SECOND
NATIONAL CONFERENCE ON ADVANCES IN COMPUTING AND INFORMATION TECHNOLOGY ISSN:2347-7385

Design and Implementation of CodGen Using NLP

Priyanka[1], Priyanka HL[1], Priyanka P[1], Ruchika[1]
1 UG students, Department of CSE, REVA ITM, Bengaluru-64.

Naveen Chandra Gowda[2]
2 Assistant Professor, School of C&IT, REVA University, Bengaluru-64. ncgowdru@gmail.com

Abstract- **Programming is an essential tool in today's world. It has its influence from agriculture to medicine in all possible fields. Learning all programming languages is highly impossible by a single individual. The time taken to write a program is also a problem under consideration. The only solution to the above problems is to automatize the process of coding. To achieve this automatization, we have designed a system that accepts an algorithm and generate the corresponding code in one particular language. We will be making use of the Natural Language Processing (NLP) techniques to achieve the task.**

Index Terms - Text Mining, Parsing, Natural Language Processing, Automatic Program Generation.

I. Introduction

Online has turned into preferred mode of communication these days. Programming is extensively used to write various software's. Learning various programming languages, remembering syntaxes and finding skilled programmers are the key challenges in this aspect. Automating the process of coding is a solution to the problem. To achieve that, we make use of Natural Language Processing (NLP) techniques and Text mining techniques.

The rest of the paper is structured as follows. Next section describes previous system that has already been built. Section III describes our proposed system. Section IV concludes the system and section V is the references we have used.

II. Related works

Of late algorithms have been applied in diverse area like Astronomy, Social Sciences and many others. In web applications like Facebook and MySpace, Graph Theory algorithms should be used. Problem lies in algorithm implementation when the programmer is not well trained. Moreover there are many programming languages and it is not easy for someone to convert programs from one language to another [3]. A critical challenge in this endeavor is to make computer analyses the natural language (Algorithm or pseudo code). Suvam and Tamal in [3] have found a solution for this by assuming the input to be in XML specification. Drawback of this is, user cannot give the pseudocode in natural language. They have to work on converting pseudocode into XML specification. In [4] Amal, Jamsheedh and Linda have used pseudocode compilation technique. The advantage of this system is user can convert program into multiple languages from one pseudocode.

III. Proposed model

Many approaches can be used to convert algorithm to program. Each approach has its own advantages and disadvantages. We make use of parsing along with few NLP techniques and dataset.
Input to our system will be a file containing algorithm (Pseudo code). Output is the name of the file containing equivalent C program.

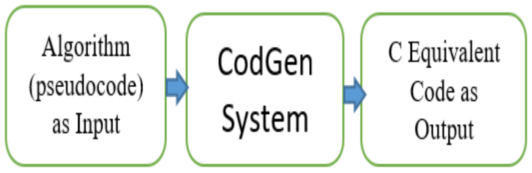

Fig 1: Proposed System

The process of converting the pseudocode-algorithm to a C equivalent program is by inspecting each and every keyword present in the individual lines of an algorithm.

1. Splitting algorithm

This module is used to extract individual lines from algorithm. If a particular line contains more than one functionality, then we split the line into multiple lines so as to obtain one functionality per line.

Example algorithm:

```
Declare b,c as floating numbers

Read b

Sum it to 3 and store the result in c

Print c
```

After splitting the above example algorithm, it looks like as shown below

```
Declare b,c as floating numbers

Read b

Sum it to 3 and

store the result in c

Print c
```

2. Variable extraction

In any program, variables has to be declared before they are used. Variables are crucial part of an algorithm. They play a major role in any program. As a result they should be carefully extracted from algorithm while converting it into program. This module does the task of extracting variables from the lines obtained in module1 (Splitting algorithm)

Consider this example algorithm:

```
Sum = a+b

Add a to b,

Store a in b,

Increment c with 1
```

Variables extracted from the above algorithm are: b and c

3. Assign data type to each variable:

This module does the task of assigning data types to extracted variables. By default integer is assigned if data type is not explicitly specified in the algorithm. A dictionary is generated for this purpose.

For the statement:

```
Declare b,c as floating
```

Following dictionary is generated, where keys represent the variables and values represent data types.

```
Dic={b:float,c:float}
```

4. Convert each line in algorithm into c code

This module converts each line of the algorithm into equivalent c code.

Consider below example algorithm:

```
Read b

Sum it to 3

Store the result in c

Print c
```

Its equivalent C code is as shown below:

```
scanf("%f",&b);

temp=b+3;

c=b+3;

printf("%f",c);
```

5. Declare the variables in c file:

This module appends extracted variables to the beginning of C file.

Variables extracted from algorithm are stored in dictionary as shown below.

```
dictionary={b:float,c:float}
```

It will be appended to the beginning of C file as:

```
float b,c;
```

6. Attach main () to C file:

Now as all of the code within the body of the function is written to C file, we need to attach the main () function's header to it.

After attaching the main () function, the code looks like this:

```
int main()

{

        float b,c;

        printf("Enter b");

        scanf("%f",&b);

        c=b+3;

        printf("%f",c);

}
```

This module appends header files to the beginning of C file based on the system functions used in C program. Below example shows the C program generated after appending header file.

```
#include<stdio.h>

int main()

{

            float b,c;

            printf("Enter b");

            scanf("%f",&b);

            c=b+3;

            printf("%f",c);

}
```

The proposed model of the system is depicted in *Fig1*.
Once after the C converted file is produced, which can be compiled and executed using the pre-loaded compiler.

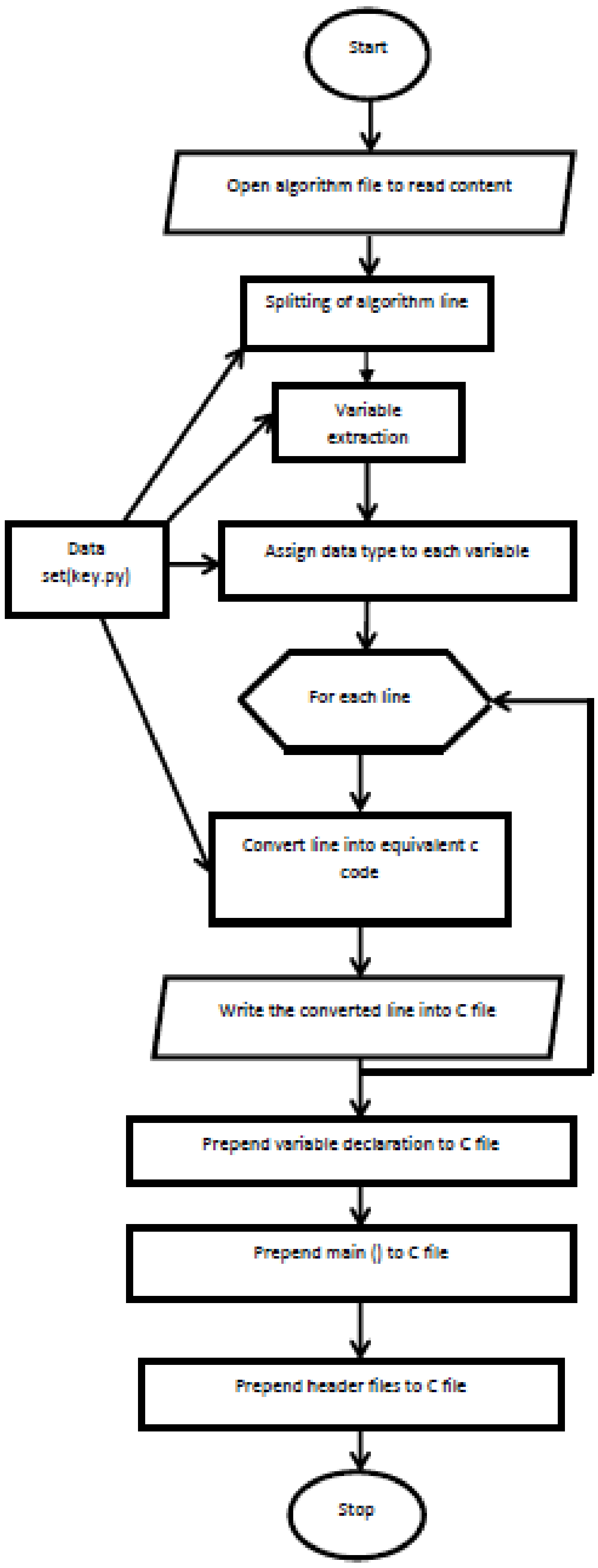
Start
Open algorithm file to read content
Splitting of algorithm line
Variable extraction
Data set(key.py)
Assign data type to each variable
For each line
Convert line into equivalent c code
Write the converted line into C file
Prepend variable declaration to C file
Prepend main () to C file
Prepend header files to C file
Stop

Fig.1. Phases of proposed System

Fig.1. Phases of proposed System

IV. Expected Output

The output of the system is expected to be a proper C program. The system is expected to convert basic algorithms like, arithmetic operation, searching and sorting algorithms into proper C program.

Consider the following algorithm to add two numbers:

```
declare b,c as integer numbers
read b and c
add b to c
store the result in sum
print sum
```

Expected C program:

```
#include<stdio.h>
int main()
{
            float b,c;
            printf("Enter b");
            scanf("%f",&b);
           c=b+3;
            printf("%f",c);
}
```

IV. Conclusion

So far we have seen the necessity of automatizing the process of coding, various approaches used for the same and how NLP and text mining techniques can be used. Though our approach too has its own advantages and disadvantages, it is giving expected outputs for the algorithms that we have tested.

V. References

[1]. Dr. Safwan Omer Hasson, Fathima Mohammed Rafie Younis, IJEIT Paper on "Automatic Pseudocode to Source Code Translation Using Neural Network Technique", University Of Mosul.
[2]. Kyle Morton and Yanzhen Qu, IJIET paper on "A Novel Framework for Math Word Problem Solving".
[3]. Suvam Mukheerjee, Tamal Chakrabarti, IJCSE paper on "Automatic algorithm Specification to Source Code Translation".
[4]. Amal M R, Jamsheedh C V and Linda Sara Mathew, IJCSITY paper on "Pseudo code to Source Programming Language Translator".