SindhUniv. Res. Jour. (Sci. Ser.) Vol. 50 (004) 613-620 (2018)

http://doi.org/10.26692/sujo/2018.12.0086

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Multilingual Interface for C++

F. Q. KHAN, S. M.BUHARI*, G.TSARAMIRSIS*, M.BASHERI*,M.ASHRAF**,S.JAN**

Malaysian Institute of Information Technology (UniKL MIIT),Kuala Lumpur, Malaysia

Received 11th May 2018 and Revised 26th October 2018

Abstract: Themultilingual interface for C++project aims to help students at the early stages to learn programming while bypassing the language barrier. Additionally, it can aid in the transfer of skills gained by the students to C++. The proposed interface utilizes the pre-processor commands of C++ and offers a simplified syntax and translate the keywords to natural language.The problem however is that keywords from some natural languages cannot be written in ASCII and there are multiple words for some keywords.The outcome of the experiment reveals that the presence of multiple natural language words to cater for a specific traditional programming language keyword causes confusion among novice programmers. The Syntax and keywords were formulated by a three phased workshop on school children and teachers.The multilingual interface for C++ was evaluated by a study on KG to 7thGrade children and novice programmers.The findings showed that natural language-based programming is not a complete solution for the difficulties faced by novice programming language learners. But, it could assist in elimination of certain syntax errors, while adding to the complexity of natural language constructs.

C++, Teaching Programming, Programming Language

I. INTRODUCTION

It is a fact that the language barrier makes it difficult for the students to learn programming. This is because, a student who is new to programming must learn the keywords of the programming language, which are usually written in English and the logic of programming at the same time (John andChotirat2001). Visual programming languages attempt to solve this problem but they are not sufficient.According to (Andrew *et al.*, 2004), among the most common problems affecting highly visual languages such as visual basic are "Textual programming interfaces are limited", "code behavior is difficult to explain" and invisible/hidden rules are difficult to show, understand and explain.

In a number of non-English speaking countries, the schools use programming languages with keywords in the native languages to teach the students the logic of programming instead of standard programming languages such as C++ or Java and so on. This helps the students to learn programming instead of English language. However, there are two issues with this approach.1) Not all the natural languages are coveredby these languages, 2) Transferring the skills obtained to a mainstream programming language may not bestraight forward. To address these two issues, we take a different approach. We propose the use of the C++ pre-processor and the utilization of the "define" command for the development of a natural language interface or C++. Our interface offers a simpler and easier to understand syntax, new inbuild functionality, for doing more complex tasks such as lists and writing to file. Some of the advantages of our approach are:

- *Additional natural languages can easily added*
- *The students will learn programming using keywords in native languages*
- *Supports code with statements written in different natural languages and/or C++*
- *Easier transition to C++*
- *Wide support as it is based on C++*
- *Can be used for any type of applications as it is using the C++ compiler*

Currently our approach can directly support natural languages where the selected keywords can be written in ASCII. Keywords from other natural languages must first be converted according to the way they sound and then be used in our approach. A major part of this research is to test if this approach will actually help the students or not.

The next section presents the related work. Section three presents the new programming language. Section four presents the experiment and the evaluation results and section five concludes this paper.

2. RELATED WORK

The use of natural language for programming was considered long since 1966 (Sammet,1966). Widespread teaching of computation skills in different disciplines

++Corresponding Author Email: fazal.qudus@s.unikl.edu.my

*Department of IT, Faculty of Computing and IT, King Abdulaziz University, Jeddah, Saudi Arabia

**Faculty of Electrical and Computer Engineering, University of Engineering and Technology, Peshawar, Pakistan

(Bundy,2007)has motivated the initiative towards natural language based programming. Computing has become an integral part of both theory and practice of various diversified disciplines. Thus, the medium of traditional programming languages as the best choice is questioned (Cortina, 2007)(Bell, *et al.,* 2009).(Lu, and. Fletcher, 2009). Also, the complex syntax and structure of traditional programming languages act as stumbling blocks for novice programming language learners.

Interaction of human with computers through natural languages have been studied in more depth since 1980s (Eduardo and Slamecka, 1984). Support of native languages and/or script other than English and/or Roman/Latin scripts have been studied. Authors have indicated about the non-feasibility of one-to-one mapping of mnemonic from English to Spanish, which is also considered as a highly formalized language. (Judith and Howland, 2007) studied the usage of natural language for programming, instead of traditional programming languages. Study reveals that some difficulties, like code generation from unconstrained format, are introduced by natural languages. Studies have also analyzed the impact on teaching computational thinking through programming languages. For novice programmers, design guidelines are provided to help them handle these difficulties. Three design strategies were followed: unconstrained natural language for code generation, language interpretation, language primitive set. The presence of language primitive set to construct computational rules achieved better results. Meanwhile, role of variables within a programming language have been studied using visualization of variables to novice programmers (Nianfeng et al., 2017) The motivation stated for this study is that about 26.4% computer science students at Luoyang Institute of Science and Technology failed the advanced programming course. Probable reason for this situation is due to applying programming constructs and not understanding the language. Results reveal that providing visualization of variables assisted novice programmers to design programs from a holistic point of view. Instead of writing the codes in native languages, Live Robot Programming (campusano & Fabry, 2017) indicates that difficulty of debugging codes that are deployed on robotic simulators and run. This is because the error should be mapped from the behaviour to the code segment. Live Robot programming provides a state machine representation along with visualization of the live processing of the robot and thus enabling the programmers to debug as well as rectify the code while the robot is actually working.

Electron based game development toolset has NWScript which makes coding complex. Meanwhile, participants of the workshop were able to develop the preliminaries of the game development using natural language easily (Good & Howland, 2017). Authors have also highlighted the confusion caused due to the use of unconstrained natural language for code generation. This is mainly due to when to use natural language and when to use traditional language syntax.

Researchers in (Capindale & Crawford, 1990)identified that natural language is effective for database queries, when the programmer is aware of the database contents. Visual programming, using narrations, has also been suggested and studied in the literature. Computational concepts could be well understood using natural languages but at the same time, it brings many other difficulties into coding. To make natural language based programming better, we have enhanced existing programming languages with support from different natural languages. Thus, the motive was to enhance the understanding while making sure that the transition as well as learning should be smooth and productive. The status of such a motive was tested using different groups of people, with and without any programming experience. A research challenge is how to make it easier to understand with the keywords used in the language. We assume that the learning process will be improved and quick.

3. MODIFICATION TO THE ORIGINAL SYNTAX

Our approach modified the standard C++ syntax in the following ways.

- Loop
- Cout
- Cin
- Public void main
- File operations
- List operations

The syntax of the three loops, “for”, “while”, “do” where replaced by the new command “loop” originally introduced in (Tsaramirsis,et al., 2014). The loop can be overloaded in three ways as shown below:

```
loop(number){ // do something }
```

In the above, the body will be executed a number of times without checking any condition. A hidden variable called “loopCount” can be used for accessing the current iteration.

```
loop(condition){ // do something }
```

The above is similar to C++’s while loop. The condition will be checked prior to executing the body of the loop.

```
loop(number,condition){ // do something }
```

The last version of the loop is going to execute the body, a number of times, prior of checking the condition. Similar to the “do while” loop but with a variable number of initial executions rather than just one.

The second modification to the original syntax of C++ was done to the "Cout<<" command, that is responsible for output stream. The command was simplified and replaced by the word "print" followed by the what it should be outputted (e.g. a string or the value of a variable).

```
print "hello";
```

Similarly, the "Cin>>" command was replaced by the keyword "read".

```
input myVariable;
```

The "public void main" that is the most usual starting definition for the "main" method of C++ was replaced by the keyword "start".

```
start()
{
// do something
}
```

The file operations were also simplified as it can be seen below.

```
writetofile("hello","myfile.txt");
readfromFile("myfile.txt");
```

The "writetofile" takes as parameters two variables, what needs to be stored and the destination file. On the other hand, the "readfromfile" method takes as parameter the file name and return a string.

The new syntax also includes a list class that allow users to add, find and remove elements. The list has methods for adding new nodes, removing nodes, set data in nodes, get data from nodes and printing the list. However, this part is not covered in this paper.

4. MULTILINGUAL SUPPORT

Apart from the above modification to the original syntax, a template for adding new languages was also developed and included below.

```
#define * loop
        #define * start
#define * if
                #define * else
#define * print
        #define * read
#define * string
        #define * int
#define * double
        #define * true
#define * false
        #define * new
#define * return
        #define * void
#define * break
        #define * class
#define * delete
        #define * public
#define * private
        #define * writetofile
#define * readFromFile                #define * List
#define * printlist
        #define * addnode
#define * deletenode
        #define * setdata
#define * getdata
```

New natural languages can be supported simply by replacing the * symbol with the corresponding keyword of the desired language. However, it must be ensured that each word is used only once and that no reserved keywords are used.

Table 1, shows the corresponding Urdu words for every English keyword. While the proposed approach supports keywords from any language we decide to map the keywords to their corresponding Urdu words but written in ASCII, based on the way the words are pronounced. This was done because Urdu is not naturally written using ASCII so this provides the opportunity to test if ASCII representation of non-Latin-based languages helps the novice non-English speaking programmers to perform better in programming. The experiment is explained in a following section.

Based on the translation from Table1, the "hello" word using the Urdu keywords will look like:

```
#include "mLToCPP.h" // include the head file with our definitions
shorukar() // equivalent to public void main
{
chaap "salam"; // print salam to the screen
}
```

Seting up a string in a variable "in", reading from the keyboard and assigning the input to "in" and output it to the screen:

```
#include "mLToCPP.h"
shorukar()
{
huroof in; // string in
lekoo in; // reading from keyboard, equivalent to cin>> in;
chaap in; // print in
}
```

The syntax of an "if" statement where if a variable "x" is more or equal than one the system should print "a", else it should print "b" would look like:

```
#include "mLToCPP.h"
shorukar()
{
   number x=0; // setting an integer x=0
        agar (x>=1) // if x is more or equal to 1
      chaap "a"; // print a
        warna // else
        chaap "b"; // print b
}
```

The syntax of a loop printing the numbers from 0 to 5 in different lines will be:

```
#include "mLToCPP.h"
```

```
shorukar()
{
   number x=0;
        barbar(5) // loop five times
      chaap x++ nayaline; // print x plus one and the
   new line character.
}
```

Write and reading to and from files is also very simple following simplified syntax from Section 3 and the corresponding Urdu keyword from table 1.

```
#include "mLToCPP.h"
shorukar()
{
        // write to file “myFile.txt” the word “salam”
filekoleko(“salam”,”myfile.txt”);
   // print to the screen the content of file “myfile.txt”
   chaapfilesepadho(“myfile.txt”);
}
```

Functions can also be easily defined. The following is an example of a function that returns the sum of two integers.

```
#include "mLToCPP.h"
// defining the function
number sum(number x,number y)
{
        wapaskarx+y;  // return x+y
}
shorukar()
{
// calling the function
   sum(1,1);
}
```

Classes are also fully supported by our approach.

```
#include "mLToCPP.h"
// defining the class
tabqaclassName // define a class
{
khula: // public
number a; // int a;
khufia: // private
number b; // int b;
};
```

As explained at an earlier part of this paper, it is possible to have more than one languages and even different syntax within the same code. The following code presents such example:

```
#include "mLToCPP.h"
shorukar()
{
chaap “Urdu”;
cout<< “C++”;
print “English”;
}
```

The following section describes the experiment and the results that tested if the proposed approach can help Urdu speaking students with no experience with programming or English language to learn programming. The experiment tested how well the students could understand the commands described in this section using English keywords and how well using the Urdu keywords.

5. EXPERIMENT AND RESULTS

5.1 The experiment

This study attempts to find answer for the following:

1. How effective is the usage of natural language involvement within traditional programming languages?
2. Does the usage of natural language in traditional programming languages make the learning of novice programmers easier? Like, reducing the syntactical errors.

Experiment: In our study, novice programmers were given the necessary construct of our natural language based programming language. Workshop was conducted in three phases: Phase 1 was for school teachers; Phase 2 included five school-aged children (2 female and 3 male, aged 4-12); Phase 3 was with Information Technology fourth year undergraduate students at our university. Participant to these workshops was made through contacts with the school and our department undergraduate students. Information about the workshop was presented to the potential candidates through a brief presentation and letting know about the need of such a research element.

Phase 1 teachers were with 4 to 10 years of experience from Albarka International School, Jeddah, Saudi Arabia. Their experiences varied from English, Computing, General Studies as well as Islamic studies. Even with diversified background, they were able to accomplish the requirements of the workshop based on the initial introduction given.

Phase 2 children were asked for their proficiency in English, Urdu (the natural language under consideration) and Programming. A minimal proficiency of about 10-15% in programming did not hinder much in understanding the language structures after the initial introduction to the workshop. At the same time, the absence of any programming skill caused the children unable to respond to Questions, Q1B to Q2.

For Phase 3, our target was to make sure that the students have substantial programming background and are of similar level of expertise or theoretical knowledge.

During the workshop, initially students were asked about their current educational background and some basic personal information. Candidates were made

aware that this workshop is looking forward towards their concerns or thoughts or any other difficulties they face. Three researchers were involved in this process, with one of them conducting the workshop and the other two monitoring or observing the progress of the participants. In an introduction, the newly developed natural language based programming was explained. Following the introduction, the subjects went through the following sessions:

1. Q1A: Use of basic "print" statement.
2. Q1B: Presence of conditional statement like "if".
3. Q1C1: Handling loops within the programming structure.
4. Q1C2: File Handling.
5. Q1D: Invoking Functions.
6. Q2: Keywords and syntax verification.

During Q2, subjects were provided with "cheat sheet" that contains suggested mappings between natural language and traditional programming language keywords.

At the end of the workshop, all participants, including both the subjects and the researchers, were allowed to have an open-ended discussion so as to identify the shortcomings and concerns in this research.

5.2 RESULTS AND DISCUSSION

(Table 2) summarizes the proficiency of the participants involved and their performance in the experiment. Excluding the kindergarten students, other participants were mostly successful in answering all the questions. This reveals that the presence or absence of prior programming skills did not affect their performance.

Table2: Student Performance in the Experiment

	Participants								
Criteria	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Age	7th Grade	4th Grade	4th Grade	UKG	LKG	English Teacher	English + Social + Computing	General	Islamic Studies
English Proficiency	80%	80%	80%	50%	20%	4 Years Exp	6 Years Exp	10 Years Exp	8 Years Exp
Urdu Proficiency	80%	50%	70%	20%	20%				
Programming Skill	50%	10%	30%	0%	0%				
Q1A: Print	100%	100%	100%	100%	100%	100%	100%	100%	100%
Q1B: If	100%	100%	100%	NO	NO	100%	100%	100%	100%
Q1C: Loop	100%	50%	100%	NO	NO	100%	100%	100%	100%
Q1C: File	100%	100%	100%	NO	NO	100%	100%	100%	100%
Q1D: Function	100%	100%	100%	NO	NO	100%	100%	100%	100%
Q2: Keywords and Syntax	100%	100%	100%	NO	NO	100%	100%	100%	100%
Keywords	100%	100%	100%	50%	NO	100%	100%	100%	100%

The outcome of the experiment reveals that the presence of multiple natural language words to cater for a specific traditional programming language keyword causes confusion among novice programmers. We distilled these findings and moved towards enhancing our programming language to cater for multiple keywords, a common error indicated in to perform a similar task. The support for keyword selection from IDE could solve this confusion effectively. The following indicated by could be effectively handled using an IDE:

1. Expression should be restricted during programming.
2. Different colour coding for keywords and other comments.
3. Syntactical completion using help.

As the whole workshop was paper based, the subjects did not have the opportunity to correct or rectify these mistakes with the help of an IDE. Also, it could be observed that natural language speakers are asked to use natural language, but in a different approach than the way they are used to.

Common shortcomings or errors or misconceptions in the learning of programming through natural language are:

1. Actual natural language structure impacts on natural language based programming language:
   a. Usage of pronouns or verbs, that are included in programming.

b. Combination of words with different usage of suffix or prefix.
c. Impact of redundancy in natural language on the programming language.
2. Issues due to natural language based programming language:
a. Multiple keywords that mean the same semantically. For example, using “say” instead of “print”.
b. The order of keywords.
c. Syntactical errors.

The overall rate of performing Q1A to Q1D was high for student bound children aging from 9 to 12. The keyword usage was questionable to many of the participants, both in Phase 1 and 2. This was mainly due to the order of prefix or suffix for a certain word in the natural language.

As a conclusion, it could be stated that natural language based programming is not a complete solution for the difficulties faced by novice programming language learners. But, it could assist in elimination of certain

syntax errors, while adding to the complexity of natural language constructs.

Out of 28 keywords suggested in the chosen natural language, Urdu, subjects suggested alternatives for 12 of them. Out of the 12 suggested alternatives, one of them is mainly due to the order of arrangement of the parts of the word.

Table 3: The twelve keywords for which the respondents answered differently

Original	Alternatives	
Warna	ziada	
Leeko	paro	Parhoo
Bandkar	rukna	Rako
Saafkar	khatamkar	Metana
Barnaumber	dugna	Dockhand
Datatakrasai	datahasilkaro	hasilkardadata
Barbar	duhrana	
Shorukar	agaz	
Ziyadakar	Badhaana	
Khula	Saafzahir	
Khufia	chupahowa	
Drust	saheh	

6. CONCLUSION

This paper introduced and utilized a natural language interface for C++ that was used to study the impact of writing code in natural language to the students. The proposed syntax and keywords were selected based on feedback from a focus group that was conducted during a three-phase targeted workshop. The impact of the proposed approach to the learning efficiency of programming was also tested during the workshop. Experimentation study reveals that even without prior programming knowledge, subjects were able to learn the programming using their native language. The main concern was the confusion of keywords over the selected words in the native language. As the selected language is non-Latin in nature, it causes difficulties in choosing the right alternative for the programming keywords. While the proposed approach does not solve all the learning inefficiencies, it has a positive impact to the learning process.

In the future, the interface can be extended to support more natural languages and evaluated with more complex experiments and higher number of test subjects.

APPENDIX AND THE USE OF SUPPLEMENTAL FILES

The code used in this research can be found below:

```
#define loop(...) LOOP_MACRO_CHOOSER(__VA_ARGS__)(__VA_ARGS__)
#define start int main
#define print cout <<
#define read cin >>

// write to file
void writetofile(string text,char* file){
  ofstream myfile (file);
  if (myfile.is_open())
  {
    myfile << text;
    myfile.close();
  }
  else cout << "Unable to open file";
}
// read from file
string readfromfile(char* file){
  ifstream myReadFile;
   myReadFile.open(file);
 string output;
 if (myReadFile.is_open()) {
 while (!myReadFile.eof()) {
    myReadFile >> output;
    return output;
 }
}
myReadFile.close();
return "";
}
```

The definitions of the loop can be found.

7. ACKNOWLEDGMENT

This work would not have been possible without the support of Albarka International school, Jeddah, Saudi Arabia, that allow us to perform our research work with their students and staff.

REFERENCES:

Andrew J. Ko, B. A. Myers and H. H. Aung, (2004) "Six Learning Barriers in End-User Programming Systems," 2004 IEEE Symposium on Visual Languages Human Centric Computing, Rome, 2004, 199-206.doi: 10.1109/VLHCC.2004.47

Bell, T., J. Alexander, I. Freeman, and M. Grimley, (2009) Computer science unplugged: school students doing real computing without computers, N.Z. J. Appl. Comput. Inf. Technol. 13 (1) 20–29.

Bundy,A. (2007) Computational thinking is pervasive, J. Sci. Pract. Comput. 1 (2) 67–69.

Capindale, R.A.,and R.G. Crawford,(1990) Using a natural language interface with casual users, Int. J. Man Mach. Stud. 32 (3) 341–361.

Cortina, T.J. (2007) An introduction to computer science for non-majors using principles of computation, in: ACM SIGCSE Bull., vol. 39, ACM, 218–222.

Eduardo M. S., and V. Slamecka, (1984)"Toward Native Language Software for Information Management", Information Processing and Management, Vol. 20, No. 4, 527-534.

John F., and P. Chotirat (2001) "ANN" Ratanamahatana, Brad A. Myers, Studying the language and structure in non-programmers' solutions to programming problems, International Journal of Human-Computer Studies Vol. 54, Issue 2, 237-264.

Judith G., and K. Howland, (2007) "Programming language, natural language? Supporting the diverse computational activities of novice programmers", Journal of Visual Languages and Computing archive, Vol. 39 Issue C, 78-92.

Judith G.(2017) Kate Howland, Programming language, natural language? Supporting the diverse computational activities of novice programmers, Journal of Visual Languages and Computing, Elsevier, 39 78–92,

Lu, J.J.,and G.H. Fletcher, (2009) Thinking about computational thinking, in: ACM SIGCSE Bulletin, Vol. 41, ACM, 260–264.

Miguel C., and J. Fabry,(2017) "Live Robot Programming: The language, its implementation, and robot API independence", Science of Computer Programming, 1331–19.

Nianfeng S., Z. Min, and P. Zhang, (2017) "Effects of visualizing roles of variables with animation and IDE in novice program construction", Telematics and Informatics archive, Vol. 34 Issue 5, 743-754.

Sammet, J. E. (1966) The use of English as a programming language, Commun. ACM 9 (3) 228–230.

Tsaramirsis, G., S. Al-jammoor, and S. M. Buhari, (2014). Proposing a New Hybrid Controlled Loop. International Journal of Software Engineering and Its Applications, 8(3), 203-210.