If we start talking about stage 1. In this stage we consider collecting all the program data and aggregating it to make sure that it works the right way

Start :

Step1 –***f=fp.read()*** // Read contents of file as a string

Step2- ***Set spinlock and check\_function\_flag as False***

//They are synchronization construct

Step3 – ***a= code.split()*** //Split the code by spaces and store it as a list

Step4 – for each word in list repeat steps 5-

Step 5- ***p=words in KEYWORDS\_1;***

***x=words in KEYWORDS\_2;***

***y=words in KEYWORDS\_3;***

***z=words in KEYWORDS\_4;***

//check if a word belongs to any category of keyword and store the result

Step6-***if '<' and '>' in words:***

***Set self.name\_of\_library.add(words);***

***Set self.number\_of\_library+=1;***

Step 7- ***if words=='':*** //check for blank lines

***Set self.number\_of\_blankLines+=1;***

Step 8- ***if p or x or y or z:*** //check for keywords

***Set self.number\_of\_keywords=self.number\_of\_keywords+1;***

***Set self.name\_of\_keywords.add(words);***

Step 9- ***if p:***

***set spinlock=True***

***set self.types\_of\_variable.add(words);***

//setting spinlock to true means the word evaluated is a variable name and not a function.

Step 10- ***if (words not in KEYWORDS\_3) and ('(' in words or '()' in words):***

***Set temp=words[0:words.find('(')]***

***if temp not in IO\_FUNC:***

***Set self.name\_of\_functions.add(temp)***

***Set check\_function\_flag=True;***

//If the function is not a statement which requires ‘(‘ or ‘)’ i.e keyword3 then it is a function. Upon detecting a function we activate the function flag

Step 11- ***if check\_function\_flag and ')' in words:***

***Set check\_function\_flag=False;***

//On encountering ‘)’ we deactivate the function flag

Step 12- ***if check\_function\_flag and words not in KEYWORDS\_1:***

***Set self.number\_formal\_variables=self.number\_formal\_variables+1;***

// If words not in Keywords1 and check function flag is activate it indicates we have detected a formal variable

Step 13- ***if spinlock and ';' in words:***

***Set self.name\_of\_variables.add(words);***

***Set self.number\_of\_variables=self.number\_of\_variables+1;***

***spinlock=False;***

***if spinlock and (',' in words or '=' in words) and not check\_function\_flag:***

***if ',' in words:***

***set words=words[0:words.find(',')]***

***if '=' in words:***

***set words=words[0:words.find('=')]***

***if ';' in words:***

***set words=words[0:words.find(';')]***

***print(words)***

***if ',' in words and len(words)>2:***

***set self.name\_of\_variables.add(words.split(','))***

***set self.number\_of\_variables=self.number\_of\_variables+1;***

***set self.name\_of\_variables.add(words);***

//If spinlock is active and words contains a termination ‘;’ it suggests a variables. Else if word contains ‘=’ then it a variable or else if words preceded by data type keywords then it’s a variable and can be split by ‘,’

Stop

Algorithm to compute the bind score

Start

Step 1- ***Set bind=bind+(self.number\_of\_variables\*1)***

Step 2***- Set bind=bind+(self.number\_of\_functions\*\*2)#sqaure***

Step 3***- Set bind=bind+(self.number\_formal\_variables\*4)***

Step 4***- bind=bind+(self.number\_of\_keywords\*2)***

Step 5***- bind=bind+(self.number\_of\_blankLines\*1)***

Step 6***- bind=bind+(self.number\_of\_library\*1)***

Algorithm to create walkthrough simulation data

Step 1- Start

Step2- Declare Stack operation\_stack and temp\_stack and a

Step 3- for each word in a repeat steps 3-

Step 4- ***if '{' in words:***

***self.operation\_stack.append(('{'))***

***self.temp\_stack.append(i);***

//If we encounter the start of a block we store the block in a stack. Stack is used to handle nested blocks. For each nested state we store the value index value in splitted code. When we encounter ‘}’ which essentially means the end of a inner nested block or otherwise.

Step 4- ***if '}' in words:***

***x=self.operation\_stack.pop();***

***y=self.temp\_stack.pop();***

***progressive\_count=i-y;***

***self.simulation\_data.append((x,progressive\_count))***

***progressive\_count=0;***

//For each nested state we store the value index value in splitted code. When we encounter ‘}’ which essentially means the end of a inner nested block or otherwise. We calculate the different between the start and end points in block code. This differences captures the span of each block and inner nested blocks. The results are appended to simulation data.

Algorithm to find clusters