Journal Homepage: - www.journalijar.com
INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)
Article DOI: 10.21474/IJAR01/4973
DOI URL: http://dx.doi.org/10.21474/IJAR01/4973
ISSN NO. 2320-5407

RESEARCH ARTICLE

ONLINE ALGORITHM CONVERTER FOR C++.

Dr. L.V. Patil.
Department of Information Technology Smt. Kashibai Navale College of Engineering, Pune, India.

Manuscript Info

Manuscript History

Received: 27 May 2017
Final Accepted: 29 June 2017
Published: July 2017

Key words:-
Algorithm; online, converter, web-based, Natural Language Processing, Logic.

Abstract

The first step in problem solving is to develop a foolproof algorithm for the same. This algorithm encompasses of the central logic that will be used to solve the problem. The logical steps that are followed while writing an algorithm play an integral part in the development of a coherent solution. Writing a correct algorithm ensures an efficient code which results in optimum utilization of resources. In order to develop a student's problem solving skills, the logical correctness of an algorithm is more important than the actual implementation. The aim of this paper is to develop a web-based application into which a student or novice programmer can enter an algorithm and get an equivalent C language code for the same. Various Natural Language Processing (NLP) constructs will be used for the same. The user will be able to focus on the logic of the solution without having to worry about the syntax. The application will also provide the facility of running the converted code using a GCC compiler to check for its correctness in terms of syntax and logic. It will be designed as a tool for learning the basics of problem-solving and leveraging those skills.

Introduction:-

With the growing complexity of computer science problems that are used in real-world applications, it is essential for students to learn how to develop their logic for effective problem solving first without having to focus on the syntax and constraints of programming languages. Students have difficulties in abstracting the problem from its description and breaking it down into sub-problems. There are several learning tools available online, but all of them focus on various programming languages without actually focusing on how a problem can be solved by using various techniques. The motivation behind this paper is to allow students first to be able to clear their logic, work on their problem-solving skills and then perfect the art of writing code. The aim is to create a web-based application which takes any algorithm written in plain English and converts that algorithm into a syntactically correct code. This can be done using natural language processing. Techniques like parts-of-speech recognition, tagging, labeling and tokenizing are used to capture the essence of the algorithm. The tagged keywords are then arranged in a syntactically correct format by the system, thus generating the final code. The proposed system also comes with a compiler which compiles the generated C-code and displays the output on the console.

This paper is aimed at developing the logic of students who are just introduced to various problems in Computer Science. The ease of understanding, combined with the easy to use user interface makes this application an efficient learning tool. It overcomes the various drawbacks of the pre-existing systems and hence is very useful. The flow of conventional problem-solving techniques is shown in figure 1.

Corresponding Author:- Dr. L.V. Patil.
Address:- Department of Information Technology Smt. Kashibai Navale College of Engineering, Pune, India.

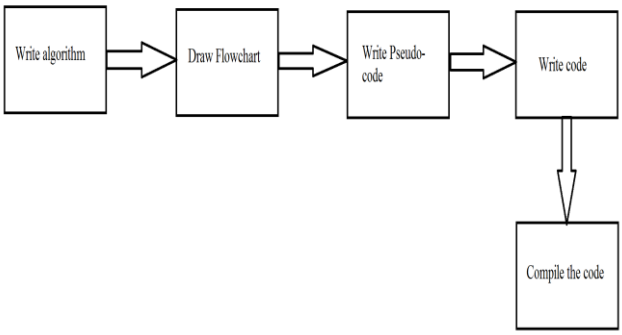

Fig. 1:- Flow diagram of conventional problem solving.

Related Works:-

Algorithm Converter:-

The Algorithm to code converter, converts the algorithm directly to a C++ code. The general technique of first writing an algorithm, then drawing a flowchart and then writing a code doesn't encourage logic development. This Converter uses advanced coding techniques to transform the algorithm into a syntactically correct code. For converting an algorithm to code firstly, the user writes his algorithm to a text document using Notepad. Here the algorithm written should follow the specification mentioned by the developers since it was developed for this converter. The algorithm file is saved by the user/student. Then the converter application is launched. The application asks for the text file name containing the algorithm which the user wishes to convert. The respective file name is entered. After doing so, the application accesses the file and converts the algorithm to code in the backend. The code is a CPP code with the name same as of the text file name containing the algorithm. The Algorithmic Converter uses Turbo C++ as the compiler to compile the generated code and show the output accordingly. Hence it requires the user to have Turbo C++ to be already installed on his machine. Turbo C++ is a Windows operating system oriented compiler that's why the application cannot run on different OS's causing inconvenience to the user. The application has to be downloaded and stored on the user machine. Hence it ends up consuming memory space. This approach is somewhat static as the guidelines to how to write the algorithms are given, and syntax is specified which becomes similar to having your own language to write codes.

Natural Language Processing:-

The entered algorithm is in plain English, and it must undergo natural language processing before it can be assembled into a syntactically correct code. In this module, the entire algorithm undergoes sentence detection to break it down into separate sentences. Each sentence represents one operation in the code. The sentences that are detected are then tokenized into individual keywords, which are then sent to the database for further processing. This process essentially breaks the algorithm down and maps it into keywords that follow the specified syntax. At the end of this process, an intermediate code is generated which consists of keywords returned from the database. This intermediate code is then processed further for code assembly. The techniques used in this phase are:

Sentence Detection:-

The Sentence Detection module is part of the Apache OpenNLP API which is a tool provided for Natural Language Processing. This module is used to separate an entire paragraph written in natural language i.e. English into individual sentences. The sentences are split by making use of various punctuation marks like ".", "?". "!". Tab and new line characters are also used to demarcate the end of a particular sentence. Thus, making use of all these conducts, the Sentence Detection module is used to separate the entire algorithm into individual sentences. This is an important step because each sentence in the algorithm represents only one operation in the code.

Tokenization:-

The Tokenization module also is part of the Apache OpenNLP API. This module is used to separate the sentences that have been detected by the Sentence Detection module into tokens. A token refers to each word present in a sentence. The tokens are separated with the help of the blank spaces that exist between each word in a sentence. This means that, if we have a sample sentence as follows, "This is a sentence", then the tokens will be as follows, "This"

"is" "a" "sentence". This Natural Language Processing step is important to store the tokens in the database and consequently map them to the syntactically correct keywords.

Code Assembly:-
The processed natural language output i.e. the intermediate code has semantically correct keywords but still has to be arranged syntactically to follow the entered algorithm. In this phase, the fragments of correct code generated in the previous step are assembled. The assembled code maintains the flow of control described in the entered algorithm. The code is assembled to remove any extra declarations, and it also defines the scope of the written program. All the initial header files and declarations previously entered by the user are also added into the program. Additionally, parenthesis matching is also done in this phase of the system. This is a crucial step which makes a great difference to the accuracy of the developed system. The output of this phase would be syntactically correct code in accordance with the entered natural language algorithm. This code is then displayed on the code frame. The student can click on the "compile" button provided below this frame to run the generated code.

Code Compilation:-
In order to test the logic behind the code, it is imperative to run the generated program and compare the output with the requirements specified before The syntactically correct code generated from the previous step is sent to the compiler residing at the server, where it is compiled to find any errors and then run. The output of this code is displayed on the console at the bottom of the page.

Database:-
The database management system used at the back-end is MongoDB. The No-SQL database provides flexibility to store the keywords in an unstructured manner. This makes accessing the database and searching for keywords effortless. There is one document defined for each keyword. The database is populated with all possible words which can be used in a particular algorithm. These words are searched for and then mapped to their specific keyword. The declaration specified by the student are used for creating a dynamic document which contains all the variables to be used in the program. This collection is flushed at the end of the conversion process. All the new codes that are generated from the algorithms are stored in a large database to improve the accuracy of the system over time.

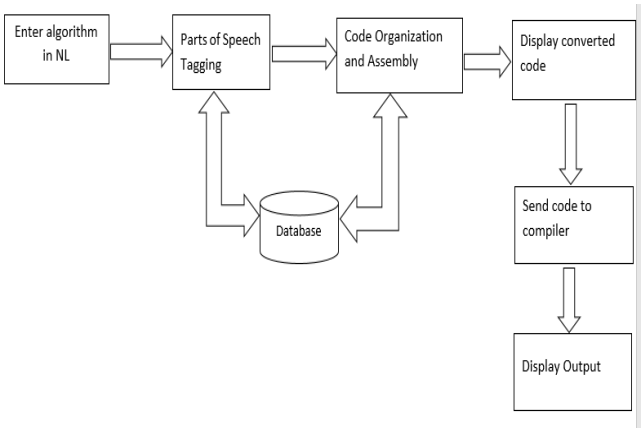

Fig. 2:- Architecture of Online Algorithm Converter

Conventions that must be followed when writing the pseudo code:
1. Write keywords of the pseudo code in lower case.
2. Each statement in pseudo code should express just one action for the computer.
3. Variable, constant and function names may be more than one word then they are joined with an underscore.
4. After and before any word in pseudo code text has only one space or spatial character.
5. Summations and counters must be initialized to zero, and other variables that require initial values must also be initialized.

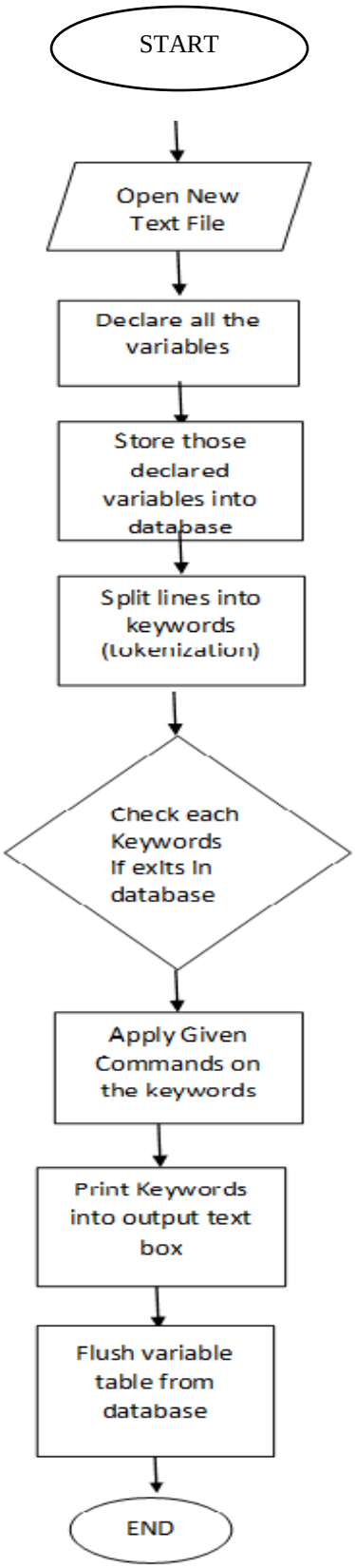

Fig. 3:- Flow chart for converting pseudo code into source code.

Steps to convert pseudo code into the source code as shown in figure 3:

1. Open New Text File
2. Declare all the variables in the declaration section.
3. Store variables those declared into the database.
4. Read one line at a time and Split lines into keywords (tokenization)
5. Check each Keyword if exits in the database then take the keyword else print the keyword in output textbox.
1. Apply Given Commands on the keywords.
2. Print keyword into output text file.
3. Flush variable table from the database.
4. Stop.

Acknowledgment:-

I would like to thanks to all the teaching and non-teaching staff members of the department and my friends who helped me in gathering information and providing encouragement to me.

References:-

1. Sharvari Nadkarni, Parth Panchmatia, Tejas Karwa, "Semi-natural language algorithm to programming language interpreter", IEEE, DOI**:** 10.1109/HMI.2016.7449190
2. Suvam Mukherji, Tamal Chakrabarti "AUTOMATIC ALGORITHM SPECIFICATION TO SOURCE CODETRANSLATION", IJCSE, ISSN: 0976-5166 Vol. 2 No. 2 Apr-May 2011
3. Nghi Truong, Peter Bancroft, Paul Roe, "A Web-Based Environment for Learning to Program", Twenty-Fifth Australasian Computer Science Conference (ACSC2003), Adelaide, Australia
4. RadaMihalcea, Hugo Liu, and Henry Lieberman. "NLP (Natural Language Processing)for NLP (Natural Language Programming)", A. Gelbukh (Ed.): CICLing 2006, LNCS 3878, pp. 319–330, 2006.C Springer-Verlag Berlin Heidelberg 2006
5. http://codershunt.weebly.com/projects/algorithm-to-code-converter (2011, July 12th)
6. Scratch, available at www.scratch.mit.edu, (2016, September 4th)
7. Snap! – Build Your Own Blocks, available at www.snap.berkeley.edu, (2016, September 6th)
8. Treehouse, available at www.teamtreehouse.org, (2016, September 6th).