Using Computer Aided Language Software for Teaching and Self-Learning

Syed Hussain Abid
Department of Software Engineering
University of Engineering and technology, Taxila, Pakistan
hussainabid99@yahoo.com

Samana Zehra
Department of Software Engineering
University of Engineering and technology, Taxila, Pakistan
samana.zehra@uettaxila.edu.pk

Haseeb Iftikhar
Department of Software Engineering
University of Engineering and technology, Taxila, Pakistan
haseeb_virgo@yahoo.com

Abstract— In this modern age of technology advancement, the computers have played a pivotal role in bringing a revolution to all the fields including teaching and learning. Learning a new programming language is not an easy task as beginners often face problems in understanding the syntax, semantics and logic of the new language. Books are not sufficient to clear concepts and some alternate methods of learning are required. As Computer Aided Learning Software (CALS) has proven to improve the teaching and learning process, therefore, we propose software, "Programming Teaching Aid", which facilitates teachers as well as self-learners to learn a new language. It has an easy to use graphical user interface (GUI) which helps the user in understanding a pre-written code to be learnt or understood. The language being focused is C++. The proposed software truly supports the pedagogical methodologies. The use of this software will make the learning of C++ language more effective and instill self-study skills among students.

Keywords- CALS, Syntax, Semantics, Component, Teaching Aid, Graphical User Interface.

I. INTRODUCTION

In this modern era of technology, the use of computers and computer aided utilities have prevailed in all fields of our lives. Information technology (IT) is being applied in every quark of the world from silicon (sand) as computer chips to the efforts for finding the proof of Big Bang Theory that led to the creation of the Universe. The importance of IT cannot be denied and hence every area of knowledge is incorporating IT.

In most branches of engineering studies, one of the fundamental IT courses relates to C++ programming. This language forms basis for learning all advanced programming languages. A C++ course usually includes the following:

- Basics of language syntax and semantics
- Coding constructs
- Programming logic
- Using this language for solving problems specific to the student's area of study.

Learning a new language is not an easy task. The beginners often attend courses, read books and get coaching from tutors/teachers and still may not be able to understand the language basics. They often have problems in understanding the syntax and semantics. There can be numerous reasons behind this. One of the problems could be the complex teaching method or teaching supplements like teaching material, books, etc.

Educators can become more efficient by the use of suitable computer aided learning software (CALS) [1, 2]. This paper elaborates computer-based teaching aid software that will assist teachers in elaborating the concepts of the programming language C++. This language has been chosen due to its applicability in most of the engineering disciplines.

It is considered that "it takes 10 years for a novice to become an expert programmer" [3]. A novice is any learner of a programming language who lacks professional and expert skills [4]. To make it simple we will call him a beginner to programming language. A beginner may find it difficult to grasp the concepts and code constructs of a programming language to which he has never been acquainted especially if he is trying self-learning. It may be hard to understand a new language without a teacher.

Considering the above problems, we are trying to engineer a software tool which will not only help teachers in teaching the concepts of the programming language but will also be a great assistance for beginners and self-learners. A user may use this teaching aid in the following ways:

1. When the user wants to learn or understand some code, he/she must enter it in the specified location using the interactive GUI.
2. When the user submits the code, the software will generate a line-by-line description of the code at the basic-level. Basic-level detail will just describe the purpose of each statement in the given code.
3. For a detailed-level elaboration, the user will have to click at the line for which he wants to understand and requires detailed explanation.
4. On clicking on any line of the code the software will generate detailed description of all the constructs appearing in that line.
5. The software can also generate pictorial description of a line of code in the form of some regarding that concept.

II. PRACTICAL CONSIDERATION

A. *Survey conducted among teachers and students*

We conducted a survey to take a general poll from teachers and students to know what kind of problems they face in teaching and learning a programming language in the course of programming fundamentals. The survey was based on a questionnaire that contained questions such as:

- Why are most students unable to grasp the correct sense of programming even after having a course of programming fundamentals?
- Do you feel any problem in elaborating/understanding C++ concepts like loops and classes when taught on board?
- Is it not good to teach students, how to read and understand per-written codes to increase their understanding of code constructs before taking them to labs for writing programs?
- Do you think teaching how to read and how to write in sequential order will increase the yield of good programmers?
- Do you think software will be helpful to students in understanding the OOP concepts by providing an easy to use GUI, where he can insert the code to be understood and the software will generate information to be taught?

The results of this survey provided us some important factual information which formed basis of requirements needed for Programming Teaching Aid.

B. *Difficulties to novice programmers*

It has been found that beginners are limited to surface knowledge of programs and generally approach programming "line by line" rather than at the level of bigger program structures [5]. Their knowledge is context specific rather then general. They find great difficulty in understanding code written in some book due to lack of reading skills and low knowledge of all code constructs at a time.

In their recent survey, Milne and Rowe [13] outlined the difficulties of students by conducting questionnaire based survey on the web for both students and teachers. One of the results was that students thought having fewer difficulties but teachers highlighted many problems because it is true that novices fail to recognize their own deficiencies [6]. The reason is that beginners to a C++ programming language, who have no or very little experience of learning a programming language, consider that they have understood it by performing a few tasks in C++ language. This thing proves to be a big obstacle in learning a programming language.

Another potential problem for a novice is the difference between natural language and a programming language. For example: some novices have the concept that the condition in a "while" loop needs to apply continuously rather than it is tested once per iteration due to the meaning of "while" in English language.

They also find difficulty in understanding the syntax of the language due to some prior knowledge of some other language like C [7]. Sometimes, beginners find it easy to understand the problem and plan a solution but when it comes to translating those ideas on paper or on developing environment, they feel helpless. This is mainly due to the reason that they do not know where to use which construct in translating their ideas into logic. Because they don't have a general knowledge about the utility of all constructs in the programming language.

C. *Difficulties faced by teachers*

Teachers of a programming language use the classical methods of teaching by using black/white board or slide presentations to elaborate the concepts of C++. It often leads to a problem when explaining the concepts like loops. Consider the following 'for loop':

```
for(int i=0; i<10;<i++)
{
    a+=i;
}
```

To explain this loop, he first has to create a sequence of execution in his mind, then deliver the same on display (board or slide) by words or by drawing some table or by some state machine diagram to explain its complete execution. It takes him just half an hour to explain this simple loop to the students/beginners. Explaining the loops and similar constructs usually takes a lot of time.

Let us consider what happens if instead of the above mentioned traditional method, the teacher now uses a computer-based teaching aid. The teacher will give the loop as input to the teaching-aid and it is the software's responsibility to assess and generate all related information regarding loop. By this way, he will save a lot of his time and explain the loop in a better way. He will not feel any difficulty in explaining the same thing again and again as most of the work will be done by the teaching-aid.

D. *Recent methods of teaching programming*

Many different methods are employed in teaching programming. Analysis of some methods can be done in this paper.

- Statement oriented approach employs language as a set of statements but it is not true because there is always some idea behind programming language and a programming language is not equal to set of statements [8].
- Software technology oriented approach is good with students who know the phases of software life cycle, as it bounds the programmer to plan and analyze the problem first.
- Language oriented approach is good for beginners as it teaches the language first before teaching how to create logic [9].

- Sample task oriented approach is a blend of all of the above as it explains the language as well as logic using a sample task [10].

Sample task oriented approach is the best approach if employed professionally for beginners. Our work presents a teaching-aid or a tutor for a programming language where a sample task can be understood by using the detail generated by the software.

III. SYSTEM DESIGN AND IMPLEMENTATION

This section describes the software tool that has been designed and developed to achieve the above mentioned goals. This tool will enable the student/teacher to input a code snippet he/she is trying to understand or explain (in case of a teacher), and the tool will generate the intelligent description of the input code. The main architectural features of the tool are discussed as follows:

A. System architecture

User will enter the code in the input area provided in GUI. The system will then start its processing from the lexical analysis of the code and will create tokens of the code. A similar activity is done as part of compilation in compilers [11] but our software tool is not a compiler. Tokens will then be used in a number of phases of software. Tokens will be used to create intelligent line by line description of the code (Fig 3). If a user want details of a specific line in the code, he can just click on that line and the tool will generate detailed description using the Object Oriented Book by Robert Lafore [13], based on the tokens present in that line (Fig 4). Tokens will also be used to generate pictorial information regarding the code as the flow chart is generated in example. (Fig 5.) The overall system architecture is as follows Fig 1.

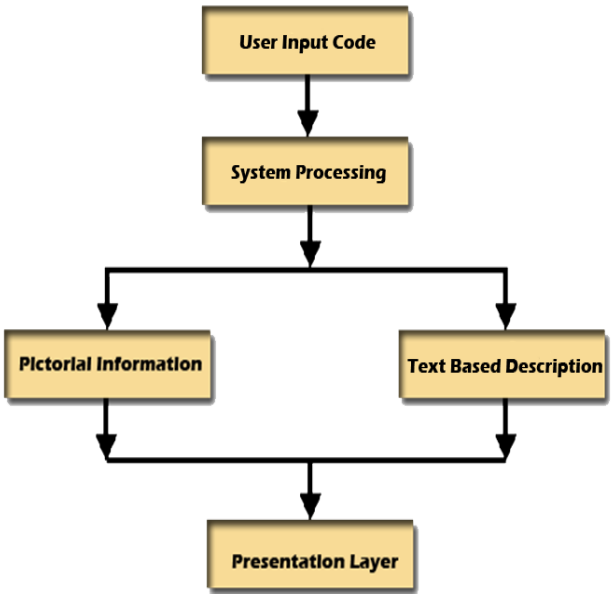

Figure 1. Complete system architecture

B. Tokenization

Tokenization is the process of breaking up a stream of text into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining. Tokenization is useful both in linguistics (where it is a form of text segmentation), and in computer science, where it forms part of lexical analysis [14].

Usually tokenizer separates tokens by doing analysis based upon whitespace characters, line breaks or some punctuation marks. They are used for doing tokenization of text in natural language (English) where there is a space after every word but same logic is inapplicable for something written in a high level language like C++. This is due to the reason that in a C++ code there is no such pattern of spaces, new line characters or punctuations. While considering a semantically and syntactically correct C++ code, we cannot reply on spaces or anything like that for tokenization. To achieve this task we have designed a tokenization algorithm. In this algorithm, we expect the next possible token after identifying one token, according to the language grammar. For example, what can be possible to appear after the keyword “int”? There can be a variable declaration, variable declaration with initialization, an array declaration, array's declaration and initialization, a function's definition etc. These are the possibilities for the keyword “int”. Following this language grammar approach, we successfully completed the tokenization of the complete C++ code. The prototype showing the tokens separated is shown in Fig. 2

C. Line-By-Line Description

To generate line-by-line description means to give one line description just like comments for every line in the code. These intelligent computer generated lines were enabled using tokens of the code with some pre-written phrases in a manner that they can form a sentence. Some scrambled phrases were pre-written which were used according to the type of token encountered and an intelligent blend of these phrase and tokens give this line wise description.

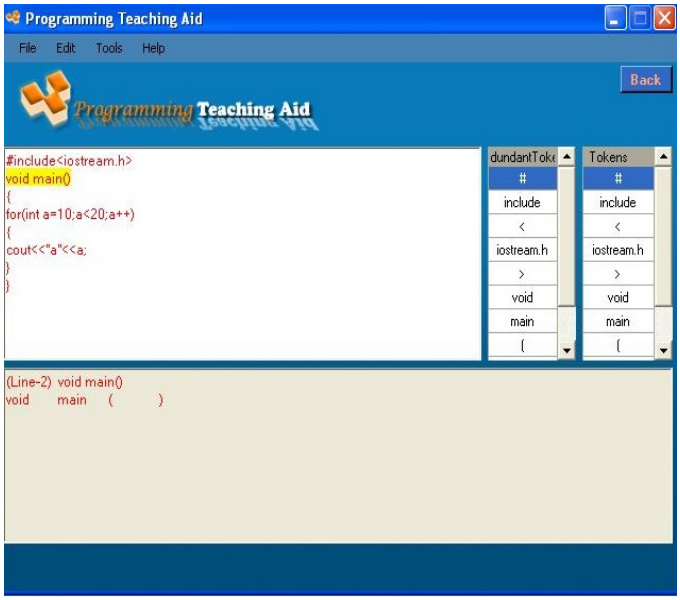

Figure 2. Original code and Tokenized code.

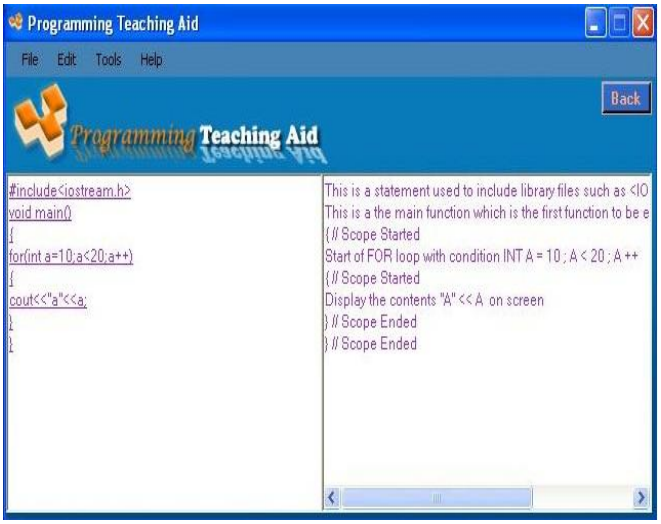

Figure 3. Line by Line description of the code generated by the software.

D. Detailed-level description

If the user is interested in detailed description of some line of code, he just has to select that line by clicking on it and the software will generate detailed-level description of that line to give its complete insight to the user. Detailed-level description will contain all the information related to every thing appearing in that line as shown in Figure 4. This detail is not generated simply by using database, it is generated by invoking some rationality in the software so that it can discriminate between for instance; a variable of some predefined data-types and a user defined data-type. This learning is invoked in order to generate the best detail of the line under consideration

E. Pictorial Description

Pictorial description involves generation of flow-chart to describe the overall flow of the code. This is also generated using tokens like in previous two stages. A diamond is for a selection statement, oval for a function call, rectangle for statement depicting a any task like initialization and connectors to show the flow of control. Token decides which shape to be used where in the flowchart.

Figure 4. Detailed description of the line selected.

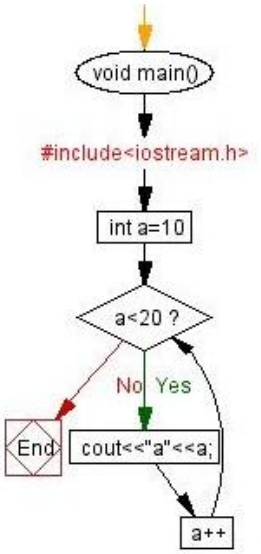

Figure 5. Flow Chart of the sample code.

IV. FUTURE WORK

We intend to develop similar application framework for other programming languages. We also intend to add some new features like animated applets and to add more intelligence in it, for example, adding a speech component to the software for explaining the input code constructs hence to make it a complete virtual tutor.

IV. CONCLUSION

Employing CALS is important in modern day teaching and learning. Our proposed software "Programming Teaching Aid" is an effective tool that will enable students in understanding the concepts of the fundamental programming language C++. If used as a teaching tool, it will save the teacher's time by giving explanation statements for code constructs and generating customized flow-chart diagrams according to the code given as input. This software can not only be used as a teaching aid by instructors but also be used by beginners and students for self-learning.

REFERENCES

[1] Muneer T. and Hawley, S. H., 1991, "Design of computer based monitoring systems for engineering education", Innovative teaching in engineering, edited by R.A. Smith Published by Ellis Horwood, West Sussex, England, 145-55.

[2] Ong, S.K. and Mannan A., 2001, "VRML Java- based approach to teaching APT programming", International Journal of Mechanical Engineering Education, Vol 29(4), 345-350.

[3]Soloway, E. & Spohrer, J. (1989). Studying the Novice Programmer, Lawrence Erlbaum Associates, Hillsdale, New Jersey. 497 p.

[4] Rist, R. (1996). Teaching Eiffel as a first language. Journal of Object-Oriented Programming, 9, pp. 30-41.

[5] Kölling, M. & Rosenberg, J. (1996). Blue - A Language for Teaching Object-Oriented Programming, Proc. of the 27th SIGCSE Technical Symposium on Computer Science Education, pp. 190-194.

[6] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, SIGCSE Bulletin, 33(4), pp. 125-180.

[7] Bonar, J. & Soloway, E. (1989). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers, in Soloway & Spohrer: Studying the Novice Programmer, pp. 325-354.

[8] Alcock, D., Illustrating BASIC! Cambridge University Press, 1977. (Hungarian translation: Ismerd meg a BASIC nyelvet! Műszaki Könyvkiadó, 1984.)

[9] Horowitz, E., Fundamentals of programming languages, Springer Verlag, 1983.(Hungarian translation: Magasszintű programnyelvek, Műszaki Könyvkiadó, 1987.)

[10] Márkusz Zs., PROLOG-ban programozni könnyű, Novotrade, 1988.

[11] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Compilers Principles ,Techniques and Tools, New Delhi, Pearson Education, 2007, ISBN: 978-81-317-2101-8.

[12] Robert Lafore, Object Oriented Programming in C++, 3rd edition.

[13] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, SIGCSE Bulletin, 33(4), pp. 125-180.

[14] Huang, C., Simon, P., Hsieh, S., & Prevot, L. (2007).Rethinking Chinese Word Segmentation: Tokenization, Character Classification, or Word break Identification.