EVALUATION OF TEACHING AN ARABIC PROGRAMMING LANGUAGE (ARABLANG)

Dr. Mansoor Al-A'ali
Department of Computer Science
College of Information Technology
University of Bahrain,
P. O. Box 32038
Isa Town, Bahrain

Abstract

School students in Arab Countries find difficulty in learning computer programming in a language other than their native Arabic language. This paper reports on an investigation into the benefits of teaching an Arabic computer programming language to secondary school students in Bahrain. The paper presents a comparison study between the Arabic programming language ARABLANG (Al-A'ali 1995) and PASCAL Languages for their suitability for teaching at Arabic schools. A comparison is made and evaluated with regards to learning problems and to frequency and types of errors made by students in each language.

Keywords: Programming, ARABLANG, Programming Languages

Introduction

Computer packages are normally Latin based but are modified to process data in Arabic. All existing system software and programming languages are Latin based. We believe that if systems software (operating systems, editors, programming languages, etc.) were all Arabic based, this will give a tremendous push to the use of computers in Arab Countries and especially at the school level. As a start we should teach computer programming at schools and gradually this would lead to the wider use of computers. Another possibility is to Arabize system software to enable the building of a new generation of Arabic software, which is not based on converting Latin based packages and tools. To teach programming to Arabic students at schools using a Latin based language requires them to know at least the meaning of the language commands before programming and understand the error messages.[5] In general, this is not an easy task for those whose native language is not English; and it is more difficult for Arabic school students since all teaching at schools is done in Arabic.

There have been some attempts to design Arabic Programming languages. Unfortunately, most of them are not fully developed and are not commercially available. Most of these reported languages are not suitable for teaching school students and some of them are limited to specific computer models[5]. Arabic BASIC, was designed for specific computer systems such as Commodore, Sinclair, and Sakher. ARABFORT[6], enables programmers to write FORTRAN programs in Arabic but was never fully developed and only accepts the source code in Arabic. The commands of the language are direct translation of some commands in FORTRAN. This language is not suitable for school students for two main reasons: (1) FORTRAN itself is not designed mainly for teaching programming, (2) Compile-time and run-time errors are not handled in Arabic in the implementation of the language. Al-Qaul[7] is a Lisp like logic programming language designed for Macintosh, which has been withdrawn from the market.

Al-Risalh[2] is an Arabic pure object-oriented programming language that has the basic mechanisms of object-orientation, objects, classes, and messages[2]. The author recommends that this language can used to teach Arabic-speaking students how to program and how to understand the basic concepts to the

idea of object-oriented. No research is published about its suitability for teaching school students.

Mulspren[1] is a visual programming environment for children. Unlike many conventional visual programming environments, Mulspren users program using two languages: an English-like language and a conventional programming language. The authors believe that showing multiple representations of a program, combined with good program visualisation support, will help children create a good mental model of conventional programming constructs. This model may be helpful later in life when they have to modify programs written in conventional languages, for example Visual Basic or Microsoft Word macros. Mulspren is not an Arabic programming language.

Design of the Language ARABLANG

It is impractical and impossible to find a language that is ideally suited to all situations for all users, for all applications, and for all computers. When designing a language, it is appropriate to identify broad classes of languages and create the language such that its design decisions are optimized for one of these classes. To classify programming languages, we have to identify the key concepts according to which classification may take place. These key concepts can be summarized in the following points: The language paradigm, application area, users, computers, and processing mode.

Regarding the language paradigm, languages can be classified as: Imperative Programming Languages, Functional Programming Languages, Logic Programming Languages, and Object-Oriented programming languages. Examples of these languages are: C or PASCAL, LISP or ML(10), Prolog(11), C++ or small talk.

Application areas can be broadly classified as: business applications, scientific applications, and teaching. Users can be classified as: professional programmers, non-programmers, and trainees. Computers can be broadly classified as: Microcomputers, mini-computers, and mainframes. Processing mode can be: sequential processing or parallel processing.

The design of a programming language may be based on the following design principles: Abstraction, Automation, Defense in Depth, Information Hiding, Orthogonality, Portability, Regularity, Security, Simplicity, Efficiency, Cost and Exception Handling. The problems that face the designer of a programming language are: (1) It is difficult to define an exhaustive list of all design principles, (2) For the most part there are (as yet) no quantitative measures of the properties of languages, (3) The design principles are not completely independent and some of them are contradictory to each other.

The programming language may support exception handling by allowing its programs to be able to intercept run-time errors and other unusual conditions, take corrective measures, and continue.

Other design principles may include: Labeling (Do not require the user to know the absolute position of an item in a list. Instead, associate labels with any position that must be referenced elsewhere), Preservation of information (The Language should allow the representation of information that the user might know and that the compiler might need), Syntactic consistency (similar things should look similar, different things different), Modeling (The language should provide types and operations that are suitable for modeling objects in the specific application area for which it is designed), Reliability of programs (The language should support the feature of its programs being reliable.

To start with, some design decisions have been taken. These decisions are: (1) The language should be a high-level one, (2) The language is of the imperative paradigm, (3) The language is designed mainly for teaching purposes, (4) Most users are trainees (or beginners in the field of

computer science), (5) It is supposed to be implemented mainly on IBM-PC compatibles, (6) The mode of processing is sequential. According to these decisions, some other design principles emerged; these principles are: (1) Simplicity, (2) limited amount of abstraction, (3) Portability, (4) Regularity, (5) Efficiency.

Simplicity, means that the language is easy to learn and easy to use. In order to make the language easy to learn, readability should be enhanced. In order to make the language easy to use, write-ability should be enhanced. These factors oppose each other. So, a compromise between both factors becomes necessary.

The overall support for abstraction is clearly an important factor in the write-ability of the language. This again affects readability. As a compromise, it was decided to limit the abstraction to procedures and functions.

To achieve portability, it was decided to use the popular and cheap Arabic interface tool known as Nafitha and later this was transferred to Arabic Windows. Also the compiler may be written in a language available on PC-Compatibles like C. YACC was used to produce the compiler for ARABLANG.

Regularity can be achieved by applying the following rules: (1) Each executable statement, constant declaration, and variable type declaration should end with a ';' (2) Each section should have a starting title. (3) The main ends with a special key word for termination in a similar way that a routine ends.

The design of the syntax of the language emphasizes efficiency of implementation (both during translation and during execution). This is achieved by applying the following rules: (1) No backtracking is required when parsing, (2) Extensive use of delimiters for each construct, (3) Use of a proper internal form, which can be executed directly, (4) No left recursion is allowed, (5) There are no useless productions, (6) The grammar is not ambiguous, (7) No dynamic storage allocation, (8) Passing of parameters can take place only by value and by reference, (9) A limited set of data structures is made available, (10) Using the concept of reserved words, (11) Emphasizing the use of blanks as a delimiter, (12) No run-time optimization is required, (13) No dynamic type checking, (14) No dynamic binding.

It should be noted that efficiency affects the cost of translation and running of the programs.

Brief Description of ARABLANG

In the following sub-sections, we give a brief description of the constructs of the language. The complete syntax is given in appendix A.

Declarations

This includes two different declarations: *constants declaration* and *variables declaration.*

Constants Declaration

It starts with the word "ثوابت " -means *constants*-, and then all the constants that will be used in the program must be declared, e.g.,

```
constants                    ثوابت
t = 3.4 ;                    ط=3,4؛
num = 18 ;                   عدد = 18؛
```

In this example the identifier name "عدد " is a synonym for a constant number 18.

Variables Declaration

It starts with the word " متغيرات" which means *variables*, and then all the variables that will be used in the program must be declared, e.g.,

```
variables                    متغيرات
integer : num, g;            صحيح : رقم، ج ؛
```

This example includes declaration of two variables, "رقم " and " ج".

Constants and variables declarations can be written in any order, and each declaration statement must be followed by a semicolon.

Main Program

An ARABLANG program starts with the declarations part followed by the main program part that contains the executable statements, which are executed in the sequence they are listed. The list of statements must be prefaced by the reserved word "**البداية**" -means *begin*- and followed by the reserved word "**النهاية**" -means *end* A semicolon must terminate each statement.

TheBegin	**البداية**
statements	**جمل**
TheEnd	**النهاية**

A statement can be any of the following: assignment statement, selection statement, repetition statement, input/output statement, or a control statement.

Assignment Statement:

One of the ways to store a value in a variable is to *assign* the value to the variable by means of *assignment statement.* Such a statement takes the form:

VariableName = *Expression* (e.g. **ب = ط + 6**)

Selection Statement:

These types of statements are used to direct the flow of the program in one direction or another, based on the value of these variables. Such a statement is represented in the following format:

If x > 50 **then** n = n + 1	**اذا** س > 50 **اذن** ن = ن + 1
else m= m + 1	**والا** م = م + 1
Eif ;	**ناذا ؛**

Which means, if the variable "*س*" is less than 50, then execute the expression "*ن=ن+1* ", otherwise execute the expression "*م=م+1* ". The word "**ناذا**" means end of if-statement.

Repetition Statement:

Repetition statement is used to repeat an operation for a certain number of times. In that case, the programmer would use a "**كرر**" loop.

For num = 1 to 18	**كرر من** *رقم=1* **الى** *18*
n = n + 1;	*ن = ن + 1 ؛*
Efor ;	**نكرر؛**

A "**بينما**" loop is used for a sequence of actions which need to be performed as long as a certain condition is true.

While increment > 15 **Do**	**بينما** *زيادة> 15* **نفذ**
total = increment + r ;	*مجموع=زيادة+ر ؛*
Ewhile ;	**نبينما ؛**

Input/Output Statement:

Input/output statements are used to transfer data from one place to another, for example, to get data from the keyboard or to display it on the screen, e.g.,

Read (n, num **);**	**اقرا (**ن،عدد**)؛**
Write (n, " number = ", num **);**	**اكتب ،** "العدد = "، عدد**)** ؛

There are other statements used to enter data from the beginning of the line, or to leave an empty line. In this case the command "**سطر**" is used for placing a new line. The command is as follows:

NewLine ;	**سطر ؛**

Subprograms

The subprograms are portions of the program; they come after the main program and perform actions under the overall command of the main program. There are two different kinds of subprograms called "**فرعي**" -like procedure in Pascal, and " **بداية** " -like function in Pascal.

Procedure *procedure_name* **(** *var* **)**	**فرعي** *اسم البرنامج الفرعي* **(** *متغيرات* **)**
definition-section	*قسم التعريفات*
Begin	**بداية**
statements	*جمل*
End	**نهاية**

Evaluation of the Design

It was mentioned earlier that for most of the design principles, there are no quantitative measures. But, a decision was made to let a small team put the design of the language according to pre-specified design principles and limitations, then let another small team evaluate the design. The evaluation process mainly concentrates on one question: How much of the design principles were achieved? Naturally, this can be broken down to a set of question as: (1) Is the simplicity principle applied in an adequate way? (2) Is there a limited amount of abstraction? (3) Did the design consider efficiency of implementation? (4) Is the compromise between readability and write-ability adequate (5) Is there any possibility for back tracking during parsing, (6) Is there any left recursion, (4) Are there any useless productions, (7) Is the grammar ambiguous. Some algorithms can be used to help in answering some of these mentioned questions.

The initial design was evaluated, and some modifications were made in order to make the answer to those questions in favour of the design. After modification, the design was considered adequate because of the following reasons: (1) Simplicity is achieved through a compromise between the factors that enhance readability and those that enhance write-ability, (2) Efficiency level is achieved in the design by ignoring the features that are well known to be inefficient in the implementation phase itself. For the other design principles, it is clear that they are taken care of in the design and/implementation strategy. In the next few paragraphs, we will discuss in detail, how we could evaluate simplicity and efficiency.

The factors that affect readability can be conceived from the existence of the following features: (1) natural statement formats, (2) structured statements, (3) liberal use of keywords and noise words, (4) provision for embedded comments, (5) unrestricted length identifiers, (6) mnemonic operator symbols. (7) free-field format, (8) complete data declarations, (9) syntactic differences reflect underlying semantic differences.

The features that affect write-ability are: (1) use of concise and regular syntactic structures, (2) Implicit syntactic conventions that allow declarations and operations to be left unspecified, (3) No or little use of noise words, (4) limited length identifiers, (5) simple statement formats, (6) use of symbols rather than mnemonic codes, (7) Elimination of redundancy.

The features that are well known to be inefficient in implementation are: (1) left recursion in the syntax descriptions leads sometimes to endless loops, (2) The existence of a lot of alternatives gives chance for back tracking, (3) little use of delimiters give chance for back tracking, (4) The existence of useless productions increases the parsing time, (5) An ambiguous grammar can not be translated in a unique form, (6) Dynamic storage allocation is inefficient, (7) If there were no reserved words then the parsing will be inefficient, (8) Handling huge set of data structures require time. Other features that are not included in the design, but can be considered during implementation are: (1) Pass of parameters other than value or reference can be inefficient, (2) Dynamic type checking, (3) Dynamic binding, (4) Optimization techniques require a lot of time.

Normally people prefer to write programs in their own language especially if their education is not based on English. Programming is one of the aspects of computers, which until now must be English because all programming languages use English words and symbols. In order to enable the Arabic users to write programs in Arabic, it is required to develop a new Arabic programming language. But this is a complex task. The easiest way to overcome this difficulty is to develop an Arabic programming language, which resembles one of the existing programming languages, except that it uses Arabic words and symbols. But any programming language needs a compiler. Instead of developing a compiler for the proposed Arabic programming language, one can make use of the existing compilers by building a system to convert the Arabic program to the corresponding Latin programming language. Hence, the user can use the existing compilers to compile the converted programs. There are a limited number of Arabic programming languages that follow this approach, e.g., ARABFORT . But this approach has a number of drawbacks. Firstly, it always depends on the current version of the compiler on which it was based. Secondly, all compiler and operating system messages will appear in English and this may defeat the original objective. Thirdly, the problem of interfacing with other programming languages is yet to be solved. Finally, the problem of programming language standards is not considered.

Building a new Arabic programming language is the best approach. This would involve all the normal procedures used in building a Latin based programming language.

We believe that in developing an Arabic programming language the following proposed criteria must be considered, in addition to all the best characteristics provided by a high-level programming language.

Proposed criteria for evaluation programming languages:

1. Source code written in Arabic syntax.
2. Compiler and linker messages in Arabic.
3. Run-time messages in Arabic.
4. Compiling is done directly from Arabic source code to machine code.
5. Arabic operating system commands for compiling, linking and running the programs.
6. Arabic editing facility.
7. Debugging facility with messages in Arabic.
8. Interfacing with other Latin and Arabic packages including programming languages.

Teaching Pascal and ARABLANG

Permission was obtained from the Ministry of Education, Bahrain and from Khawla Secondary School for Girls where teaching was conducted. The school arranged for us the place, the timings and the groups (senior secondary school students). The computer instructor provided the lab.

Subject

Senior students, aged 16 to 17 were chosen as participants in the research for three main reasons:

(a) Study and career decisions made by students at this stage are important.
(b) The secondary school environment makes it possible to sample students who might have access and experience to computers
(c) Senior school students are more close to the university students in age.

Procedure

Before we began our work we have chosen a sample of thirty students and divided them randomly into two equal groups, the first group was taught Pascal, and the second group was taught ARABLANG language. Each student in both groups was given a single copy of the booklet related to her group. The period of learning both languages was seven weeks, two hours weekly for each group.

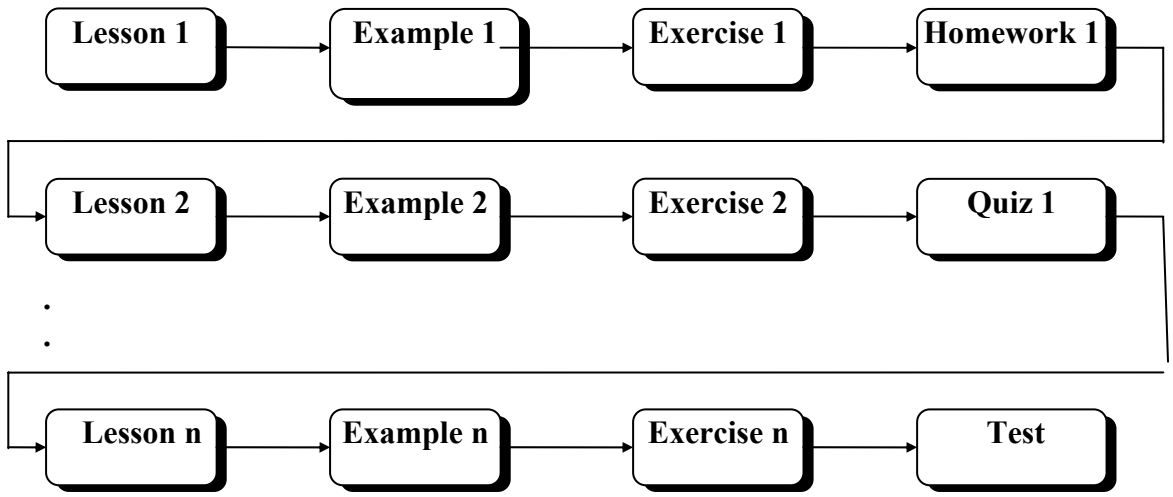

Figure 1

The above figure shows the teaching process for Pascal and ARABLANG.

As shown in figure 1, examples were provided to the students after each lesson as well as exercises. At the end of each lecture home works were given to the students, quizzes were also given. Different kinds of questions were given to the students such as multiple choice, true or false, solving problems, writing small programs. At the end of the period a final test was given to the students.

Problems

We faced the following problems during our course of teaching:

1) The students were busy in other subjects.
2) A small number of students were not interested as no credit was allocated to this course.
3) The school management refused to give us more time because the students were having exams.
4) Students had no practice in using the keyboard.
5) In some cases we used to translate in Arabic for the Pascal group although this course was suppose to be in English.

At the beginning; we wrote two booklets: the Pascal booklet and the ARABLANG booklet to use them for teaching. Each booklet consists of an introduction plus six chapters, chapter one (The symbols and operations of each language), chapter two (The assignment, read and write statements of each language), chapter three (Simple programs), chapter four (The If statement, For and while loop of each language), chapter five (Arrays) and chapter six (Files). At the end of each chapter there are exercises and problems to solve, where the answers of these exercises were provided at the end of the booklet.

Evaluation of the Teaching of ARABLANG and Pascal

During the experiment of teaching, a lot of mistakes were made by the students and, as will be explained later, each mistake was studied and a re-write of some of the ARABLANG syntax was carried out after careful consideration of each mistake. Of course Pascal was not revised, as it is not the aim of this project.

In the following two tables we show a brief discussion based on the findings of the teaching process. A compete discussion of the teaching process and its findings is available with the authors but for obvious reasons cannot be written here. The following two tables illustrate how each aspect of ARABLANG was taught and the types of errors that were made by the students. A complete list of all errors made by the students in both ARABLANG and Pascal is given in Tables 1 and 2 later in this paper.

1. The use of the words: Constant & ثوابت

ARABLANG: Constant (ثوابت) :	PASCAL: Constant :

86.60% of the students use the word 'الثوابت ' instead of ' ثوابت', because as known in Arabic language we always use 'أل' before the word if it comes at the beginning of the sentence and if it comes alone. For example:
برنامج الدرس؛
الثوابت
ت = 5 ؛
البداية
النهاية
In this condition the following error message appears: ' خطأ في العبارة'
We changed the syntax of the word 'ثوابت' to 'الثوابت' in ARABLANG
	20% of the students use the word CONSTANT as a full word instead of using CONST. For example:
	Program school ;
	constant
	a = 7;
	begin
	.
	end.
	
	In this case the following error message appears:
	'Begin expected '

2- If Statement (إذا) :

ARABLANG: If Statement (إذا)	PASCAL: If Statement

In this case students usually do not know when they should use this statement and how to use it. The students either do not understand the logic of the if statement 'اذا' or how it works.
Different examples were given to the student to test their understanding of the if statement 'اذا', and what types of mistake they usually do.
 	
- **66.66 %** of the students were using the word 'فان' instead of 'اذن'
For example ,
 	
أكتب تعليمه تقارن بين متغيرين أ و ب بحيث تكون أ أكبر من ب و تكون أ لا تساوى 10
أطبع " آهلا وسهل"
 	
الحل:
إذا (أ > ب) و (أ < > 10) فان
أكتب (" أهلا و سهلا ")
 	
In this case the student gets the error message :
'اذن متوقعة'
Since a high percentage of the students prefer to use the word 'فان' instead of 'اذن' , we changed the syntax of the word 'اذن' to 'فان' in the system, also we have changed the error message 'اذن متوقعة' to 'فان متوقعة'
 	
- **93.33 %** of the students did not understand the word 'ناذا', because it does not sound like an Arabic word, so the students suggested to use
'البداية' and 'النهاية' at the beginning and at the end of the 'اذا' statement.
 	
Here is an example of the student's solutions in the quizzes
أكتب تعليمه تقارن بين المتغير س وثابت قيمته 50 و إذا كانت قيمة المتغير
	13.33 % of the students forget to use the word then
	For example
	var
	k : real;
	begin
	.
	if K < 5
	K := K + 1 ;
	.
	End.
	In this case they get the following error message
	' Then Expected '
	
	- **73.33 %** of the students use the word do instead of then
	for example
	
	var
	a : integer;
	begin
	.
	if a > 2 do
	. . .
	End.
	In this case they got the error message
	' Then Expected '
	
	- **46.66 %** of the students forget to use begin and end if there are too many statements to be executed.
	For example:
	
	var
	g,m : integer;

س أكبر من 50 , ضيف إلى المتغير م قيمة 1 وإلا إلي المتغير ن قيمة 1
.
الحل:
إذا س > 50 فان
م = م + 1 ؛
وإلا
ن = ن + 1 ؛
	begin
	.
	if g > 7 then
	g := g + 1;
	m := M + 1;
	end.

Analysis of Programming Errors

As known in the programming world errors happen to all of us. Some of these errors happen by mistake. Because of that each language compiler reports error messages. In our research we have selected 50 of the most important error messages from the Pascal help and translated them to Arabic and implemented them on the ARABLANG compiler. Each error message was explained by example.

In our first practical class in the school we have distributed the error messages sheet to each student. 40% of the students in ARABLANG lab were asking us to help them correct their errors, 25 % of the students were using the sheets which were distributed at the lab, 15 % of the students were asking their friends for help and 20 % of the students tried by themselves to detect the errors.

Type of error	**%**
1. Using reserved word as a variable	20
2. Priority of arithmetic operation	60
3. Forget the semicolon after the statement	73.33
4. Difficulty in finding the output	20
5. Difficulty in declaring variable	26.66
6. Difficulty in writing full program	33.33
7. Using comma instead of semicolon	40
8. Using double quotation instead of single quotation	13.33
9. Forget the dot after the end	86.66
10. Forget to write the word برنامج at the beginning of the Program	13.33
11. Forget to write the name of the program	26.66
12. Use the word الثوابت instead of ثوابت	86.66
13. Use the word المتغيرات instead of متغيرات	80
14. Use ':=' instead of '=' in constant deceleration	13.33
15. Forget to use the word البداية	20
16. Forget to use the word النهاية	26.66
17. Use the word فان instead of إذن	66.66
18. Does not understand the word ناذا	93.33
19. Does not understand the word نكرر	80
20. Does not understand the word نبينما	93.33
21. Forget to use the word نفذ	46.66
22. Forget to use the left or right bracket	33.33
23. Forget to specify type of variables	13.33
24. Read identifier from constant section	20
25. do not close quotation mark	46.66

Table 1. Error messages as a result of teaching ARABLANG.

Type of error	%
1. Using reserved word as a variable	53.33
2. Priority of arithmetic operation	53.33
3. Forget the semicolon after the statement	73.33
4. Difficulty in finding the output	20
5. Difficulty in declaring Variable	46.66
6. Difficulty in writing full program	20
7. Using comma instead of semicolon	33.33
8. Using double quotation instead of single quotation	26.66
9. Confusing between Do and Then	73.33
10. Using If statement instead of While	53.33
11. Forget to use Begin	33.33
12. Forget to use End	33.33
13. Forget to write the name of the program	33.33
14. Use the word constant instead of const	20
15. Using ' := ' sign instead of ' = ' sign	26.66
16. Using the word variable instead of var	13.33
17. Using equal sign instead of colon in var section	20
18. Use the name of the program in var section	13.33
19. Forget the dot after the end	73.33
20. Forget to use Then	13.33
21. Do not know how and where to use the for loop	66.66
22. Forget to use right and left bracket	46.66
23. Use identifier to be read from constant section	26.66
24. Do not close quotation mark	46.66

Table 2. Error messages as a results of teaching Pascal.

After teaching ARABLANG and observing students difficulties and the types of error messages made, we changed the syntax of some instructions as shown in (table 3)

Before testing	After testing
ثوابت	الثوابت
متغيرات	المتغيرات
اذا علاقة منطقية اذن ناذا	اذا علاقة منطقية فان البداية النهاية
كرر أ :=رقم1 الى رقم2 نكرر	كرر أ :=رقم1 الى رقم2 البداية النهاية
بينما علاقة منطقية نفذ نبينما	بينما علاقة منطقية نفذ البداية النهاية

Table 3. Modified ARABLANG syntax based on errors made by students.

Survey of student awareness of computing and computer programming.

As shown from (table 4) 66.66 % of the students prefer to program with Arabic language

What do you think about computer technology	Easy 13%	Difficult 70	I don't know 16.6%
What is your opinion in computer programming	Easy 26.66%	Difficult 20%	I don't know 53.33
In which language do you prefer to program?	Arabic 66.66%	English 6.66%	Both 26.66
Do you know about the following languages	Basic 50	Pascal 0%	C 0%
In which level do you think that the programming should start	Interme diate 73.33%	Secondary 26.66	don't know 0%

Table – 4 General survey of student awareness of computing and computer programming.

because it is the native language. On the other hand 6.6 % of the students prefer to program using English language because it is more useful in practical life. 26.66 % do not mind if they use both languages in programming. 73.33 % of the students prefer to start programming at the intermediate level, while the remaining students prefer to start programming in secondary level.

Conclusion

Teaching students a programming language in their native language may encourage them to get into the computing field. In this paper, we presented a new Arabic programming language called ARABLANG for school children. ARABLANG has been tested in Bahrain schools to determine its effectiveness in teaching a computer programming language to Arab students. A major part of the effort was devoted towards setting up the teaching material and evaluation criteria for teaching. The language was taught to secondary school students and the results clearly show that ARABLANG is definitely a way forward for the future of Arabic programming languages. Further research is required to investigate many other aspects related to programming in Arabic, such as, the choice of mnemonics and terms, the starting level, etc.

References

1. Wright, T.; Cockburn, A. Mulspren: A Multiple Language Simulation Programming Environment (Conference Paper). Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments. IEEE Comput. Soc, Los Alamitos, CA, USA. pp. 101-3; 2002.

2. Amin, M.R., "The Arabic object-oriented programming language Al-Risalh" (Conference Paper). Proceedings ACS/IEEE International Conference on Computer Systems and Applications. IEEE Computer Soc, Los Alamitos, CA, USA. pp. 424-7; 2001.

3. Rader, C.; Cherry, G.; Brand, C.; Repenning, A.; Lewis, C. Designing mixed textual and iconic programming languages for novice users (Conference Paper). Proceedings. 1998 IEEE Symposium on Visual Languages (Cat. No.98TB100254). IEEE Comput. Soc, Los Alamitos, CA, USA. pp. 187-94; 1998.

4. Al-A`ali, M.; Hamid, M., "Design of an Arabic programming language (ARABLAN) (Journal Paper). Computer Languages. Vol. 21, Iss. 3-4, pp. 191-200; Oct.-Dec. 1995.

5. Mansoor AL-A'ali, Moheb R. Girgis, "Arabization: Actual and Objectives", 4th International conference on Multi-lingual Computing, Cambridge University, London, UK, pp.9.1.1 -9.1.10 April, 1994

6. Girgis, MR, "Writing FORTRAN Programing in Arabic Language", Journal of the institute of Maths & Computer Science (Computer Science series), Vol. 3, No. 2, 1992,pp.51-63,1992

7. VERBUM, Paradigm Software Inc., Cambridge, Massachusetts U.S.A., 1989.

Acknowledgements

To the memory of my friend and research colleague Dr. Hamid who sadly passed away just before we started this project.

Biographical Information

Dr. Mansoor Al-A'ali finished his B.Sc. in Computer Studies from the University of Teesside, UK in 1982. He received his M.Sc. in Computer Science from the University of Aston in Birmingham, UK in 1984. He received his Ph.D. from the University of Aston in Birmingham, UK in 1989. Mansoor is currently working in the Department of Computer Science at the University of Bahrain. Since 1989 he has been working as a consultant for a number of leading Bahraini organizations leading the design and quality assurance issues of major industrial computer systems. His research interests include: AI, computers in education, algorithms, Ethics and Arabization. He has over fifty refereed publications in these areas. Mansoor was the Chairman of the Department of Computer Science at the University of Bahrain from 1992 to 1995 and was the Director on Continuing Education from 1996 to 2000.

Appendix A: syntax of ARABLANG

In the table below, we show the *context-free grammar* (**CFG**) of the ARABLANG language in English and in Arabic.

program → declarations_block main_program subprograms_block	*برنامج* ← *قسم_التعريفات* *البرنامج_الرئيسي* *قسم_البرامج_الجزئية*
declarations_block → declarations_list | ϕ	*قسم_التعريفات* ← *قائمة_التعريفات* | ϕ
declarations_list → declarations | declarations declarations_list	*قائمة_التعريفات* ← *تعريفات* | *تعريفات قائمة_التعريفات*
declarations → constants_dec | variables_dec	*تعريفات* ← *تعريف_الثوابت* / *تعريف المتغيرات*
constants_dec → **constants** const_list	*تعريف_الثوابت* ← **ثوابت** *قائمة_الثوابت*
const_list → const_assign | const_assign const_list	*قائمة_الثوابت* ← *قائمة_الثابت* / *قائمة_الثابت قائمة_الثوابت*
const_assign → **id** = constant_value ؛	*قائمة_الثابت* ← **مف*** = *قيمة_الثابت*؛
constant_value → number | **" string_const "**	*قيمة_الثابت* ← *عدد* / **" رموز_منصصة "**
number → **int_num** | **real_num**	*عدد* ← **عدد_صحيح** | **عدد_حقيقي**
variables_dec → **variables** var_dec **;**	*تعريف_المتغيرات* ← **متغيرات** *قائمة_المتغيرات* ؛
var_dec → type **:** identifier_list | type **:** identifier_list **;** var_dec	*قائمة_المتغيرات* ← *نوع* **:** *قائمة_المعرفات* / *نوع* **:** *قائمة_المعرفات* ؛ *قائمة_المتغيرات*
identifier_list → identifier | identifier ، identifier_list	*قائمة_المعرفات* ← *معرف* | *معرف* ، *قائمة_المعرفات*
identifier → **id** | **id [** const **]**	*معرف* ← **مف** | **مف** [*ثابت*]
const → **id** | **int_num**	*ثابت* ← **مف** | **عدد_صحيح**
type → **integer** | **real** | **char** | **string [** const **]**	*نوع* ← **صحيح** | **حقيقي** | **رمز** | **رموز** [*ثابت*]
main_program → **TheBegin** optional_statements **TheEnd**	*البرنامج_الرئيسي* ← **البداية** *جمل_اختيارية* **النهاية**
optional_statements → statement_list | ϕ	*جمل_اختيارية* ← *قائمة_جمل* / ϕ
statement_list → statement | statement statement_list	*قائمة_جمل* ← *جملة* / *جملة قائمة_جمل*
statement → assign_statement | selection_statement | loop_statement | io_statement | control_statement | subprogram_call	*جملة* ← *جملة_يساوي* / *جملة_اختيار* / *جملة_تكرار* / *جملة_ادخال/اخراج* / *جملة_تحكم* / *مناداة_برنامج_جزئي*
assign_statement → var = simple_expression	*جملة_يساوي* ← *متغير* = *عبارة_جبرية_بسيطة*
var → **id** | **id [** simple_expression **]**	*متغير* ← **مف** | **مف** [*عبارة_جبرية_بسيطة*]
selection_statement → **if** expression **then** statement_list else_statement **eif ;**	*جملة_اختيار* ← **اذا** *عبارة_جبرية* **اذن** *قائمة_جمل* *جملة_والا* **نااذا ؛**
else_statement → **else** statement_list | ϕ	*جملة_والا* ← **والا** *قائمة_جمل* | ϕ
loop_statement → **for from** var = simple_expression **to** simple_expression optional_statements **efor ;** | **While** expression **do**	*جملة_تكرار* ← **كرر من** *متغير* = *عبارة_جبرية* **الى** *عبارة_جبرية* *جمل_اختيارية* **نكرر ؛** | **بينما** *عبارة_جبرية* **نفذ** *جمل_اختيارية* **نينما ؛**

optional_statements **ewhile ;**	
control_statement → **newline ;**	جملة_تحكم ← **سطر ؛**
io_statement → **read** read_variables **;** | **write (** parameter_list **) ;**	جملة_ادخال/اخراج ← **اقرا** *متغيرات_اقرأ* ؛ | **اكتب (** *قائمة_المعطيات* **)** ؛
read_variables → **(** var_list **)** | ϕ	*متغيرات_اقرأ* ← (*قائمة_متغيرات*) / ϕ
var_list → var | var **,** var_list	*قائمة_متغيرات* ← *متغير* / *متغير* ، *قائمة_متغيرات*
parametre_list → parameter | parameter **,** parameter_list	*قائمة_المعطيات* ← *معطى* | *قائمة_المعطيات* ، *معطى*
parameter → simple_expression | **"** **string_const** **"**	*معطى* ← *عبارة_جبرية_بسيطة* / **" رموز_منصصة "**
subprogram_call → **id** actual_parameters**;**	*مناداة_برنامج_جزئي* ← **مف** *معطيات* ؛
actual_parameters → **(** parameter_list **)** | ϕ	*معطيات* ← **(** *قائمة_المعطيات* **)** | ϕ
expression → simple_expression relop simple_expression	*عبارة_جبرية* ← *عبارة_جبرية_بسيطة* *عملية_علاقة* *عبارة_جبرية_بسيطة*
simple_expression → term | sign term | term addop simple_expression	*عبارة_جبرية_بسيطة* ← *حد* / *اشارة* *حد* / *حد* *عملية_جمع* *عبارة_جبرية_بسيطة*
sign → + | -	*اشارة* ← - | +
term → factor | factor mulop term	*حد* ← *عامل* | *عامل* *عملية_ضرب* *حد*
factor → number | var | **(** simple_expression **)** | factor ^ factor | subprogram_call	*عامل* ← *عدد* / *متغير* / **مف** *معطيات* / **(** *عبارة_جبرية_بسيطة* **)** | *عامل* ^ *عامل*
relop → = | < | > | <> | <= | >=	*عملية_علاقة* ← = | < | > | < > | = < | = > | < > =
mulop → * | /	*عملية_ضرب* ← * | /
addop → + | -	*عملية_جمع* ← + | -
subprograms_block → subprograms | ϕ	*قسم_البرامج_الجزئية* ← *برامج_جزئية* / ϕ
subprograms → subprog | subprog subprograms	*برامج_جزئية* ← *برنامج_جزئي* | *برنامج_جزئي* *برامج_جزئية*
subprog → function | procedure	*برنامج_جزئي* ← *دالة* / *برنامج_فرعي*
function → type : **function id** formal_parameters **;** declaration_block **begin** optional_statements **return** simlpe_expression **end**	*دالة* ← نوع : **دالة مف (** *قائمة_المتغيرات* **) ؛** *قسم_التعريفات* **بداية** *جمل_اختيارية* **ارجع** *عبارة_جبرية_بسيطة* **نهاية ؛**
procedure → **sub id** formal_parameters **;** declaration_block **begin** optional_statements **end**	*برنامج_فرعي* ← **فرعي مف (** *قائمة_المتغيرات* **)**؛ *قسم_التعريفات* **بداية** *جمل_اختيارية* **نهاية**
formal_parameters → **(** var_dec **)** | ϕ	

***مف**: رمز لكلمة معرف.