ISSN 1679-1614

PROGRAMMING LANGUAGES IN ECONOMICS: A COMPARISON AMONG FORTRAN77, C++, AND JAVA[1]

Wilson da Cruz Vieira[2]
Levi H. Santana de Lelis[3]

Abstract - The main objective of this paper was to compare the computer programming languages Fortran77, C++, and Java under four aspects: runtime efficiency, readability, ease of learning, and reliability. For this comparison, we revised the specialized literature on programming languages and used pieces of codes of these three programming languages. The purpose of this comparison was to provide some objective information for economists interested in learning one or more of these languages.

Key words: programming language, runtime efficiency, readability, and reliability.

1. Introduction

There has been an increasing interest in computer programming languages by economists in recent years. Perhaps the most evident feature of that is the increasing number of papers on programming languages published in economic journals in the last few years [see, for example, Rust (1993), Belsley (1999), Cribari-Neto (1999), Kendrick and Amman (1999), and Nerlove (2004)].

Note that, not so long time ago, most of economic research could be conducted in a satisfactory way with little help of computers. Since then, what we can see is an increasing use of computers in both theoretical and applied economic researches. At least three factors have contributed

[1] Research supported by CNPq. The authors gratefully acknowledge helpful comments from Francisco Armando da Costa. Received June 2005 Accepted August 2005.

[2] Associate Professor, Department of Agricultural Economics, Federal University of Viçosa. 36570-000 Viçosa – MG – Brazil. E-mail: wvieira@ufv.br.

[3] Computer Science undergraduate student, Federal University of Viçosa. E-mail: levilelis@gmail.com.

to that: a) the development of the personal computer accompanied by the rapid evolution of both hardware and software industries; b) the dramatic reduction in computation costs which has permitted a great expansion of the market for this type of service; and c) the development of complex economic models which require some type of computer assistance.

The computer needs software to carry out a specific task, that is, a computer program written in a specific programming language. The software contains some commands (code) that inform the computer what to do and in what sequence. A compiler and/or an interpreter "translates" the programming language (similar to human language) in what specific commands were written into the machine language, that is, the language that the machine can understand and execute.

There are many programming languages that can be used in economic research and each of them has particular aspects and/or is ideally suitable for specific applications. According to Kendrick and Amman (1999), we can divide the computer programming languages in three groups: a) high-level languages like GAUSS, GAMS, Maple, MATLAB and Mathematica; b) low-level languages like Fortran, Basic, C, C++, and Java; and c) languages for programming graphical user interfaces (GUI) like Visual Basic, Visual C++, and Java.

To those economists interested in learning a computer programming language, Kendrick and Amman (1999) have suggested to begin with one of high-level languages according to his/her area of specialization in economics and then work downward in the chain and learn a low-level language. Finally, to those interested on graphical interfaces and Internet applications they suggest Visual Basic, Visual C++ or Java. Cribari-Neto (1999), on the other hand, has suggested the C language for econometricians.

As such a diversity of programming languages and recommendations, an important question arises naturally: what programming language(s) is/

are best suited for economists? This is a difficult question to answer and, besides, as emphasized by Prechelt (2000, p.23), the "debate about different programming languages remain inconclusive." On the other hand, the choice of one or more programming languages depends on the user's needs that can be well diversified.

Although that question is difficult to answer, it is important because the learning of a programming language can be very helpful to an economist nowadays[4]. Without wanting to give a definite answer to that question, this work had, as main objective, to compare the low-level programming languages Fortran77, C++, and Java under four aspects: runtime efficiency, readability, ease of learning, and reliability. In doing so, we have attempted to provide valuable information for economists in their choice of one or more of these programming languages.

The choice criteria of these programming languages were the following: a) existence of compilers or interpreters free of charge; b) better control of execution of tasks by the computer when the programmer uses a low-level programming language; c) great source of resources on programming languages (tutorials, FAQs etc) available on the Internet; and d) flexibility of these programming languages to build personalized graphical interface.

2. Basic concepts related to programming languages

The computers recognize internally only situations of the type "yes-no", "0-1" or "off-on". According to Nerlove (2004), the von Neumann's great contribution to computation was to have proposed that a machine could be programmed to execute a sequence of commands (code) previously stored in its memory as "off-on's". This author has cited, as an example, in an historical perspective, the ENIAC at the University of Pennsylvania, which, in 1946, the switches were set by hand.

[4] Economists as well as other scientists, engineers, etc.

Before the advent of programming languages, the programmer had to use “machine language”, that is, binary bits “0-1”[5], in order to get the computer to execute a specific task. The problem with the machine language is that it is tedious, complex and easy to make mistakes. The first progress in terms of programming languages was the called “assembly” language (semi-symbolic form) that uses instructions (abbreviated words) in a one-by-one correspondence with instructions in machine language. An “assembler” translates the assembly language symbols into machine language.

The great progress that has permitted the more effective use of the computers was the invention of the low-level programming languages. According to Nerlove (2004), the firsts low-level programming languages invented were: Fortran (**For**mula **Tran**slation) in 1957 by IBM; Algol (**Algo**rithmic **L**anguage) in 1958 by a European consortium; and Cobol (**C**ommon **B**usiness **O**riented **L**anguage) in 1960. Since then, new versions of these languages have appeared[6] and other programming languages were invented. Nowadays, the great majority of software/applications has been developed using these old or new programming languages.

A low-level programming language has a strict syntax (words and a set of rules) and, when standardized, become portable, that is, it can be used in different computer systems. A computer programs written using this type of language to curry out a specific task needs a compiler and/or an interpreter that translates the instructions in machine language. In this process, there is loss of runtime efficiency (computing time) when we compared the same program written in machine language.

Low-level programming languages can also be used to write Internet applications. One of the most common ways to write specific programs that access the Internet is by mean the Common Gateway Interface

[5] It is possible to use numeric code not binary, but these codes must be translated in binary code because these are the only one that can be directly executed by computers.

[6] From these three programming languages, only Fortran continues to be used extensively, specially by scientists and for engineering applications.

(CGI), a standard protocol (it is not a programming language) to run program in a server via Web[7]. It is important to note that there are specific programming languages whose computer programs can run in a browser (Internet Explorer, Netscape etc) such as Java and JavaScript.

3. Criteria for comparison of programming languages

There are various criteria that can be used to compare computer-programming languages[8]. From the point of view of the economist, that frequently uses numerical methods and econometrics and who is not a professional programmer (programming expert), some programming languages aspects can be more interesting such as runtime efficiency, readability, ease of learning, and reliability.

The importance of the runtime efficiency (computing time) becomes evident when the structure of economic problems becomes more complex and has great dimension to a given hardware. Thus, programming languages that have numerical libraries more precise and efficient would be more adequate to the economist work.

Readability is related to ease of reading computer program codes. If a code is easy to read and to understand, it is readable. The ease of learning of a language is determined by its simplicity and readability, which is of great relevance to non-professional programmers like most economists. In the same way, portability is another relevant aspect to be considered because it helps the work of the user that need to use different computer platforms. Finally, reliability depends on various aspects of a language, such as portability, and dynamic memory allocation.

Taking into account the four aspects previously highlighted, we have compared, in this paper, the programming languages Fortran77, C++, and Java. These are languages with many similarities, but they have

[7] Web is an acronym for World Wide Web.
[8] See, for example, Prechelt (2000).

some distinct aspects. They are also very popular in the academic mean and in engineering applications. We have done this comparison based on the specialized literature on computer programming languages and pieces of codes written in these three languages.

4. Results and discussion

In the following the pros and cons of the three languages studied are analyzed according to previously defined aspects. This analysis is based on the specialized literature on computer programming languages and pieces of codes presented to emphasize specific aspect of each language.

Runtime efficiency

The first version of Fortran was released in 1956. At that time, the hardware was expensive and questions about reliability and maintenance were not so critical. Hence Fortran was created to be simple and efficient. Since then, new versions of Fortran were released with new functionalities since than such as Fortran77 (structured language) and Fortran90 (object-oriented language)[9]. The addition of new functions to the language made it less efficient in computing time. However, this language continues to be more efficient than C++ and Java if we consider the same task to be executed using the same hardware[10].

C++ emerged as an evolution of the C programming language. We may recognize it as the object-oriented version of the programming language C. Although C++ is less efficient than C, it is still very fast. The creators of C and C++, in some points, prioritize efficiency instead of portability and reliability.

[9] In an object-oriented programming languages we can define functions (types of operations) in addition to data type of a data structure to get an **object**. The objects of a program can be related or we can simply create a new object that **inherits** many of its features from existing objects. This aspect and others make object-oriented programs easier to modify than structured programs.

[10] Lelis and Vieira (2005) have done this comparison (computing time). These authors have used the Simplex Algorithm (Linear Programming) coded in these three languages.

The first version of Java was released in 1995, that is, a period when the hardware has became relatively cheap and abundant. For this and other reasons, the Java creators' prioritize others aspects such as portability. To make this language totally portable, it was necessary to create a Java Virtual Machine (JVM). After the compilation procedures, the results are bytecodes that can be interpreted by the JVM. So the bytecodes (Java programs) can run in any operation system that has a JVM internally.

This type of compilation executed in a Java environment is classified as hybrid, that is, the code is first compiled and afterward interpreted. Note that this process makes the code more efficient than if executed in a language purely interpreted and less efficient than if executed in a language purely compiled such as Fortran77 or C++. According to Nerlove (2004, p.191), "compilers transform an entire program from one language to another (for example, the assembly language) for subsequent execution; interpreters execute a program sequentially, translating at each step. Compiled programs almost always run faster than interpreted programs but are a lot harder to debug".

Recently compilers called Just-in-Time (JIT) have been embedded into Java Virtual Machine to improve the runtime efficiency of Java. The function of theses compilers is to translate bytecodes in machine language with gains in computing time. With these software improvements and the hardware evolution the differences in runtime efficiency among programming languages will become narrower in the coming years.

Readability

The readability is related to ease of reading and to be able to understand a computer program. Note that it is much easier to find errors and to alter codes if the program is readable. This aspect is very important and most programming language creators' nowadays is concerned about this in the development of a new language. The reason for such a preoccupation was the so-called "software crisis" (Pressman, 1997) that has its roots in difficulties with program maintenances.

At the time Fortran was invented there was not great preoccupation with readability and this language had some restrictions related to this aspect. This problem was partially eliminated in the version Fortran77 because the language has become structured and the existence of loop commands has diminished the use of **goto** command. Note that the goto command obligates the programmer “go up” and go down” in the codes and this process makes difficult the program understanding.

Fortran77 has aspects that influence positively the readability of the programs. It is very difficult a programmer to find unknowns code because there are little commands in this language, which enhance its simplicity. The possibility of division of the program into subprograms and procedures also contribute to its readability.

The readability of the C++ language is a critical question because this aspect was sacrificed to get more efficiency. A classical example is the copy of two vectors whose codes can be written in the following manner:

```
for (;s < vectorSize && *p=*q;p++,q++,s++);
```

Even experienced programmers can have doubts about the results generated by this command. On the other hand, only one line of codes is sufficient to do the copy of two vectors and to solve the problem. Note that, in this case, the readability was sacrificed in detriment of the saving of codes.

The use of pointers contributes widely to the low readability of programs written in C++. The pointers are used to allocate dynamically computer memory, which become its use more efficient. They are known as the **goto** of data structures. As an example, consider the following command:

```
p = q -> tail;
```

Note that it is difficult to know, at a glance, what the data structure is being actualized.

Another determinant factor for inclusion of pointers is related to the efficiency of program executions. If a programming language does not give the possibility of use of pointers it must include a system for memory management of the utilized space. This system is known as garbage collection and reduces the efficiency.

Although the **goto** command is a reserved word for Java, the compiler does not accept it. The Java syntax is widely based on the syntax of C++, but has some aspects that give it more readability. The decision of the creators of Java do not permitting the **goto** and pointers let this language more reliable and readable.

Ease of learning

If a programming language permits to solve a problem in many ways, it is more difficult to learn. Fortran77, for example, is a language easy to learn because has little commands. Most of the time this language does not have many ways to solve the same problem. Thus, the chance a programmer has to find an unknown code in a program written in this language is very low.

C++ has 59 reserved words, which is a very high number when compared to other languages. The compiler does not permit the use of these words to declare variables, functions, classes etc. C++ has many commands and offers many ways to solve the same problem which difficult the learning of this language. As an example, the following commands have the same effect on a specific context:

```
c = c + 1;
c++;
++c;
c+=1;
```

The programmer has to be careful when using pointers. If they are used incorrectly, they can cause errors difficult to find. Learning to use pointers correctly demands much time of studying and much programming practice. Instead of other languages such as Java, the C++ programmer has great responsibility in the dynamic memory allocation.

Java has 49 reserved words, which is also a very high number. Even though there are situations where Java offers more than one way to solve the same problem, there are other aspects that become it easier to learn than C++. These aspects are: Java does not permit the use of pointers and has a garbage collection. The garbage collection "cleans" automatically an area that has not been in use, leaving it available to new uses. This function of the garbage collection is very important because the programmer can spend more time in other aspect of programming than solving problems related to dynamic allocation of memory.

The allocation of memory is very important and requires good practice of programming to get efficient results. To show how the Java programming language manages this aspect consider the following pieces of codes designed to allocate a matrix in the three languages, that is, Fortran77, C++, and Java:

Fortran77:

```
…
real matrix (100,100);
…
```

C++:

```
…
double** matrix; // pointers declaration
/* matrix allocation according to values of height and width*/
matrix = new(double*[height]);
for(int i=0; i<height; i++)
        matrix[i]=new(double[width]);
…
/*after having used the allocated area, it is necessary to reverse the process*/
for(int i=0; i<height; i++)
          delete[](matrix[i]);
delete[] matrix;
```

Java:

```
//variable declaration and height allocation
double matrix[][]=new double[height][];
/*matrix allocation according to height and width*/
for(int i=0; i<matrix.length; i++)
        matrix[i] = new double[width];
```

Once Fortran77 does not support dynamic memory allocation the programmer must know a priori how the dimension of the matrix is. If the programmer does not know a priori the dimension of a matrix, he/she should allocate more space than necessary and, in this case, there is waste of memory.

In the C++ language the programmer can allocate memory for a matrix easily with pointers especially if he/she does not know its dimension. On the other hand, the use of pointers can make difficult the memory allocation if they are not used efficiently. Note that, in the case of Java, it is not necessary to reverse the memory allocation explicitly. When the memory is not in use, the matrix goes to the garbage collection.

Reliability

Fortran77 is a language with low reliability. Its creators did not concern with questions such as exception handling and free use of reserved words, which limited the readability and reliability of the language. If a program is difficult to read and to understand, it is not reliable because there is more chances to make mistakes. The readability affects directly the reliability.

C++ has exception handling that contributes to the reliability of the language. With exception handling if there are errors, they can be eliminated during the execution of the program. In many situations, the C++ language sacrifices reliability in detriment of efficiency. For example, this language does not have garbage collection because it reduces the time of program execution.

Another aspect that reduces reliability in the C++ language is the use of pointer. The programming with pointers requires constant attention of the programmer because it is common to commit errors with their manipulation. Common problems that happen with the manipulation of pointers are:

a) Type system violation errors

Consider the following example:

```
int main() {
        int k = 0, l = 10;
        int * m = &l;
        m--;
        k = *m + 1;
        return 0;
{
```

In the program above, an integer-pointer was declared to integers, which are called m. In the beginning, it points to the integer variable l, but when the value of m is reduced, we do not know more to what type m is pointing. In this case, k receives an unexpected value.

b) Outstanding objects

Consider the following piece of code:

```
int k = 4, j = 10;
int* m = new int[j];
int* p = new int [k];
p = m;
```

In this piece of code, two vectors were allocated, one with dimension 10 and the other with dimension 4, both pointed respectively to **m** and **p**. When **m** is attributed to **p**, the vector of dimension 4 to which **p** has pointed stays lost in the memory, that is, it becomes not accessible. If this type of problem occurs frequently, it causes the leak of memory, that is, the available space to allocate is reduced.

c) Outstanding references

Consider the following piece of code:

```
int g = 10;
int* m = new int [g];
int* p = m;
delete m;
```

Note that we have allocated dynamically a vector of integers and next we have done the pointer **p** to point to the same vector. When we remove the space allocated to m, **p** becomes an outstanding reference, that is, it keeps pointing to garbage.

The reliability was one of the main aspects observed when the Java language was invented. Java has others aspects that become it one of the most reliable languages nowadays. Some of these aspects will be discussed in more details in the following.

a) Hybrid language

When Java was invented, its initial purpose was to be a portable language that, in principle, could be run in any electronic device such as televisions, microwaves, liquidizers, cell phones, and vacuum cleaners. To be executed in these devices, it is necessary an interpreter of Java programs installed in them. However, if the Java were purely interpreted, this would reduce considerably the reliability of this language. Consider, for example, the following pieces of codes:

```
if (a) {
        // code 1
} else {
        // code 2
}
```

In a language purely interpreted, if the codes above were executed many times, it could occur the situation in what the statement "a" is true and only the code 1 is verified. If the software were released to the market, it could happen the situation in what the statement "a" is false and the code 2 is interpreted. Note that logical and syntax errors can emerge if the later were not tested. To avoid this type of problem and to gain efficiency, the creators of Java have became it hybrid, that is, the Java program is first compiled and, after that, it is interpreted.

b) Garbage collection

To leave also the codes more reliable, it was implemented in the Java language the garbage collection. As pointed out earlier, the garbage collection manages the dynamic memory, that is, "clean" the areas that are not more in use. This function realized by the garbage collection reduces considerably the possibility of errors.

c) Dynamic verification of indices

Consider the following pieces of codes:

```
try {
   do {
      if(matrix[j][positionI
         aux1 = matrix[j][t
         positionVariable =
      }
      j++;
   } while (aux1 < 0);
} catch (IndexOutOfBound
     // exception handling
}
```

Java possesses dynamic verification of indices and the problem presented in the pieces of codes above (simplex algorithm)[11] can be solved elegantly with the exception handling. If we try to access an index out the matrix in

aux1 = matrix[j][b] / matrix[j][positionHighestZ];

automatically would be released an exception, an **IndexOutOfBoundsException**, and the flow of the program would go to the **catch** block that treat the exception. Without the dynamic verification of indices, the program would finish with errors and the user would not know what would have happened.

d) Java does not permit use of pointers

The use of pointers is not permitted in the Java language. This aspect avoids programming errors such as type system violation, outstanding objects and outstanding references. The absence of pointers leaves the codes more readable and easier to be modified.

5. Concluding remarks

In this paper we have compared the programming languages Fortran77, C++, and Java according to the following aspects: runtime efficiency, readability, ease of learning, and reliability. In doing this comparison we have attempted to give valuable information to economists interested in learning one or more of these languages.

After the presentation of the pros and cons of these three languages, we have concluded that the Java language is the best option for economists that are, in most cases, not professional programmers (programming expert). Besides being a language versatile and relatively easy to learn, Java helps the work of the user because a series of aspects such as garbage collection, exception handling, does not permit the use of pointers, dynamic verification of indices and high reliability.

[11] See Lelis and Vieira (2005) for more details.

Another aspect of Java language we did not emphasize is that through the Java Server Pages (Servlets) we can easily do the communication between a HTML page and a Java program. Once a program has been written in Java all what we need is to create an interface for the right case, that is, Web or Desktop. In other words, the same code can be utilized in Web or Desktop applications.

The drawback of the Java language is the runtime efficiency when compared to Fortran77 and C++. Note that the difference in runtime efficiency among these languages is evident only for great problems that require many hours or days of computing time. For small problems, this difference is insignificant. The trend is the reduction of the difference in runtime efficiency among these programming languages with the hardware evolution.

Finally, in the case of Brazil, the failure to support the Java technology in a free basis has limited the diffusion of the Java language. As an example, we mention the case that there are not Java Web Servers free of charge in Brazil. In recent years, the Brazilian government has encouraged the use of free software by public institutions. This is a good opportunity to give more support to this technology in a free basis especially in the academic mean and public institutions.

References

BELSLEY, D. A. Mathematica as an environment for doing economics and econometrics, **Computational Economics**, 14: 69-87, 1999.

CRIBARI-NETO, F. C for econometricians, **Computational Economics**, 14: 135-149, 1999.

KENDRICK, D. A.; AMMAN, H. M. Programming languages in economics, **Computational Economics**, 14: 151-181, 1999.

LELIS, L. H. S.; VIEIRA, W. C. **Linguagens de programação em economia**: análise comparativa e desenvolvimento de aplicativo. Viçosa: DER/UFV, 2005. 102p. (Relatório de Pesquisa).

NERLOVE, M. Programming languages: a short history for economists. **Journal of Economic and Social Measurement**, v.29, n.1-3, 2004. p.189-203.

PRECHELT, L. An empirical comparison of seven programming languages. Available at: <htpp://www.cis.udel.edu/~silber/470STUFF/article.pdf> Accessed in: December/2000.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Editora Makrom Books, 1995. 1025.

RUST, J. GAUSS and MATLAB: a comparison. **Journal of Applied Econometrics**, 8, 307-324, 1993.

Resumo - O principal objetivo deste trabalho foi comparar as linguagens de programação Fortran77, C++ e Java sob quatro aspectos: eficiência (tempo de computação), facilidade de leitura, facilidade de aprendizagem e confiabilidade. Para esta comparação, foi feita revisão da literatura especializada sobre linguagens de programação e utilizadas seqüências de códigos escritas nestas três linguagens. Esta comparação teve como propósito fornecer informação para economistas interessados em aprender uma ou mais destas linguagens de programação.

Palavras-chave: linguagem de programação, eficiência, facilidade de leitura, confiabilidade.