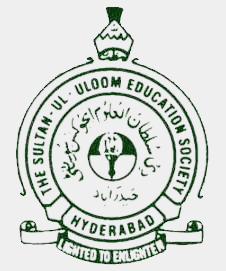
**MUFFAKHAM JAH COLLEGE OF ENGINEERING TECHNOLOGY**

(Affiliated to Osmania University)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad-34.



**DEPARTMENT OF INFORMATION TECHNOLOGY**

***CERTIFICATE***

This is to certify that the Mini Project work titled “**SudoCode**” is a bonafide work prescribed by the Osmania University for B.E III/IV Vth semester during the academic year 2017-2018 carried out by **Ayush Singh (1604-17-737-022) and Pranav Hindupur (1604-17-737-004).**

Course Coordinator Head-ITD

Mr. Riyazuddin Dr Mousmi Ajay Chaurasia

A

Mini Project Report

On

**SudoCode**

By

**Ayush Singh (1604-17-737-022)**

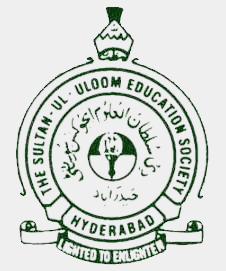
**Pranav Hindupur (1604-17-737-004)**

Of

II1/IV B.E. Sem-5 (IT-A)

Under the Guidance of

Mr. Riyazuddin



DEPARTMENT OF INFORMATION TECHNOLOGY

**MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY** (Affiliated to Osmania University)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad-34.

**ACKNOWLEDGEMENT**

The austerity and satisfaction that one gets on completing a project cannot be fulfilled without mentioning the people who made it possible with gratitude.

We are grateful to the almighty God who helped us all the way throughout the project and also has molded us into what we are today. We express our sincere thanks to our parents who encouraged us always to achieve our goals.

We offer our sincere thanks to Muffakham Jah College of Engineering and Technology for allowing us to do our mini project in their esteemed institution.

We show gratitude to **Dr. K. N. Krishnan sir, M.Tech, PhD** and **Principal** for having provided all the facilities and support. We would like to thank **Mrs. Mousmi chaurasia** (**Head of the Department, Information Technology)** for her expert guidance and encouragement at various level of project.

We are Thankful to our guide **Riyazuddin** **Sir Assistant Professor of the Department** and for his sustained inspiring Guidance and cooperation throughout the process of this project report.

We express our deep sense of gratitude and thanks to all the **Teaching** and **Non-Teaching Staff** of our college who stood with us and helped us to make it a successful venture.

**ABSTRACT**

SudoCode as the name suggests, is a supreme coding tool for everyone. Using SudoCode, one can start programming with actually knowing the coding language (in this case C or C++). This particular tool helps in converting a given pseudocode to a code in either C.

Code readability and code quality are improved to a large extent. Using this tool, our time is also saved to a large extent. It cleans the code, and makes sure the indentations are in the right manner.

PURPOSE OF THE SYSTEM:

The purpose of this tool is to help budding programmers focus on logic and save time, and let the tool bring out the code in the desired language (C in this case).

It helps bring proper indentation, and an error free final code. It improves code readability and quality.

EXISTING SYSTEM:

Here, we have a test code, i.e. the pseudocode which we run in our main file. We run this on a shell platform (on google cloud services) where on running the pseudocode in our main code, we get the output, we convert it internally to C code and Hence, give the output.

The functionalities that can be performed here are, for loop, while loop, function definition and calling etc.

PROPOSED SYSTEM:

It’s a fairly simple project, and does not involve any machine learning at this stage, but implementing this using machine language is a part of the development process.

SYSTEM CONFIGURATION:

Hardware requirements:

* Processer : Any Updated Processer
* Ram : Min 1 GB
* Hard Disk : Min 100 GB

Software requirements:

* Operating System : Windows family
* Technology : Python 3
* Web Technologies : Google Cloud Platform, notepad

**CONTENTS**

1. **Introduction**
2. **Literature survey**
3. **Software Requirement Specification**
4. **System Design**
   1. **System Architecture**
5. **Implementation**
6. **Testing**
7. **Screenshots**
8. **Conclusion**
9. **Future Enhancement**

**CHAPTER 1**

**INTRODUCTION**

SudoCode, as the name suggests, is the supreme coding tool for everyone. Using sudocode, one actually need not learn how to code in C or C++.  
This particular tool helps you with converting your pseudocode to code.  
All you need to do is follow some rules laid here and you'll be good to go. For now, this particular thing only converts pseudocode to C code. But I'll soon be expanding this to convert pseudocode to python and C++ code too.

Using this tool, our time is also saved to a large extent. It cleans the code, and makes sure the indentations are in the right manner.

It's a fairly simple project and does not involve any Machine Learning, but yes, implementing this using ML is definitely on the cards.  
But for now, it's going to be just some code written with some common sense to help achieve this.

**CHAPTER 2**

**LITERATURE SURVEY**

**LITERATURE SURVEY**

**EXISTING SYSTEM**

Here, we have a test code, i.e. the pseudocode which we run in out main file. We run this on a shell platform (on google cloud services) where on running the pseudocode in our main code, we get the output, we convert it internally to C code and Hence, give the output.

The functionalities that can be performed here are, for loop, while loop, function definition and calling etc.

**PROPOSED SYSTEM**

Here, we have a test code, i.e. the pseudocode which we run in out main file. We run this on a shell platform (on google cloud services) where on running the pseudocode in our main code, we get the output, we convert it internally to C code and Hence, give the output.

The functionalities that can be performed here are, for loop, while loop, function definition and calling etc.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**SYSTEM ANALYSIS:**

The program works fine, when the pseudocode is written in the rules specified.

* **Starting main body**

To start the code that's to be entered in the main body you need to start off with the word start

* **Initialising Variables**

Follow the following format:

initialise <optional\_type> <var\_name>=<var\_value>

EXAMPLE: (1) initialise int temp=0

(2) initialise var=100

Both the examples above are valid.  
Mentioning the type is optional. Incase you don't mention the type, the default type of the variable is float.

* **Print**

Print helps you print things on console. It could be statements or variables.  
The best part about this application is that it understands whether you want to print a string or a variable.  
The only drawback here being that, you can print only one particular variable in one print statement for now.  
Let's see how it works!

print <statement\_or\_variable to be printed>

EXAMPLE: (1) print hey there

(2) print inside loop now

(3) Let's assume there's a variable "temp" in the main body, then

print temp

The last statement says print temp. Here, the application understands that temp means the variable temp and not the string temp to be printed. It also understands whether the variable mentioned is an integer or a float value and accordingly mentions the format specifier in C and C++.

* **For Loops**

Incase you want utilise the functionality of for loops, you can do it in the following manner:

for int <var\_name>=<start\_value> to <stop\_value>

EXAMPLE: (1) for int i=0 to 10

-------

-------

endfor

(2) for int i=50 to 10

-------

-------

endfor

endfor is used at the end of the for loop. As, you can see above, you can loop from a higher to a lower value and also the other way round.  
Also, the ----- can be filled with statements or operations that you want to perform within the for loop.  
By default, the increment and decrement in the loop is 1. Sometimes you might want to increment or decrement the looping variable by a value greater than 1.  
If that's the case, don't worry, we have provision for that as well. Follow the following steps:

for int <var\_name>=<start\_value> to <stop\_value> with gap=<gap\_size>

Example: (1) for int i=0 to 60 with gap=3

-------

-------

endfor

(2) for int i=60 to 10 with gap=3

-------

-------

endfor

You need not mention a negative gap as the application understands that it's supposed to give a negative gap.

Now, an example of a loop:

for int i=10 to 1 and gap=2

print i

endfor

Here, the application understands that the program is supposed to print i as a variable and hence it print the value of i and not just i as it is.

* **While Loops**

While loops can also be generated in this application. To utlilise the feature of while loops, follow the format:

while <condition>

-----

-----

endwhile

EXAMPLE: (1) while i<=10

print i

i++

endwhile

(2) initialise int temp=110

while temp>=100

print temp

temp--

endwhile

As you can see above, endwhile is used at the end of the while loop.  
So basically, you can enter any statement or code in while loop, just that it should end with an endwhile.

* **Functions**

To define a function that has n arguments, you can follow the following format:

function <func\_name> returns <return\_type> with args <var\_type> <var\_1>, <var\_type> <var\_2>,.....,<var\_type> <var\_n>

EXAMPLE: (1) function test returns int with args int a, float b, int c

(2) function test returns void with args int size, float array\_size

Incase you don't want to return anything, then you'll have to mention void. Now to make sure you have some statements within the function, you can follow the format as mentioned for all above avaiable functionalities.  
Just make sure that you end your function with endfunction.  
An example is given below that gives you a clearer picture of how to define statements within a function.

function test returns int with args int a, int b, int c

print test func

a++

b++

c++

a = b + c

return a

endfunction

As you can see above, there's a return statement. This statement is optional.  
Incase you decide to return a value, just make sure you mention the value you want tor return.

**DURATION**

The project took 3 months to code

**CHAPTER 4**

**SOFTWARE REQUIREMENT SPECIFICATIONS**

**SOFTWARE AND HARDWARE REQUIREMENT SPECIFICATIONS**

Hardware requirements:

* Processer : Any Updated Processer
* Ram : Min 1 GB
* Hard Disk : Min 100 GB

Software requirements:

* Operating System : Windows family
* Technology : Python 3
* Web Technologies : Google Cloud Platform, notepad

**CHAPTER 5**

**SYSTEM DESIGN**

**SYSTEM DESIGN**

The program is run on an online shell, i.e. the google cloud platform. The programs, are saved on

www.github.com

These are then cloned onto the shell and executed. First the pseudocode is opened, and the executed code is displayed. We can also see the code, i.e. converted into C.

This is done as follows:

1. Clone the source with git:

$ git clone https://github.com/Pranav24021/Project1

$ cd sudocode

1. Running sudocode:

$ python3 main.py path/filename.txt

e.g. - $ python3 main.py testfiles/sudocode.txt

**CHAPTER 6**

**IMPLEMENTATION**

**SAMPLE CODE**

**sudocode.c (also there in C++ and phython)**

|  |
| --- |
| #print(\_\_name\_\_) |
|  |  |
|  | def get\_code(filename): |
|  | return\_list = [] #stores the last defined function details to get return statement accordingly. |
|  | variables = [] #Stores the list of all variables |
|  | funcs = [] #stores list of all functions and number of args to each func |
|  | func\_args = 0 #variable to store number of args in ever function |
|  | file\_ptr = open(filename, "r") |
|  | code\_file\_ptr = open(filename[0:len(filename)-4]+".c", "w") |
|  | code\_file\_ptr.write("#include <stdio.h>\n#include <stdlib.h>\n\n") |
|  | no = 0 |
|  | for line in file\_ptr: #going through every line |
|  | no+=1 #incrementing line count |
|  | line\_elem = line.split(" ") #tokenisation at space |
|  | line\_elem[-1] = line\_elem[-1].replace("\n","") #replacing new line char with null char |
|  | line\_elem[0] = line\_elem[0].lower() #converting the first word in line\_elem to all lower case letters |
|  | #print("\"",line\_elem[0],"\"") |
|  | #print("gap" in line\_elem) |
|  | line\_of\_code = "" #initialising var to get the final line of code to be inserted in file |
|  |  |
|  | if("start" in line\_elem): #start keyword starts main function |
|  | line\_of\_code +="int main()\n{\n" |
|  |  |
|  | elif(("initialise" in line\_elem) and ("int" not in line\_elem) and ("float" not in line\_elem)): |
|  | line\_of\_code += "float " + line\_elem[1] + ";" #default type is float |
|  | variables.append("float") #storing var type in stack |
|  | line\_elem[1] = (line\_elem[1].split("="))[0] #need to store var name so tokenising at = in order to get name of var |
|  | variables.append(line\_elem[1]) #pushing var name into variables stack |
|  |  |
|  | elif(("initialise" in line\_elem) and (("int" in line\_elem) or ("float" in line\_elem))): |
|  | line\_of\_code += line\_elem[1] + " " + line\_elem[2] + ";" |
|  | variables.append(line\_elem[1]) #storing var type in stack |
|  | line\_elem[2] = (line\_elem[2].split("="))[0] #need to store var name so tokenising at = in order to get name of var |
|  | variables.append(line\_elem[2]) #pushing var name into variables stack |
|  |  |
|  | elif(("for" in line\_elem) and ("gap" not in (line\_elem[-1].split("="))[0])): #by default gap is 1 and since gap is not in line\_elem, it'll be considered at 1 only. |
|  | #print(line\_elem[2][-1], line\_elem[4]) |
|  | start\_for = (line\_elem[2].split("="))[-1] #tokenising at = to understand what the start value is |
|  | variables.append("int") #appending the type int to variables stack as we're assuming that the looping var is a newly defined temp one |
|  | variables.append((line\_elem[2].split("="))[0]) #tokenising at = to know what the var name is to push into stack |
|  | if(int(start\_for) < int(line\_elem[4])): #check to see whether loop is incrementing loop or decrementing loop |
|  | line\_of\_code += "for(int " + line\_elem[2] + "; " + (line\_elem[2].split("="))[0] + " <= " + line\_elem[4] + "; " + (line\_elem[2].split("="))[0] + "++)\n{" |
|  | else: |
|  | line\_of\_code += "for(int " + line\_elem[2] + "; " + (line\_elem[2].split("="))[0] + " >= " + line\_elem[4] + "; " + (line\_elem[2].split("="))[0] + "--)\n{" |
|  |  |
|  | elif(("for" in line\_elem) and ("gap" in (line\_elem[-1].split("="))[0])): #since gap is mentioned, the jumps of looping variable must be changed |
|  | start\_for = (line\_elem[2].split("="))[-1] #tokenising at = to understand what the start value is |
|  | variables.append("int") #appending the type int to variables stack as we're assuming that the looping var is a newly defined temp one |
|  | variables.append((line\_elem[2].split("="))[0]) #tokenising at = to know what the var name is to push into stack |
|  | if(int(start\_for) < int(line\_elem[4])): #check to see whether loop is incrementing loop or decrementing loop |
|  | line\_of\_code += "for(int " + line\_elem[2] + "; " + (line\_elem[2].split("="))[0] + " <= " + line\_elem[4] + "; " + (line\_elem[2].split("="))[0] + "+=" + (line\_elem[-1].split("="))[-1] + ")\n{" |
|  | else: |
|  | line\_of\_code += "for(int " + line\_elem[2] + "; " + (line\_elem[2].split("="))[0] + " >= " + line\_elem[4] + "; " + (line\_elem[2].split("="))[0] + "-=" + (line\_elem[-1].split("="))[-1] + ")\n{" |
|  |  |
|  | elif("while" in line\_elem): #check if while loop implementation |
|  | line\_of\_code += "while(" + line\_elem[-1] + ")\n{" #condition added in while |
|  |  |
|  | elif(("endfor" in line\_elem) or ("endwhile" in line\_elem)): #checking if for or while loop is ending |
|  | line\_of\_code += "}" |
|  | if("endfor" in line\_elem): #if for loop ending, pop out last 2 values as they are temp var type and name |
|  | variables.pop() |
|  | variables.pop() |
|  |  |
|  | elif(("if" in line\_elem)): |
|  | line\_of\_code += "if(" + line\_elem[1]+ ")\n{" |
|  | elif("else" in line\_elem): |
|  | line\_of\_code += "}\nelse\n{" |
|  |  |
|  | elif("fi" in line\_elem): |
|  | line\_of\_code += "}" |
|  |  |
|  | elif("print" in line\_elem): #print function implementation |
|  | line\_of\_code += "printf(\"" |
|  | for i in range(1,len(line\_elem)): #getting each word to be printed |
|  | if(line\_elem[i] in variables): #checking if print statement is referring to variables being printed |
|  | index\_var = variables.index(line\_elem[i]) #get index of that particular variable |
|  | line\_of\_code += "%" |
|  | if(variables[index\_var-1]=="int"): #checking one index before the var name to check var type in order to get right format specifier |
|  | line\_of\_code += "d\\n\"," + variables[index\_var] |
|  | if(variables[index\_var-1]=="float"): |
|  | line\_of\_code += "f\\n\"," + variables[index\_var] |
|  | break |
|  | else: |
|  | if(i==len(line\_elem)-1): #check if last word of string is being printed |
|  | line\_of\_code += line\_elem[i] + "\\n\"" #adding \n char for new line |
|  | break |
|  | line\_of\_code += line\_elem[i] + " " #spacing need for each word |
|  | line\_of\_code += ");" |
|  |  |
|  | elif("function" in line\_elem): #check for functions part |
|  | funcs.append(line\_elem[1]) #adding func name to funcs stack |
|  | return\_list.append(line\_elem) #storing the line\_elem list vals in return\_list to get return type |
|  | #temp = [] |
|  | #print(line\_elem) |
|  | line\_of\_code += line\_elem[3] + " " + line\_elem[1] + "(" #func defn |
|  | index\_arg = line\_elem.index("args") #check where args is indexed |
|  | #print(line\_of\_code) |
|  | for i in range(index\_arg+1,len(line\_elem), 2): #start going through list in gaps of 2 to get var type and name |
|  | func\_args += 1 #incrementing the func\_args by 1 |
|  | variables.append(line\_elem[i]) #pushing args to variables list for checking return value validity |
|  | variables.append(line\_elem[i+1]) |
|  | if(i+1 == len(line\_elem)-1): |
|  | line\_of\_code += line\_elem[i] + " " + line\_elem[i+1] + ")\n{" |
|  | break |
|  | line\_elem[i+1] = line\_elem[i+1].replace(",","") |
|  | line\_of\_code += line\_elem[i] + " " + line\_elem[i+1] + "," |
|  | #print(func\_args) |
|  | funcs.append(func\_args) #appending number of args to stack |
|  |  |
|  | '''temp = list(line\_of\_code) |
|  | temp[-1] = ")\n{" |
|  | line\_of\_code = "".join(temp)''' |
|  |  |
|  | elif(("return" in line\_elem) and ("print" not in line\_elem)): #check for return statement |
|  | line\_of\_code = "return" |
|  | for i in range(1,len(variables),2): #jumping through 2 in number to remove the commas at the end |
|  | variables[i] = (variables[i].split(","))[0] |
|  | if(line\_elem[1] in variables): #check if return var in variables stack |
|  | index\_return = variables.index(line\_elem[1]) #getting index of return var name |
|  | #print(return\_list) |
|  | if(variables[index\_return-1] == return\_list[0][3]): #checking types are same or not |
|  | line\_of\_code += " " + variables[index\_return] #then adding to line\_of\_code |
|  | else: |
|  | raise("the return type in function definition and variable type of returned value don't match!") |
|  | line\_of\_code += ";\n" |
|  |  |
|  |  |
|  | elif("endfunction" in line\_elem): #endfunction keyword check |
|  | i=1 |
|  | var\_len = len(variables)/2 |
|  | while(i<=var\_len): #popping out all function arg vars |
|  | variables.pop() |
|  | variables.pop() |
|  | i += 1 |
|  | line\_of\_code += "}" |
|  | #return\_list = [] #return list set to empty |
|  |  |
|  | elif("call" in line\_elem): #to call functions in main |
|  | num\_values = 0 |
|  | index\_values = 0 |
|  | #print(return\_list) |
|  | for func in return\_list: |
|  | #print(func, line\_elem[1]) |
|  | #if(line\_elem[1] in funcs): checking if func name is funcs stack |
|  | if('void' in func): |
|  | line\_of\_code += line\_elem[1] + "(" |
|  | #index\_num\_args = funcs.index(line\_elem[1]) + 1 |
|  | index\_values = line\_elem.index("values") |
|  |  |
|  | else: |
|  | line\_of\_code += return\_list[0][3] +" var = " + line\_elem[1] + "(" |
|  | index\_num\_args = funcs.index(line\_elem[1]) + 1 |
|  | index\_values = line\_elem.index("values") |
|  |  |
|  | for i in range(index\_values+1,len(line\_elem)): |
|  | num\_values += 1 |
|  | if(i == len(line\_elem)-1): |
|  | line\_of\_code += line\_elem[i] + ");" |
|  | break |
|  | line\_elem[i] = line\_elem[i].replace(",","") |
|  | line\_of\_code += line\_elem[i] + "," |
|  | #print(num\_values) |
|  |  |
|  | elif("" == line\_elem[0]): #if nothing exists then leave line |
|  | code\_file\_ptr.write("\n") |
|  | continue |
|  | else: #for normal statements like adding and all. |
|  | line = line.replace("\n","") |
|  | line\_of\_code += line + ";" |
|  |  |
|  | code\_file\_ptr.write(line\_of\_code+'\n') #writing line of code into file |
|  | #print(variables) |
|  | #print(funcs) |
|  |  |
|  | while(len(variables)>0): #popping out all variables type and name |
|  | variables.pop() |
|  |  |
|  | while(len(funcs)>0): #popping out all func names and no. of args |
|  | funcs.pop() |
|  | #print(variables) |
|  | #print(funcs) |
|  |  |
|  | code\_file\_ptr.write("}") #ending the code with a last } |
|  | code\_file\_ptr.close() #closing code file ptr. |
|  | file\_ptr.close() #closing file ptr |

**Cleaner file**

|  |
| --- |
|  |
| import os |
|  | import subprocess |
|  |  |
|  | #print(\_\_name\_\_) |
|  |  |
|  | def code\_cleaner(filename): |
|  | try: |
|  | temp\_file = filename[0:(len(filename)-2)] + "\_temp.c" #Adding the "\_temp.c" to filename |
|  | main\_file\_ptr = open(filename,"r") #Creating file pointer for the main code file |
|  | temp\_file\_ptr = open(temp\_file,"w") #Creating file pointer for temp file |
|  | count=0 #variable to keep count of the number of "{" or "}" |
|  | count\_close = 0 |
|  | count\_open = 0 |
|  | for line in main\_file\_ptr: |
|  | spaces = '\t'\*count #Giving count number of tabs from next line onwards |
|  | tab\_count = line.count('\t') |
|  | line = line.replace('\t'\*tab\_count,'') |
|  | #print(tab\_count) |
|  | if "{" in line: |
|  | count+=1 #incrementing count whenever "{" |
|  | #print count |
|  | count\_open +=1 |
|  | if "}" in line: |
|  | count-=1 #Decrementing count whenever "}" |
|  | spaces = '\t'\*count |
|  | count\_close +=1 |
|  | #print count |
|  |  |
|  | temp\_file\_ptr.write(spaces) #First writing spaces into every line |
|  | temp\_file\_ptr.write(line) #Then copy contents of every line from main code |
|  |  |
|  | os.remove(filename) #Deleting main code file |
|  | os.rename(temp\_file,filename) #Renaming the temp file with main code file |
|  |  |
|  | #print count\_close, count\_open |
|  |  |
|  | if((count\_close - count\_open) > 0): #Checking if } more than { |
|  | print("Looks like you have more number of \"}\" somewhere") |
|  | if((count\_close - count\_open) < 0): #Checking if { more than } |
|  | print("Looks like you have more number of \"{\" somewhere") |
|  | return count |
|  | except IOError: #If wrong file name gives |
|  | print("Sorry, but no such file exists in your current working directory") |
|  |  |
|  | def code\_execute(filename): |
|  | find = filename.split(".") |
|  | #print(find) |
|  | if(find[1] == 'c'): |
|  | compile\_command = "gcc " + filename + " -o " + filename[0:(len(filename)-2)] + "\_c" #Command for compiling with gcc along with filename and executable name |
|  | os.system(compile\_command) #Running the above command from script to compile the code |
|  | exec\_file = "./"+filename[0:(len(filename)-2)] + "\_c" #To execute file |
|  | os.system(exec\_file) |
|  | elif(find[1] == "cpp"): |
|  | compile\_command = "gcc " + filename + " -o " + filename[0:(len(filename)-4)] + "\_cpp" #Command for compiling with gcc along with filename and executable name |
|  | os.system(compile\_command) #Running the above command from script to compile the code |
|  | exec\_file = "./"+filename[0:(len(filename)-4)]+"\_cpp" #To execute file |
|  | os.system(exec\_file) |
|  | else: |
|  | print("Sorry, but can't execute the your " + find[1] + " code mate! :(") |
|  |  |

**CHAPTER 7**

**TESTING**

The purpose of testing is to identify the errors in the program.

Here is an example:

The pseudocode to be changes to C is as follows:

|  |
| --- |
| function test returns int with args int a, int b, int c |
|  | print test func |
|  | a++ |
|  | b++ |
|  | c++ |
|  | a = b + c |
|  | return a |
|  | endfunction |
|  |  |
|  | start |
|  | Initialise i=0 |
|  | Initialise int k=100 |
|  | For int i=0 to 10 |
|  | Print hey dude |
|  | k++ |
|  | k+=100 |
|  | Endfor |
|  | For int i=0 to 10 and gap=2 |
|  | While i<=5 |
|  | Print in loop 1 |
|  | Print wooho |
|  | i++ |
|  | Endwhile |
|  | Print success |
|  | k++ |
|  | Endfor |
|  | While i<=10 |
|  | Print here |
|  | Print there |
|  | Print everywhere |
|  | i++ |
|  | Endwhile |
|  | Print It's over mama |

**The output i.e. the output after converting to C language and executing is as follows:**

**The output i.e. the converted code in C is as follows:**

**CHAPTER 9**

**CONCLUSION**

**CONCLUSION:**

The SudoCode is a program used to help budding developers and programmers come up with a working logic without worrying about the programming language. It takes the pseudocode, which is written in a certain way, that the program takes it, and converts it a code in the language we desire.

It promotes code quality and readability. On a larger perspective, it helps save important time in industries, as one doesn’t need to waste time on synatactical errors and directly convert the pseudocode to program in our language.

**CHAPTER 10**

**Future Enhancement**

**FUTURE ENHANCEMENTS:**

In the near future, we hope to incorporate Machine Learning in this, and also more number of languages we can change the code into, in almost no time.

The languages we aim to convert the pseudocodes toare:

* C++
* Java
* Phython