

NEW EVALUATION OF THE LANGUAGE C FOR EDUCATIONAL, ENGINEERING AND SCIENTIFIC PURPOSES

Petre Tzv. Petrov

MicroEngineering, Sofia, Bulgaria

Abstract

The focus of the paper are the difficulties experienced with the digital programming language C and its derivatives when these languages are thought to students simultaneously with other programming languages. Only part of the most confusing and perturbing learning process characteristics of the language is discussed. Suggestions of repairing some of the drawbacks are proposed. It is considered that the “minimalism” of the digital programming language C should not go against the logic of the human language and mathematical notation in its basic forms. During the teaching of the subject the logical justification of the material is important and the taught programming language must be a logical follower of the human language, standard mathematical notations and pseudo code. There is a strong need to develop programming languages for students, newcomers and occasional programmers. The paper is not complete review with corrections of the C language and its standards. ANSI C or C89 is the main discussed dialect of C.

Keywords: Programming languages, educational difficulties, pedagogy, psychology, terminology, quality, educational ecology.

INTRODUCTION

The programming language C is created around 1970 [1]. As it could be seen from that paper it represents significant educational difficulties for the learners, teachers and programmers. There are several version of the C language including ANSI version and ISO/ICE standards but they have small differences between them, e.g. the first version of C, ANSI C, Turbo C, Microsoft C, Quick C, Lattice C, etc.

C language influenced more or less many programming and script languages as C++, C#, Cshell/tcsh, Objective C, Java, JavaScript, Perl, PHP, etc. Although the ideas of C have changed into time, more that forty years after its creations it was seen that a lot of its drawbacks are irreparable and are representing a significant difficulties for the learners, teachers and professionals. This paper is addressing particularly the topic and is listing some of them.

The paper is based on research done in classes and observations made in professional environment. It is not intended to show the strong sites of C and similar languages (e.g. they are the most popular “high level” programming languages). The main goal is to show the weak sides which are making them impenetrable to some learners who need the

digital computer as educational and professional tool.

Also the main goal of the paper is to call for introduction of non C-like languages closer to the “standard” human language, pseudo code based on the human language and with established mathematical notations. It is preferred to have localized programming and pseudo code languages, not only digital languages based on English and its dialects.

THE IMPORTANCE OF THE PROGRAMMING LANGUAGE

The coding is occupying only 5 to 20% from the classroom and professional activities within a software project. This could lead to the wrong conclusion that selecting a programming language has a little importance to the final cost of the education or the industry project. This is wrong due to the following reasons:

* The lessons and homework preparation and time of mastering of programming language vary in large scale from language to language, e.g. there are languages witch are relatively easy to master (e.g. Algol-60, QBASIC and Pascal) and languages which are very difficult or impossible to master (C, C++, C#, Visual Basic, etc.).

* The working atmosphere in the class room and in the office of the software firm is strongly influenced by the chosen language. C-like languages seem to create nervous and exhausting atmosphere. On the contrary QBASIC and Pascal seem to creating friendlier environment.

* The cost of the verification and correction of the students' works and maintenance cost of the software project is strongly influenced by the selected language.

* The cost the documentation, reuse, the readability and maintainability of the software are influenced in many ways by the programming language.

* The way of thinking of the student and the software developer is strongly influenced by the language and this could change to some degree their private life.

* Problems as contradictions with "standard human grammar and logic", "strange" lexical details, unexpected syntax rules and semantics significantly influence the productivity and the overall human satisfaction from the programming language. The programming language must follow the logic of the human language and use the standard mathematical notations as close as possible. This is natural because programming is combining human language and mathematics in one tool.

* Changing, modifying, "bug fixing" and the price of the introduced new "bugs" after these activities are function of the selected programming language.

* Structure of the computer program depends of the language and is impacting the readability, the cost and effectiveness of the code reviews.

* When well designed programming language is used for describing the algorithms and to create appropriate very close pseudo-code this is making the development, testing and the documentation much easier.

* The availability and the price of comprehensive printed and electronic materials and tools concerning the selected programming language is affecting the entire learning and developing process.

* Badly designed programming languages could have huge test and validation costs and lead to disastrous results, especially in embedded systems.

From the points above the obvious conclusion is that the selection of the program language is important, expensive, vital and live long investment for the projects and the education as for example the learning and practicing of a foreign language.

It must be noted that the evaluation of a program language could be subjective to some degree because depends on the way of thinking of the individuals, including educational, professional, social and cultural backgrounds.

MAIN DIFFICULTIES OF TEACHING AND LEARNING THE C AND SIMILAR LANGUAGES

Below are listed some of the main difficulties when C and similar languages are learned and taught. The reasons for most of them are explained briefly.

* Raising psychological tension and creating conflicting situations due to unfamiliar and unexpected syntax and constructions;

* Unjustified basic principles of the language (minimalism, over simplification, uncertainty, unclear terminology);

* Negligence to the expectations of the learners in many of the expensive and voluminous published printed materials with teaching purposes [2-24];

* Unjustified small number of key words contradicts to the duplication of the functions of some of them;

* No explicit interrupts and exceptions handling. They are important for embedded systems;

* Unjustified undefined behavior: e.g. reading the value of a variable before the initialization, access outside the bounds of the array, overflow, underflow, etc.

* Unsigned numbers in C are practically useless in the programming world because it is much simpler and safe to consider that all numbers have signs. Also practically all processors has a "sign flag" called sometimes "negative flag".

* Lack of "sign", "carry", "half carry" ("auxiliary carry") and "overflow" keywords. They are important for embedded systems.

* Significant cognitive load of the learning process.

* Starting with low case letter contradict to the standard grammar and is perturbing feature.

* Starting with formatting symbol as % is found uneasy.

* The “register” variable is contracting to “portability” and “high level language” due to the enormous difference in the programming model of the digital computers. Consequently the “portability” of C is more fiction than a reality.

* The “low-level capabilities of C” are weak and controversial. In practice they have little positive impact, if any, on the developed software.

* Loop and combinational structures and blocks of code are not easily identifiable. There is no “Begin_Name(ID) – End_Name(ID)” key words.

* The access to the resources with “pointers” is so complicated that it represents difficulties not only for the learners of C but also to the experienced programmers who are using the C language from time to time.

* Function parameters are passed by value. The difficulty to interpret unclear mechanism of “pointers” could be used to pass parameters “by reference” or “by address” but this is explained in unclear way [1].

* Misleading weak typing e.g. ASCII characters can be used as integers. Although that possibility is not so bad it should be controlled more strictly and warning should be given.

* Low level access to the memory through converted pointers is possible and dangerous. If it exists in that case the low level access to the I/O should also be provided through some mechanism.

* The large number of compound operators is very confusing because it contradicts the mathematical notation and its logic. E.g. instead of writing:

`a+=1;`

it is much clearer to write simply one of the following:

`a+1;`

`incr (a);`

`a = a+1; etc.`

The question is: “Is it really necessary to abbreviate the notation as `a=a+1`; `a=a-1`; etc.

mixing things from the left and right side of the sign “=”?”

* It is considered confusing that the preprocessor directives do not end with semicolon, e.g.

```
#define HIGH 10 /* Higher limit*/
```

The sign # is enough to show that the line is for the preprocessor. A semicolon should be obligatory terminator for all lines. This is the straightforward solution.

* The constants are fully capitalized and this is perturbing:

```
#define LOW 100 /* low limit*/
```

It is much better to use something like the following expression:

```
#define constant Const_Low 100; /* low limit*/
```

```
#Define Constant Const_Low 100; /* low limit*/
```

```
#Define Constant S16b_Const_Low 100; /L. limit*/
```

* The function `printf` and some more functions from the libraries have confusing syntax.

* Particularly confusing is the usage of “==” as test of equality instead of abbreviation as “EQ” or “.EQ.” or “.e.q.”;

* The well documented programs in C are still incomprehensible due to the particularity of the language.

* Unclear and unjustified terminology in C and similar languages. In fact comprehensible “Glossary on C-terms” for educational purposes cannot be found.

* Low readability of the programs in C or incompressible statements. E.g. the statements below are relatively short and simple but are difficult to read, justify and understand. They look like a piece of cryptographic work:

```
while ((long [i]=line[i]) !='\0') ++i;
sign=(s[i]=='-')?-1:1;
for (j=i, k=0; a[k]!='\0' && s[j]==a[k]; j++, k++);
printf ("%d %d\n", ++n, power (2,n));
printf ("white = %d, other = %d\n", nwhite, nother);
for (;;)
*iq=10* *iq + (c-'2');
printf ("%s%s", *++ar1, (ar2>1) ? " " : "");
*x++ = a;
```

* There are no clear relation between the terms for the allowed data types and the presentation of these types (8 bit, 16 bits, 32 bit, 64 bit, signed, not signed, “floating point”, etc.). E.g. the terms “char”, “int”, “float”, “double”, “short”, “long”, “long long”, etc. speak noting about the exact presentation of the data. It is easy to confuse these terms with the names of the data structures because there is no rule always to start the name or identifier (ID) with capital letter.

It is much clear when the keywords for the data types speak clearly about the format of the data, e.g.

Si16b – signed integer 16 bits,

Se16b – signed entire number 16 bits,

Sfp32b – signed 32 bits floating point,

Sm24e8 – signed floating point with 24 bits for the mantissa and 8 bits for the exponent of the number,

BCDus8d – eight digit unsigned binary coded decimal number, etc.

* Starting octal numbers with the letter ‘o’ or ‘O’ is confusing. Starting with “Q” is much more appropriate and cannot be mixed with “0”.

* The logic of the explanation of the language and the programming in C looks strange [1].

* The grammar of C contradicts to the “standard” human language and grammar.

* The notation of C does not follow the “standard” mathematical notation.

* Oversimplified and in the same time overcomplicated and strange syntax.

* Over-confidence on the programmers. But the programmers are humans and the humans should not be trusted, especially in large projects developed by hundreds and thousands of programmers during many years.

* C language for embedded systems is worst than good macro-assembler with appropriate library.

* Dangerous “reuse” of the programs.

* Difficult to debug from the very beginning to the very end.

* No real portability due to a lot of particularities of each implementation.

* Structural and modular programming in C seems more fiction than reality.

* Complicated, unusual and annoying operators and group of operators, e.g. !, =, +=, *=, etc.

* Strange notation of “logical” and “bitwise” operators.

* No rotations of data which are very useful for embedded designs, e.g. left and right rotations with and without “additional” bit of “carry/borrow”.

* The C language does not follow the syntax of some good (macro) assembles and high level languages (e.g. FORTRAN) using abbreviations as LT (less than) or lt, GT (greater than), or gt, EQ (Equal), LTE (Less Than or Equal) etc. for comparison purposes.

* Using exclamation mark “!” as a logical operator is disturbing because usually that sign is found on the end of the sentence and for different purpose.

* The undefined behavior of the language C in many cases is making the programs insecure and not portable.

* The designers of the C-language were clearly neglected the cognitive difficulties of the learners.

* Unclear, irrelevant, inconsistent terminology at level of basic terms as names, keywords or the basic data types

* Lack of useful features and duplication of not useful (three “loop structures”, two unconditional jumps (“goto” and “break”), etc.

* Not explicitly defined and justified naming rules for the functions, macros, key words, libraries etc. Inconsistent and inefficient naming rules in the libraries, difficulties to decode name of functions, e.g. rand, srand, atoi, etc.

* The non-symmetry between the printf and scanf is disturbing, e.g. there is an “ampersand” in scanf. There is no “&” in printf, e.g.

scanf (“%d/%d/%d”, &month, &day, &year);

* Symbols as “-->” is found dangerous because easy could be mistyped as -- or - or > and this is difficult to discover in some cases. When typing the program in MSWORD these symbols could be automatically altered;

THE CHAIN OF DIGITAL PROGRAMMING

The usual simplified way of the digital programming chain is:

- * Solution of the task in human language (textual and sometimes graphical solution).
- * Solution in one or more pseudo-code(s) (language(s) close to the human language and mathematics).
- * Solution in high or middle level language programming language.
- * Translation into assembler.
- * Translation into machine code.
- * Executions and testing the code.

It is known that the strength of the chain depends on the straight of the weakest stage. When a C-language or C-like language is used it is the weakest, the most controversial and most expensive stage in the chain of digital programming and maintaining the software.

The following negative effects are observed in the classroom compared to the other academic subjects and activities: increasing psychological tension, conflicting situations, increasing uncertainty, low speed of advancement in the language.

HISTORICAL BACKGROUND OF C

May be the reasons for the “minimalisms” and the drawbacks of C could be found in the technology at the time of creation of its creation. But these reasons seem not applicable for languages as BASIC, FORTRAN, PL/I, etc., created approximately at the same time.

C language was developed firstly for PDP-7, PDP-8 and similar minicomputers. The input/output systems during the PDP7/PDP-11 era included front panel interface with switches and electrical bulbs, tele-printer, a paper type reader, paper type puncher, punched card readers, magnetic type unit, etc. All of them were slow peripheral devices.

The tele-printer also called “teletype writer” or “teletype” was the principal and most sophisticated interface with the operator and the programmer. Very roughly speaking the “teletype” is combination between a “hard keyboard” and “very slow printer” which is exchanging information with computer (e.g. PDP-7/PDP-11, etc.) at speed around 50-110

bauds per second. At that and similar speed and when the programming was very rare “coding” activity (up to 1973) and the computer memory was very expensive there were some reasons to create over-compacted and oversimplified computer languages with strange syntax for relatively small computer programs, e.g. very rarely larger than 10K lines of code.

The resources at that time were very restricted. The typical twelve-bit PDP-8 had only 4 KWords of memory. The size of the addressing space of 16-bit PDP-11 was only 64 Kbytes and was used partially in most of the cases (e.g. only 32 Kbytes).

Today a lot of 16-bits microcontrollers had more than one megabyte of addressing space and the internal memory could be more than 64 KB RAM and more than 256 Kbytes memory for the program, e.g. “flash”. The resources of the modern “personal computers” are much higher, e.g. one gigabyte of memory is a common case. The today interfaces are more or less with graphical capabilities. Keyboards and printers are much faster than the teletypes. A modern printer is with one or more embedded computers more powerful than PDP7/PDP11.

Under these circumstances and when the “integrated development environments” are helping a lot it is not necessary any more to use C and C-like languages with their cryptic notations. Modern assemblers have commands with processing power comparable to the old “standard libraries” (Intel, Zilog, Motorola, Texas Instruments, etc.) The hardware has evolved a lot. The software should evaluate too and become closer to the human language and to the standard mathematical notations.

The “evolution” of C into other languages as C++ and C# is showing continuous complication and making the terminology more unclear and ambiguous. The popularity of these and similar languages is embarrassing.

THE EARLY EIGHT BIT MICROPROCESSORS

The introduced in the period 1973-1976 eight bits microprocessors as 8085, Z80, 6502, 6809, etc had shown level of performances comparable or higher than the PDP7/8

minicomputers. They had addressing space of 64 Kbytes and some of them were with additional input/output space (8080, 8085, Z80, etc). The assembler of these microprocessors included software interrupts (6800/9), loop instructions (Z80), several rotations and shifts (6502, 6809, etc), powerful hardware interrupt system (all of these microprocessors and especially 8080/85 and Z80) and many more features which outclassed the basic ideas incorporated into C-language.

That is why the first high level languages chosen for the 8-bit and 16-bit microprocessors at that time were based on BASIC, Pascal, PL/I and not on C-language. Especially popular were MPL, PLM-80 and PLM-86. Although underdeveloped from the nowadays point of view they were better and easier to use than C.

C AND THE CRYPTOGRAPHY

Although the digital programming and the cryptography seems to be similar activities they have different implementations and goals. The digital programming is intended to code rapidly, comprehensible and cheaply problems and solutions formulated into human language. The produced computer programs must be comprehensible.

The computer programs must be decoded rapidly by humans, reused and maintained cheaply from professional and non professionals. They are valuable asset for the organizations and individuals. Writing a program is kind of “technical writing” and should be comprehensible.

If the thoughts above are accepted the programming in C-like languages is more cryptography than programming because the coding and decoding at that language is difficult and it is taking more efforts and resources that necessary for that activity with other languages.

TECHNICAL, LINGUISTIC AND MATHEMATICAL ABSTRACTIONISM

Although not formally defined the “technical abstractionism” became especially evident with the creation of the binary digital computer and oversimplified sampling theorems and similar theoretical concepts. The

abstractionism in the case of C and similar languages could be characterized with:

- * It is a strong deviation from the reality, especially the reality of the mathematical notations and human language”. If we make straight line “human language – pseudo code based on the human language” the language C is far from that line. Other languages e.g. Algol-60 and Pascal are much closer to it.

- * There is a “non-objectivism” and “non representational logic and notations”. In that case that means that the algorithmic concepts from the human language are not represented in clear and justified manner when written in C.

- * Introducing the “object technology” in languages similar to C made the situation more difficult to understand and obscure. Teaching and learning these languages became less productive.

In fact the “minimalism of C and C-languages” is a technical, linguistic and mathematical abstractionism due to the following reasons:

- * The notations and terminology is too far from the human language, e.g. from “Standard English”, „Academic English”, “Business English”, “Simple English”, „Basic English” and similar more or less justifiable linguistic concepts.

- * The notation of C is too far from the standard mathematical notations and the straightforward mathematical logic. Also, the notation of C is too far from the technical and technology level established after approximately 1980. The persistent oversimplistic approach in digital computing found not only in C and similar languages is very disturbing because could be considered as degradation of the programming art and science.

The code of programs of C and C-like languages looks like pieces of the abstract art and cryptic science. Unfortunately they must be read, studied, corrected, reused and maintained by people during many years.

INCONSISTENCIES IN THE NOTATIONS

As mentioned above:

Using symbol “#” before the word “define” and not using “;” at the end of the statement is more disturbing than useful.

Group of “define statement” could be separated with appropriate couple of key words, e.g. “begin... end” or “start... stop”.

Starting with underscore is also perturbation to the reading of the program.

Maintaining maximum symmetry of the notations is considered “useful” and the asymmetry is considered “blocking”.

INCONSISTENCY TO THE CONCEPTION OF “MINIMALISM”

It was discussed before that the “minimalism” of the key words is inconsistent because in fact “switch – case” could be replaced with “if-else” statements.

*The set of key words are in very short, inefficient, unjustified and with “duplication of the functions”: e.g. there are three loop structure (“for”, “do” and “while”), the key word “register”, etc. A loop structure could be organized with “if – then – else” statement and consequently there is no need of “for”, “do” and “while” structures. The existence of three “loop structures” contradicts to the “minimalism” and is disturbing.

“Goto” and “break” could be replaced with a appropriate “unconditional jump”.

INCONSISTENCY TO THE EDUCATIONAL PROCESS

In the world of the digital computers everything must be defined exactly and explicitly because the computers could execute only the exact orders in the specified what, under those circumstances the “undefined and unclear behavior of C” is not acceptable.

C is different from assemblers and macro assembler. Also C is different from Basic, Pascal, FORTRAN, etc. There is degradation from the ideas found in Pascal.

There is a continuous destruction of the learning process with the strange and unpredictable behavior of the C-programming.

Badly organized Help menus of the software products, e.g. integrated development environments, compilers are complicating the educational process.

There are voluminous incomprehensible new standards, treating C and similar languages.

The efficiency at the “low price” or “no price (free)” development and debugging tools is more a myth than a reality, because the inconsistency of the languages is asking for more efforts, more attention and making the reuse and the portability more difficult. In fact programming in C is very expensive activities evaluated in long term.

It is observed inconsistent naming rules in C and associated libraries. The first word is frequently neglected and this is creating difficult to decode abbreviations of the function in the library, e.g. fflush(),

Comparing the explications of “indexed and indirect” addressing modes with the explication of “pointers” is not favorable for C. Frequently C is learned simultaneously with one or more assemblers and the differences in the logic of utility of these languages is clearly seen.

UNJUSTIFIED SET OF KEY WORDS

The key words of ANSI C [1] are listed in Table 1.

Table 1. The keywords of ANSI C

auto*	double*	int*	struct*
break*	else	long	switch
case*	enum*	register*	typedef
char*	extern*	return	union
const*	float*	short*	unsigned*
continue*	for*	signed*	void*
default	goto	sizeof	volatile*
do*	if	static*	while*

The keywords with asterisk (*) have an issue, e.g.: 1/ unclear, irrelevant, not justifiable key word; 2/unnecessary duplication of function or meaning (for, do, while); 3/ irrelevance (e.g. “case”); 4/ unjustifiable abbreviation; 4/ the keyword “register” is against the idea of portability of a high level programming, etc.

CONCLUSIONS

The time of the fundamental ideas of C-like languages is gone. These languages are creating unnecessary load to the learners and professionals and are blocking the advancement in the field.

The C-language is not suitable for education purposes especially at early stages of education because it is not a straightforward language.

The C-language is not suitable for engineering and scientific purposes because of its controversial logic and disturbing particularities.

The C-language cannot be repaired because in fact there are no significant ideas in the language which deserve to be kept and developed.

The C-language is not suitable for mathematical purposes because it contradicts to the standard mathematical notations

The C-language is not suitable for research in linguistic area because it contradicts to the human language and cannot be mastered rapidly and used safely from specialist in the field. The books, listed below are not suitable for “teaching manuals”, especially for non native speakers of English.

There are many reasons for failures of the software projects. Some of the mains raisons are: 1/ Lack of programming discipline; 2/ Not formulating formal explicit restrictions in order to prevent multiple; 3/ Error provoking aptitude from the programming language; 4/ Illogical and inconsistent naming rules. All these drawbacks are found in C.

NOTES ON THE REFERENCES

Printed books are listed below. They are developed from international educational organizations or well known authors for newcomers in the field. These books have different styles and values and are valuable source of information. But they have common drawbacks which make the unsuitable for teaching materials, e.g. they do not include:

- 1/ comprehensible glossary of C-terms,
- 2/ comprehensible list of abbreviations,
- 3/ comprehensible list of the used functions and modules
- 4/ justification of the methods, symbols and keywords used in the programming language.

The readers are taught by incomprehensible examples, especially difficult for non native speakers of English. The books seem not to be good tutorials for beginners and for the programmers from other languages as Basic, FORTRAN and Pascal.

REFERENCE

- [1] Brian W. Kernighan, Dennis M. Ritchie, The C Programming language, First edition 1978, second edition 1988. Prentice Hall.
- [2] Greg Perry. C by Example. Pearson Education, Inc./Que publishing. 1999 (SoftPress OOD, Bulgaria, 2005). ISBN 954-685-331-3.
- [3] Jim Buyens. Faster Smarter Beginning Programming, 2005, Microsoft Corporation. ISBN 954-685-327-5 (Soft Press, Bulgaria, 2005)
- [4] Pooja Bermbey, Kuljit Kaur. Microsoft Visual Basic .NET Professional Projects, Premier Press, 2002, ISBN 954-8396-11-4 (Duo Design, Bulgaria, 2003)
- [5] Michael Halvorson. Microsoft Visual Basic Professional 6.0 Step by step. Vol. I and Vol. II. Microsoft Corporation. 2000 (SoftPress, Bulgaria, 2000). ISBN 954-685-122-1.
- [6] Hugh E. Williams, Dave Lane. Web Database Application with PHP&MySQL. O Reilly & Associates, Inc. 2002(ZeST Publishing House, Ltd. Bulgaria. 2003). ISBN 954-91165-6-5.
- [7] Herbert Schild, Java 2 A Beginners Guide., 2001, The McGraw-Hill Companies. ISBN 978-954-685-172-7 (Soft Press, Bulgaria, 2009)
- [8] Herbert Schild, Teach your self C, Fourth Edition, 1997, Osborne/The McGraw-Hill Companies., ISBN 978-954-685-168-0 (Soft Press, Bulgaria, 2007)
- [9] Herbert Schild, Teach your self C++, Third Edition, 2000, The McGraw-Hill Companies. ISBN 978-954-685-685-7 (Soft Press, Bulgaria, 2009)
- [10] Scott Meyers., More Effective C++: 35 New Ways to Improve Your Programs and Designs. Addison Wesley Professional, 1996, ISBN 954-9341-03-8 (Zest Press Publishing House Bulgaria, 2004)
- [11] Gerhard Willms, C++, First Edition, Data Becker GmbH 2000 (Edition in French Micro Application, France, 2000), ISBN 2-7429-1530-3.
- [12] Chuck Sphar. Learn Microsoft Visual C++ 6.0 Now. Vol I and Vol. 2. Microsoft Corporation. (Soft Press, Bulgaria, 2000) ISBN 954-685-118-3.
- [13] Geetanjaly Arora, B. Aiaswarny, N. Pandey. Microsoft C# Professional Projects, Premier Press, 2002, ISBN 954-8396-10-6 (Duo Design, Bulgaria, 2003)
- [14] Dick Steflik, Prashart Sridharan. Advanced Java Networking, Second Edition, 1997/2000, Prentice Hall PTR, ISBN: 0-13-084466-7 (IfoDar, Bulgaria, 2001)

- [15] Java, SP Interface Bulgaria OOD Publishing House, 2000 (In Bulgarian only) www.bgbook.dir.bg
- [16] Make Morgan, Java 2 for professional programmers, Volume I and II, Sams Publishing, 1999 (InfoDAR Ltd, Bulgaria, 1999). ISBN-954-8485-65-6.
- [17] Sripriya, Sai Kishore. Microsoft Visual C++. NET Professional projects. Premier Press 2002 (Duo Design, Bulgaria OOD, 2003) ISBN 954-8396-07-6.
- [18] Angshuman Chacraborri, Roopendra Jeer Sandhu, Uday Kranti, Microsoft .NET Framework Professional projects. Premier Press, Inc. 2002, (Duo Design OOD, Bulgaria, 2003) ISBN 954-8396-12-2.
- [19] Paul Hatcher, John Gosney. JavaScript Professional Projects. Premier Press (Duo Design OOD, Bulgaria, 2004). ISBN 954-8396-16-5.
- [20] Angshuman Chakraborti, R. Jeet Sandhu, U. Kranti. Microsoft .NET Framework Professional Projects. Premier Press, Inc. 2002 (Duo Design OOD, Bulgaria, 2003). ISBN 954-8396-12-2.
- [21] Sripriya, Sai Kishore. Microsoft Visual C++.NET Framework Professional Projects. Premier Press, Inc. 2002 (Duo Design OOD, Bulgaria, 2003). ISBN 954-8396-07-6.
- [22] Hersh Bhasin. Microsoft ASP.NET Professional Projects. Premier Press, Inc. 2002 (Duo Design OOD, Bulgaria, 2003). ISBN 954-8396-13-0.
- [23] Dan Ransom. Dynamic Web Form Professional Projects. Premier Press, Inc. 2002 (Duo Design OOD, Bulgaria, 2003). ISBN 954-8396-09-2.
- [24] Kent Reisdorph. Borland 4 in 21days. Vol. I and Vol II. Sams Publishing. 1988 (InfoDar OOD, Bulgaria, 1988) ISBN 954-8485-39-7.
- [25] Richard M. Stallman and the GCC Developer Community. Using the GNU Compiler Collection for GCC version 4.5.0. GNU Press, Free Software foundation. USA. 2008.