VB, it encumbers the initial steps if the appropriate key combination is not known.

According to the default settings, in every case, the compiler evaluates the overall Boolean expression. This is why it was necessary to add an even number to the end of the array (as $N+1$. element) for stopping the loop. It is not helpful, because it makes us deviate from the algorithm. According to our experiences, this kind of evaluating strategy occurs just in this environment out of the examined ones. The remaining environments use the lazy strategy as default.

2.1.5. Ruby

It is a very young language [13] (it was "born" in 1995), which inherited much from Perl, Python and Smalltalk languages. Like the languages discussed above, Ruby's developing environment[8] is free as well.

In comparison with the others, this language itself looks different. It differs a lot in the application of the keywords.

Figure 11 shows that the way how the program handles the data types. This is not so strict: the type of a variable can be known only at assignment (like in PHP and in Perl). This syntactic compliance does not support the legibility of the program and it is prone to generate errors.

[8] http://www.rubyonrails.org/down


```
1  def input(numbers)
2    puts 'This program decides if there is an even number in the array.'
3    puts 'Please type the amount of the numbers!'
4    n = 0
5    n = Integer(gets)
6    for i in 1..n
7      puts 'Please type the ' + i.to_s + '. number: '
8      numbers[i] = Integer(gets)
9    end
10   return n
11 end
12
13 def process(n, numbers)
14   i = 1
15   while (i <= n) and ((numbers[i] % 2) != 0)
16     i = i + 1
17   end
18   return (i <= n)
19 end
20
21 def out(isEven)
22   if (isEven) then puts 'I found an even number!'
23     else puts 'I did not find any even number!'
24   end
25 end
26 #Main program
27   numbers = Array.new
28   n = input(numbers)
29   isEven = process(n, numbers)
30   out(isEven)
```

Figure 11. Ruby

Parameter passing functions are similar to those in Java.

It can be disturbing that a function does not differ radically from a procedure. Considering their heads, they are the same. The only difference between them is the "return" statement somewhere in the program body, which gives a value for the procedure, and so transforms it to a function.

The syntax of the write and the read reveals incomprehensible differences. In case of reading or writing a string, the name of the variable should be written after the statement or the string constant (gets 'text', puts s). In case of numbers, the reading seems to be an assignment, just like in Java (n = Integer(gets)), at writing, there is a need for type conversion (puts n.to_s).

The language is very young, that is why the program editors are not ready for having all of the "comfort" services. It is true in general (like in the case of

the SciTE program in Figure 11) that it does not support the programmer with anything except for highlighting the keywords.

2.1.6. Python

This language [14] is also very young; it was created in 1991. It has a free programming editor too.[9] The performance of the task can be seen in the editor, which can be installed by the default package:

```
# -*- coding: cp1250 -*-
def read(numbers):
    n = input('Please type the amount of the array' ')
    for i in range(0, n):
        numbers.append(input('Please type the '+str(i+1)+'. number' '))
    return n

def process(n, numbers):
    i = 0
    while i < n and (numbers[i] % 2) != 0:
        i = i + 1
    return (i < n)

def out(isEven):
    if isEven:
        print "I found an even number!"
    else:
        print "I did not find any even number!"
#Main program
print 'Welcome, this program decides is there is an even number in the array.'
numbers = []
n = read(numbers)
isEven = process(n, numbers)
out(isEven)
```

Figure 12. Python

The structure and the linguistic elements of the program look like those of Ruby (latter considers Python one of its parents). Compared to the previous languages, it has a new "invention". There are no `Begin-End` pairs, the corresponding program blocks are in the same paragraph, so the programmer should write the parts belonging to different loops or branches in separate paragraphs, otherwise the program will not function as he/she expects it. This is the so-called "margin rule" which provides the legibility of the code. Let us recognize that this makes the coding easier because of the accordance with our algorithmic language.

[9] `http://www.python.org/download/`

Its coding rules are identical to those of Ruby.

The counting loop of the language is strange: its syntax is uncommon and from this it is hard to find out the semantics. The "`for i in range(1, n)`" suggests, that the loop-core will run in case of $1 \leq i \leq n$. That is not true, because the loop-core will run in case of $1 \leq i < n$, that is why Figure 12 shows that the loop will run n times between 0 and n.

The program editor is very simple. It does not provide more services, except for the colouring in supporting the legibility.

2.2. Compiling, the correction of syntactical and static semantical errors

The content and the quality of the compilers of languages and their information (error messages) belong to the feature of the environments. It is important that from the error message the programmer would know what kind of error had occurred, where the error could be, and how it could be solved.

What makes an error message good? It is good if the error message is neat, polite, consistent, positive, constructive and uses active idiom. It describes the error well, and, if it is needed, the right page of the Help menu can be reached directly from the error message [15].

In this paper we cannot show and analyze examples of every kind of errors, that is why we discuss only two types of errors: Let us assume that the student have typed the identifier of one of the statements/variables in a wrong way, and he/she has not closed a program-block.

It is the feature of the *Borland Pascal* environment's compiler that it stops at the first given error and it shows only that one, so if there are more errors, the code should be compiled at least as many times as the number of the mistakes in the program. Time could be spared, if it highlighted all the errors. However, like this, the impact of the subsequent errors does not encumber the programmer, which is beneficial from the educational point of view.

The error messages are not very informative and they are not always clear. We get the same error message ("unknown identifier") if we type wrongly either an identifier of a variable, or a name of a procedure (see Figure 13). It is beneficial that in case the compiler would find an error, the cursor would jump to the incorrect word or at least to its environment. The figure shows that the error message does not draw the attention to the missing `End`, it will be highlighted only after having corrected the slip of "the pen" (see Figure 14).

Or, probably it is not even highlighted in this case. The compiler misses the semicolon; however, the real error is the missing `End`. Directly from the error

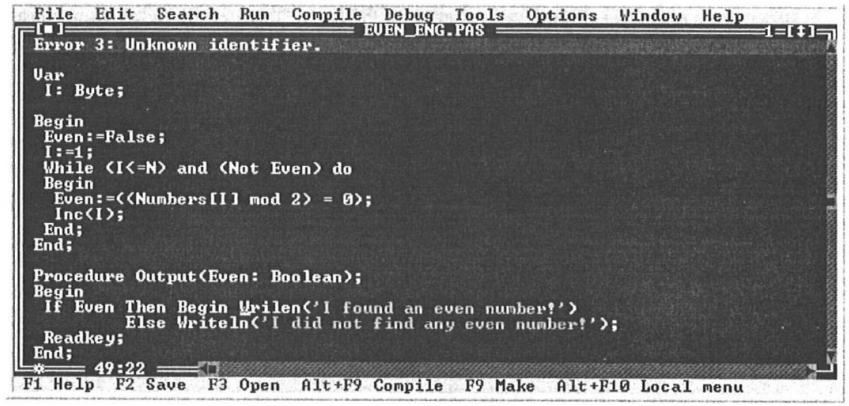

Figure 13. Error message in BP

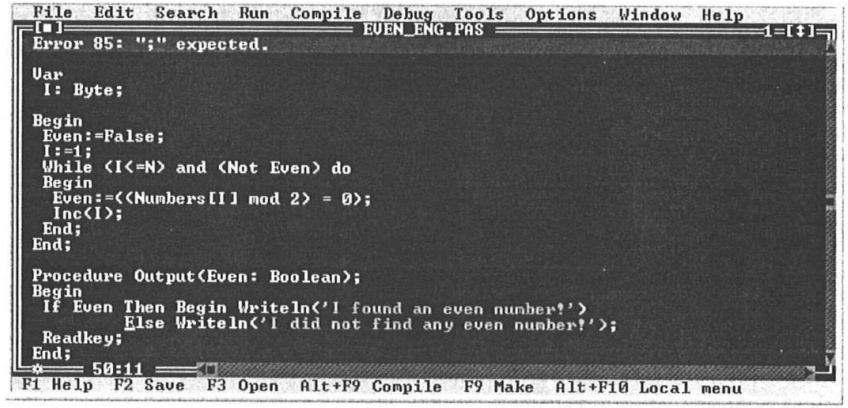

Figure 14. The error message is not accurate

message, a beginner programmer will not recognize what he/she has done wrong. The Help is useless, unfortunately.

In conclusion, the error messages are neat but they do not provide much help and they are not exact enough.

Delphi prints out not only the first but all the possible error messages. Moreover, it gives error signals *already during the coding process*: The compiler in the

background warns us continuously about the actually committed syntactic errors (see left column in Figure 15). At compiling it selects the erroneous lines and indicates the messages several times if there are more mistakes in the program. This solution identifies better where the errors have occurred.

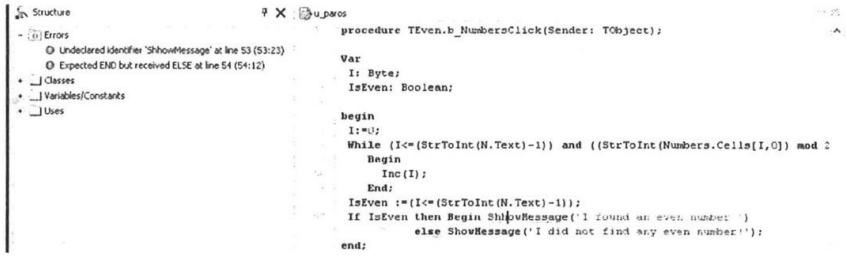

Figure 15. Delphi identifies the errors before compiling

A mistake detected already during the coding, before the compilation is set off, may save a lot of time for the developer. In the case of our example two error signs show up during the coding process, which help to detect the problem, even to find the solution: the red underlined "`ShhowMessage`" and "`else`" draw attention to the wrong "spelling" as well. The error messages are neat, correct, they define the problem. The Help supports us well.

The compiler of Dev-C++ informs us about all of the errors.

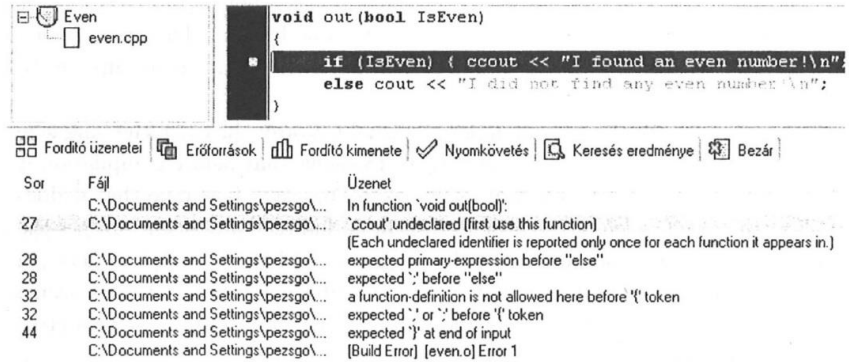

Figure 16. C++: error messages do not help efficiently

The error list above helps to detect the mistyping easily, but the remaining five error messages do not express clearly that a closing bracket is missing. From the error message, a more experienced programmer would find out that the number of opening and closing brackets is not equal, however, for a beginner, it does not provide effective help compared to the effectiveness of the error messages in Delphi.

The error messages of Visual C++ have also very little information content, just like the environment discussed above.

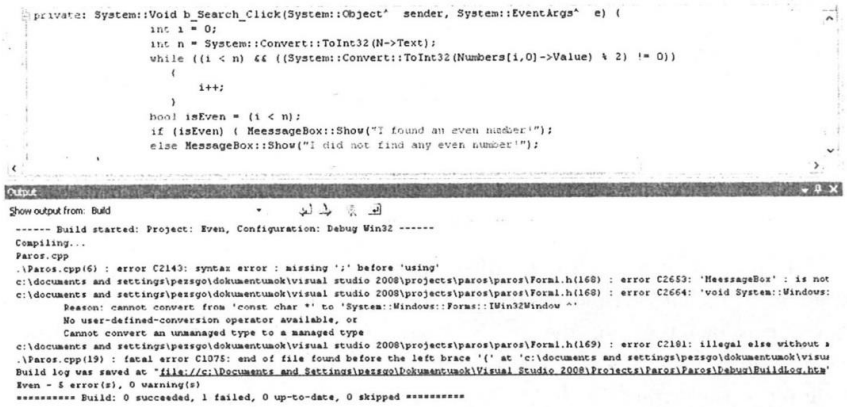

Figure 17. Visual C++: Complicated error messages

The figure above shows that the first incorrect line is not highlighted, that is why we have to click on the lines of the error list one by one. The error, which warns about the mistyping, can be found simply, but nothing refers directly to the missing closing bracket.

The compiler and the error handler of Visual C# are—just like the language itself—more friendly and meaningful. Figure 18 shows that before compilation it will be highlighted that there is a missing closing bracket. It is true that it does not show the exact place of the absence but in an indirect way it encourages the programmer to count the number of the beginning and the closing brackets. In this phase, the mistyping does not show up, but after compilation it does occur.

If the variable "`i`" does not have an initial value, the compiler signals error as well (see Figure 18).

In VB it is very difficult to generate these kinds of errors deliberately, because the environment's intellisense service actively pays special attention to use only

the right identifiers, to have the beginning and the ending of every program block. So if the student writes down "`If condition`", and presses Enter, then "`Then`" and "`End If`" will appear automatically.

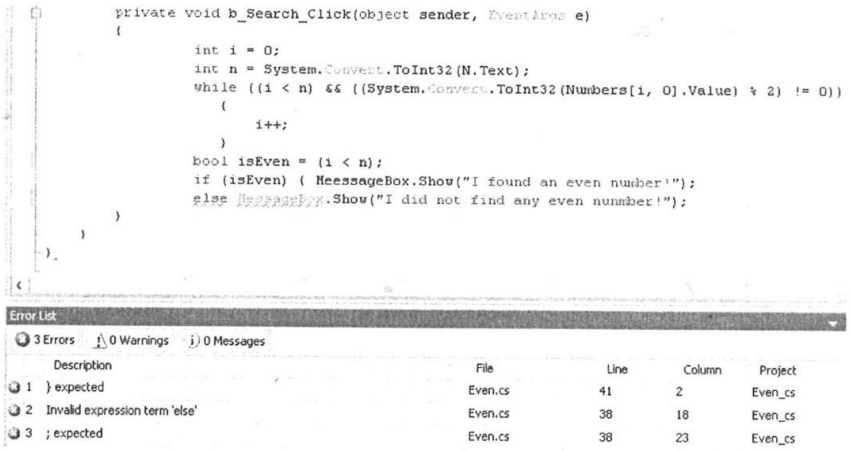

Figure 18. C# gives warning before compiling

The figure above shows a very unlikely error, but it is clear that the environment indicates what kind of syntactic error the programmer has made—without compiling—by a blue underline and a message in the error list at the bottom appears.

The information content of the Java's compiler is the same as the one we have seen in the C# language.

Before compilation, the environment underlines the incorrect words and the error messages show already the real problems. It is a good solution that the system marks the incorrect lines with a red x. The typing errors can be fixed by the "Quick fix" service of the environment. The environment pays attention to the correct definition of the variables (see Figure 21), so it warns us of the semantic errors as well.

Python and Ruby differ from the above discussed languages: as they are script-languages, they are interpreted during runtime by the compiler.

Before running the program, the Python's interpreter checks the code. If it finds error, it halts without any specific error message ("Invalid Syntax").

```
Private Sub b_search_Click(ByVal sender As System.Object, ByVal e A
    Dim i As Integer
    i = 0
    Dim n As Integer
    n = Convert.ToInt32(Me.N.Text)
    Numbers(n, 0).Value = 0
    While i < n And Convert.ToInt32(Numbers(i, 0).Value) Mod 2 <> 0
        i += 1
    End While
    Dim isEven As Boolean = (i < n)
    If isEven Then
        MeessageBox.Show("I found an even number!")
    End If
    Else
        MessageBox.Show("I did not find any even number!")
    End Else
End Sub
```

Error List

3 Errors 0 Warnings 0 Messages

	Description	File	Line
1	Name 'MeessageBox' is not declared.	Form1.vb	24
2	'Else' must be preceded by a matching 'If' or 'ElseIf'.	Form1.vb	26
3	'End' statement not valid.	Form1.vb	28

Figure 19. Visual Basic: "Forced" error, correct error message

```
public static void out(boolean isEven)
{
    if (isEven) { System.out.println("I found an even number!");
    else System.out.println("I did not find any even number!");
}
```

Problems Javadoc Declaration Console

3 errors, 0 warnings, 0 infos

Description	Resource	Path	Location
Errors (3 items)			
Syntax error on token "else", delete this token	Even.java	Even_search/src	line 52
Syntax error, insert "}" to complete ClassBody	Even.java	Even_search/src	line 54
The method println(String) is undefined for the type PrintStream	Even.java	Even_search/src	line 51

Figure 20. Java — every error will be highlighted

The figure above shows that the compiler uses the red colour. After having corrected this error, we get the following message:

Contrary to the previous message this error message defines the problem well and exactly.

The figure above shows that we get the same error message ("Invalid syntax"), but the compiler does not localise the error exactly and we cannot get any other information on the error message which could bring us closer to the solution.


```
public static boolean process(int n, int[] numbers)
{
    int i;// = 0;
    while ((i < n) && ((numbers[i] % 2) != 0))
    {
The local variable i may not have been initialized
    }
    return (i < n);
}
```

Figure 21. Java gives warning before compiling if initialization is missing

```
def out(isEven):
    if isEven
    println "I found an even number!"
    else:
        print "I did not find any even number!"
```

Syntax error

There's an error in your program: invalid syntax

OK

Figure 22. Python: inexact, too general error message

```
def out(isEven):
    if isEven:
    println "I found an even number!"
    else:
        print "I did not find any even number!"
```

Syntax error

There's an error in your program: expected an indented block

OK

Figure 23. Python: exact error message

```
def out(isEven):
    if isEven:
        println "I found an even number!"
    else:
        print "I did not find any even number!"
```

Figure 24. Python: where is the error?

It is also as hard—if not harder—to interpret the error messages of Ruby.

From these two error messages we can recognize that there may be something wrong with begin-end pairs, but there is not a single hint on the meaning of "$end" and "kEND". The mistype error in line 22 ("`pputs`" instead of "`puts`") will be

```
21  - def out(isEven)
22  -   if (isEven) then begin pputs 'I found an even number!'
23        else puts 'I did not find any even number!'
24      end
25    end
26    #Main program
27      numbers = Array.new
28      n = input(numbers)
29      isEven = process(n, numbers)
30      out(isEven)
even.rb:24: warning: else without rescue is useless
even.rb:30: syntax error, unexpected $end, expecting kEND
  out(isEven)
```

Figure 25. Ruby: meaningless error message

identified during runtime, but only in case the input parameter of the procedure "out" is true.

Due to the features of the language (script-language) syntactic and semantic errors will be identified during runtime.

2.3. Testing, debugging, the correction of semantic errors

The dynamic semantic errors are more uncomfortable than the syntactic or static semantic ones, because the program runs, but not correctly: it does not execute what it is supposed to do. It takes a longer time to find and correct them. The debugging system helps the programmer to explore these errors.

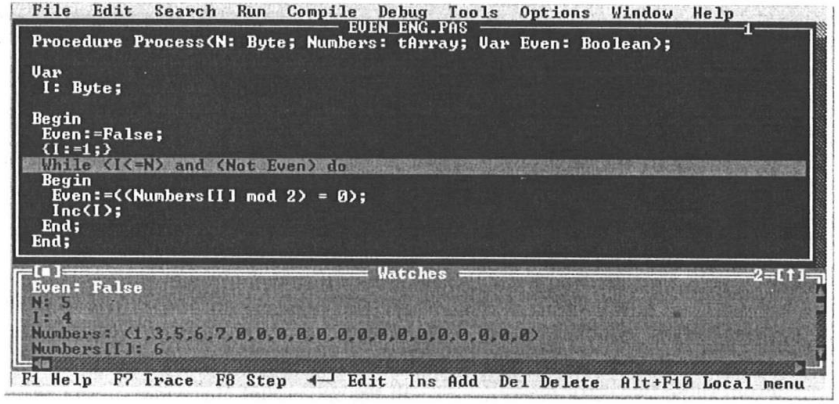

Figure 26. Debugging in BP

What can be the expectation from a debugging system? When can it be regarded good and useful? It is beneficial if the debugging functions are at the same place i.e. in the same menu; it is good if the view allows the code, the content of the variables, and the output to be seen and examined at the same time; and there should be opportunity to create breakpoints and to execute the program line by line, or procedure by procedure.

BP provides an easily manageable interface for the programmer (see Figure 26). The values of the variables inserted onto the watch and the actually necessary functions (on the bottom of the figure) can be seen in a separate window. Breakpoints can be inserted into the code for running just the incorrect part line by line. All the tools of debugging can be found easily in the same menu (Debug).

The errors during the coding process can be detected easily by these tools. In this example the errors are: the lack of the initial value of the variables and the incorrect loop condition.

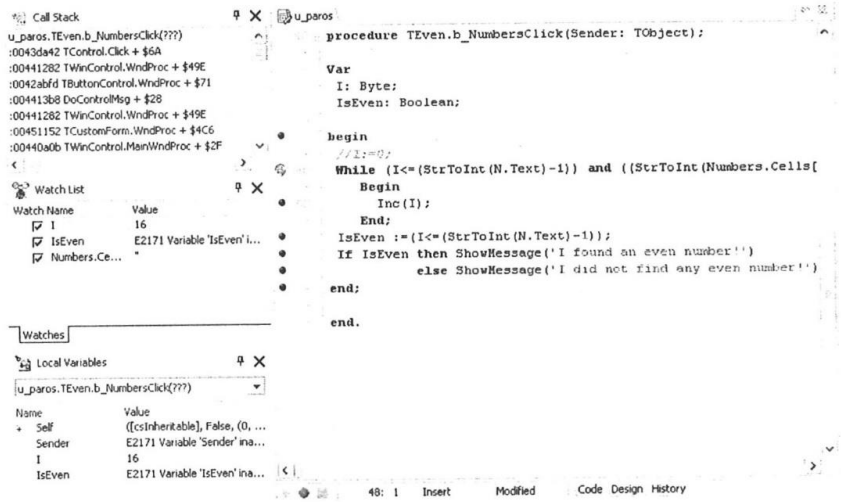

Figure 27. The debugging system of Delphi

VS and Eclipse have a common useful feature —unlike the two script-languages and Dev C++—: they can show pretty much information at the same time and at the same place. It is very helpful that all of the local variables can

be seen automatically and in the code; and during the step by step execution, the programmer can get to know the value of the variables whenever he/she puts the cursor over it, so the program's inner state will be clear for him/her. This makes it easier to debug because everything is at the same place, indeed.

Figure 27 shows a brilliant solution. The variables which are selected by the programmer or which are shown automatically are on the left side. In the middle of the figure it can be seen how the cursor "shows" the value of one of the components. Evidently, the debugging system is easy to use.

Out of searching for errors, the debugging system of visual developing environments is able to help to explore the structure and the inner function of the current objects. The figure shows that the class level variables and their features can be seen by clicking on "`Self`" on the left side.

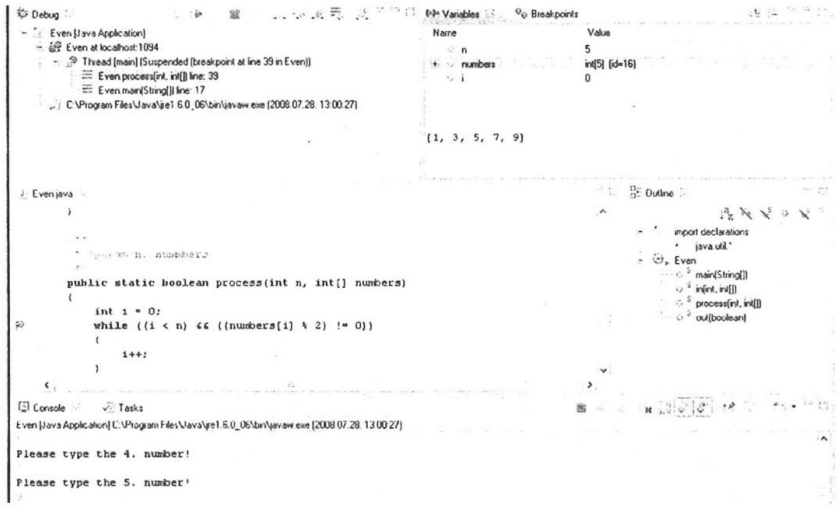

Figure 28. Java — all information is nearby

Eclipse's debugger (Figure 28) is the best in terms of arrangement. It shows all information without problem, even the program's output fits in to the screen confortably. The debugging system can be given a Boolean expression, so the loop condition and the output value of the "process' function can be seen separately. The latter can be found by clicking on "Expressions" tab.

Visual C# is a good example of using colours and messages of the debugging system. In Figure 29, the breakpoint and the current incorrect line are well highlighted and segregated (former in red, latter in yellow colour). The error message is complete; it notifies not only the error but suggests a solution as well.

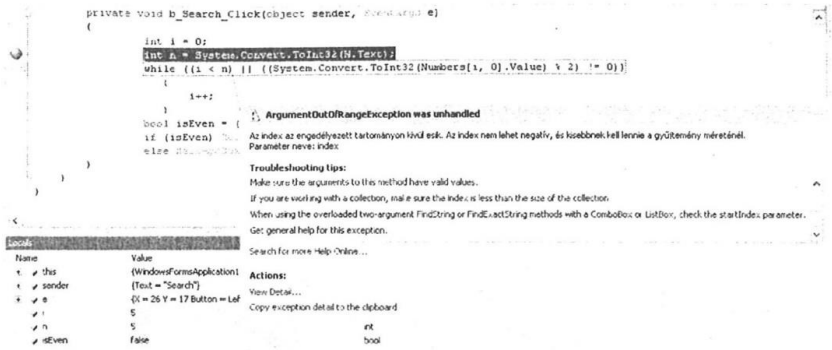

Figure 29. C#'s debugging system — an error message with a lot of information

Dev C++ and Eclipse should be compiled and executed in a special debugging mode in order to follow the inner life of the program. The disadvantage of the former one contrary to the latter one is that the settings should be changed. However, in Eclipse a simple click is just needed to switch to debugging mode.

The disadvantage of the developing environments of the two script-languages in our study is that they do not have any debugging system. If we find a semantic error, the programmer should print out the values of the important variables (see Figure 30). This solution takes a lot of work and a lot of time, moreover; a complex problem could be explored in a more difficult way.

3. Summary

From this comparison, we can see that there is no perfect language or environment; each one has its advantages and disadvantages, too.

In terms of education, it is beneficial if the chosen language consists of easily memorisable keywords and uses simple program structures. It is important to realize how simple it is to write the first *meaningful* program, thus the first program which has practical gain as well. A language is needed which is near to our

```
13  - def process(n, numbers)
14      i = 1
15      #Debugging
16      puts 'n: ' + n.to_s
17      puts 'Entering into the loop...'
18  -   while (i <= n) or ((numbers[i] % 2) != 0)
19        puts 'i: ' + i.to_s
20        puts 'numbers[i]: ' + numbers[i].to_s
21        puts '(i <= n): ' + (i <= n).to_s
22        #End of debugging
23        i = i + 1
24      end
25      #Debugging
26      puts 'Exiting from the loop...'
27      puts 'i: ' + i.to_s
28      puts '(i <= n): ' + (i <= n).to_s
29      #End of debugging
30      return (i <= n)
31    end
```

```
n: 5
Entering into the loop...
i: 1
numbers[i]: 1
(i <= n): true
i: 2
numbers[i]: 3
(i <= n): true
i: 3
numbers[i]: 5
(i <= n): true
i: 4
numbers[i]: 7
(i <= n): true
i: 5
numbers[i]: 9
(i <= n): true
even.rb:18:in `process': undefined method `%' for nil:NilClass (NoMethodError)
        from even.rb:41
```

Figure 30. The "debugger" of Ruby

algorithmic language, so that the student learns and understands the place, role and operation of the basic elements of programming. Pascal's strong (and strict) typing, understandable and memorisable program-structure teaches the learner not to forget to declare a variable, or to define a type; and during the programming, always consider what and how he/she wants to use while implementing the current procedure. The ad hoc variable declaration for the remaining languages does not support this approach. Pascal is a good basis from which we can go on towards object-oriented programming and/or 4GL developing system. The latter can help to understand the object-oriented paradigm thanks to its visuality.

Compared to Pascal, the object-orientation of C++, C#, and Java languages opens new perspective on linguistic level. C++ is too difficult as first imperative

language, and to progress they should use C# and Java. They seem to be a better choice due to their maturity and clarity.

VB has many beneficial features for beginners in programming: it gives a readable and well-structured code, but the linguistic roots and the strong dependence on the developing environment make it more complicated to go on.

The weak typing of the two mentioned script-languages does not facilitate an advanced programming style, in addition, it degrades the legibility of the program because more time is needed to understand the code compared to a strongly typed language. Moreover, they have lot of error-possibilities. It is not advisable to teach Ruby and Python as first languages. It is also questionable if they have any benefit in secondary school education.

In terms of supporting tools to the legibility of the code, there was not any significant difference between the discussed environments. It should be noted that beyond highlighting the keywords, the environments with intellisense service provide more help for programmers.

Between the debugging services of the developing environments—except for the two script-languages and Dev-C++—there were major differences. In terms of error messages, the compilers of C++, Python, and Ruby were the weakest because of their unspecific, irrelevant error messages. Delphi, Visual Basic, C#, and Eclipse help directly to correct errors with their error messages.

There were questions about whether it is worth using an object-oriented language/environment. The concept of an object can be introduced easily, but in this case we should make our algorithmic language suitable for explaining algorithmic thoughts in this paradigm. We need more study to answer how it affects the efficiency of programming education in secondary schools.

The following method seems easier and more understandable for the students: we concentrate only on the important lingual elements in terms of the algorithm in the first phase of education when we teach programming theorems.

In this phase of education, it is not beneficial to apply a 4GL system or one of the mentioned script-languages for executing a programming tasks for a variety of reasons: the C++ language is too difficult as first imperative language, so Borland Pascal's environment seems better to understand and teach the first steps.

Later it is worth changing to a 4GL system, to get to know the object-oriented programming and—just as well—to apply a new language for performing more complicated tasks. As a new language we suggest a more sophisticated variation of

the C language-family: C# or Java, because of their already mentioned beneficial features.

References

[1] D. C. Geary, Biology, culture, and cross-national differences in mathematical ability, in: *The Nature of Mathematical Thinking*, (R. J. Steinberg & T. Ben-Zeev, eds.), Lawrence Erlbaum Associates, Mahwah, New Jersey, 1996, 145–171.

[2] Péter Szlávi, László Zsakó, *Methods in Teaching Programming* (in Hungarian), `http://digo.inf.elte.hu/~szlavi/ProgModsz/SzlaviZsako.pdf`.

[3] Péter Szlávi, *Valuation of the programming languages and the application systems* (in Hungarian), `http://digo.inf.elte.hu/~szlavi/InfoOkt/SzoftErt/IndSzoftErt.html`.

[4] Kathleen Jensen and Nikolaus Wirth, *PASCAL—User Manual and Report*, `http://www.cs.inf.ethz.ch/~wirth/books/Pascal/`.

[5] Gábor Törley, *Programming in secondary school in visual environment*, Diploma thesis, Eötvös Loránd University, Faculty of Informatics, Department of Teacher's Training in Computer Science, Budapest, Hungary, 2005 (in Hungarian).

[6] Péter Szlávi, *Specification, algorithm and program code*, Eötvös Loránd University, Faculty of Science, Department of Teacher's Training in Computer Science, Budapest, Hungary, 1996 (in Hungarian).

[7] B. Stroustrup, *The C++ Programming Language*, 3rd edition, Addison-Wesley Longman, Reading, Mass., USA, 1997, ISBN 0-201-88954-4.

[8] Visual C++ Developer Center, `http://msdn2.microsoft.com/hu-hu/visualc/default(en-us).aspx`.

[9] Visual C# Developer Center, `http://msdn2.microsoft.com/hu-hu/visualc/default(en-us).aspx`.

[10] Java™ 2 Platform Standard Edition 5.0 API Specification, `http://java.sun.com/j2se/1.5.0/docs/api`.

[11] Visual Basic Developer Center, `http://msdn2.microsoft.com/hu-hu/vcsharp/default(en-us).aspx`.

[12] Gusztáv Nagy, Programming languages in education, in: *Computer science in higher education, Programming Languages in Education 2005*, Conference publication (in Hungarian), `http://agrinf.agr.unideb.hu/if2005/kiadvany/papers/E73.pdf`.

[13] Programming Ruby—The Pragmatic Programmer's Guide, `http://www.ruby-doc.org/docs/ProgrammingRuby`.

[14] Guido van Rossum, *Python Tutorial*, `http://docs.python.org/tut/tut.html`.

[15] Mónika Bölecz, *Designing user interface* (in Hungarian), `http://www.szt.vein.hu/~bolecz/szf/felhasznaloiFeluletek.doc`.

[16] E. Horowitz, *Fundamentals of Programming Languages*, Springer-Verlag, 1983.

[17] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice Hall Inc., 1976.

GÁBOR TÖRLEY
EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS
DEPARTMENT OF MEDIA AND EDUCATIONAL TECHNOLOGY
BUDAPEST
HUNGARY

E-mail: `pezsgo@elte.hu`

(Received June, 2008)