Journal Homepage: - www.journalijar.com
INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/ 4003
DOI URL: http://dx.doi.org/10.21474/IJAR01/ 4003

RESEARCH ARTICLE

CONTEXT AWARE TEACHING AID CEM FOR LEARNING COMPILER DESIGN CONCEPT FOR STUDENTS WITH LEARNING DISABILITY.

Mrs. Sini Anna Alex[1] and Dr. Anita Kanavalli[2].
1. Assistant Professor, Department of Computer Science and Engineering, M.S.R.I.T, Bangalore.
2. Professor, Department of Computer Science and Engineering, M.S.R.I.T, Bangalore.

Manuscript Info

Manuscript History

Received: 14 February 2017
Final Accepted: 09 March 2017
Published: April 2017

Key words:-
Lexical, Syntax Directed, Annotated Parse Tree, Directed Acyclic Graph

Abstract

Context awareness (CA) refers to systems can both sense and react based on their environment. The systems have to collect data based on the circumstances under which they are able to operate and adapt their behavior based on rules specified. Such systems are a component of a mobile computing environment. Currently context has been considered as part of a process in which users are involved, hence specifying and developing context models for designing a compiler and to develop context-aware applications to support and adapt interfaces or connectivity to all the phases.

We present a complete description of a project for a compiler called Compiler Evaluator Module (CEM) that generates intermediate representations for each of the phases in the design of a compiler. A Table driven parsing model for Lexical and Syntax Analysis phase. An annotated parse tree for a given Syntax Directed Definition using suitable data structures and algorithms. The syntax-directed translation techniques will be applied to type checking and intermediate-code generation. The main application is the construction of syntax trees. Since some compilers use syntax trees as an intermediate representation, a common form of Syntax Directed Definition turns its input string into a tree. To complete the translation to intermediate code, the compiler may then make the syntax tree, using another set of rules that are in effect an Syntax Directed Definition on the syntax tree rather than the parse tree. All these translations are designed through game based learning[3] for the students with Learning Disabilities like slow learners, hyper active students who can't concentrate.

Introduction:-

Context-aware [1] mobile systems are concerned with the acquisition of context data, understanding of context data, and application based on the recognized data in the context. In some applications, as the user's activity is crucial in collecting the data, context awareness has been concentrated more deeply on activity recognition. Three important aspects of context are what type of information is required and from where the data is available.

The context [1] data differs from one module to the other. Mostly the data is given to the system as a whole and according to the need of the resources in each and every module the data is undergone various procedures for a

Corresponding Author:- Mrs. Sini Anna Alex.
Address:- Assistant Professor, Department of Computer Science and Engineering, M.S.R.I.T, Bangalore.

change. We need to develop new sub-compiled systems which are in need for developing the final compilation. The problem is complicated as we required making many intermediate representations. The systems which are available are lacking of perfection based on context data. Here Context Aware [1] Teaching Aids means the generators developed for developing each phase of compiler.

The construction of compiler is a very time-consuming and tiresome process. A lot of efforts is required in focusing on how to implement high level constructs. The typical way of building a compiler is o split the compilation into several phases with well defined interfaces. Conceptually these phases operate in sequence, but in practice they are often interleaved. Each phase taking the output from the previous phase as its input. The first phase takes the input as a high level programming language a s a whole. The theory of formal languages developed concurrently with the theory of compilers [2]. This helps in the identity which patterns were beyond the power of a particular formalism. The clear understanding of context free grammars and languages is required for building the system. These characteristics necessitate the redesigning of applications in order to maintain good quality of service for execution.
.
An outline for the lexical analysis phase is it takes raw input and converts into a flow of a set of tokens. The lexical analyzer is the only phase that processes input character, so speed is so critical here. Here using a tool Compiler Evaluator Module (CEM) we produce a table driven lexical analysis for high level programming languages like C, C++ and JAVA. By hand for the better understanding of concepts we introduced Activity Learning through Think-Pair-Share strategy for all the phases. For the next phase Syntax Analysis, Context Free grammars are used to specify the syntax of formal languages. While detecting and correcting the errors the meaning of the grammar should be preserved. The parsing concepts are well defined for the students using Game-Based Learning[3].

Each phase has a well defined functionality. Analysis-Synthesis model checks for the representation and the usage of the identifiers. For the easy analysis to the development of machine code an intermediate code is generated as the next phase. The next task is to implement the code improvement techniques which can be referred as optimizations. Finally the translation to the target code is done.

Related Work:-

ParsingEmu:-

A simulator tool called ParsingEmu [4] is already implemented using context free grammar as input. This tool helps the learner or students to understand the concepts up to a point. But some limitations are there for this tool. For the students they should know at least the important part for the process of parsing since the initial process is not implemented in this simulator. For the top down parsing process the Context free grammar has to made suitable since some restrictions are imposed on grammar. Those restrictions the user has to do by hand for giving input to this simulator. That is a major drawback. Another limitation is for the bottom up parsing strategy the augmented grammar is not introduced in the ParsingEmu Simulator. In our simulator we overcome all the limitations Parsing Emu had and introduced additional modules for the other phases of compiler.

Grammatica:-

In simulator tool Grammatica [5] they have used a command line interface. It is a parser generator where lexical analyzer and syntax analyzers are made. The input should be complete grammar file and Java source code for creating a parser. Grammatica have Information about the automatic error recovery mechanisms.

It handles grammar ambiguities and the tokens and the syntax of the production declarations in the grammar is checked. This also will work for only specific set of grammar components not all the Context Free Grammars generated.

Jaccie:-

The Simulator tool Jaccie [6] is an acronym for Java based compiler with interactive environment. Jaccie includes a scanner generator and a variety of parser generators that can generate LL(1), SLR(1), LALR(1) grammars. It has a debugging mode where you can operate it non-deterministically. Jaccie won’t build a compiler generator. It is only a parser generator. In CEM we introduced all the six phases of compiler and it is a complete compiler generator.

Implementation:-

Intervention: Why CEM is named as Context Aware Teaching Aid?

CEM is a simulator tool which will do its task of generating compiler design process. In CEM we involved a Game based Teaching strategy [3] which helps students with Learning disorder. Those students experienced the fun in understanding the most difficult concept through well known games. Here the context [1] is to help students with Learning Disability (Disorder). Helping strategy is implemented through games.

Features of CEM (which is not in any other compiler generators)

CEM includes:-

1. Lexical Generator for C, C++, JAVA
2. Esoteric Programming Language generator
3. An emulator for Predictive Parsing,SRP, LR Parsers
4. Emulator for HTML to CSS+HTML
5. Code optimizer Generator with the help of macros.
6. Parser Generator for gcc compiler for analyzing segmentation faults
7. Math Compiler
8. DAG and Syntax Tree Generator for arithmetic expression statements, array addressing and for control structures.
9. Syntax Analyzer for XML scripts
10. Code generator for simple arithmetic expressions.
11. Generator for identifying the handles in a given input string for C
12. Error Detector and Recover emulators for handling lexical, syntax and linker errors for c programs (vi editor).
13. Eclipse plug-in for browsing the results of a type state analysis, to help programmers determine where a program violates temporal specifications.
14. Eclipse plugins using interactive interface for browsing the results of an analysis.
15. Eclipse plug-in to analyze and determine where a program violates temporal specifications.
16. Eclipse plug-in, including browsing of the run-time type information that was used to resolve dynamic dispatch.
17. Eclipse plug-in to find the scope/visibility of the variables used.
18. Mobile App for Parser Generator

Learning Algorithms:-

Ubiquitous Lexical Analyzer:-

For generating a Lexical Analyzer [2] the input should the high level language(C, C++ or JAVA). The main tasks of the lexical analyzers are done in our Tool Compiler Evaluator Module (CEM). CEM will eliminate all whitespaces, comment statements, it will track line numbers. Separate header files from program fragment and finally it will tokenize the input.

The Algorithm 3a describes the table driven lexical analysis process.

Algorithm 3a: Table driven lexical analysis process:-

1. The Input to a Lexical Analyzer [2] is a High Level Language (HLL) program.
2. The HLL is scanned character by character and specify them into a well known common character set called as tokens.
3. The token description varies from type to type.
4. A symbol Table is maintained to collect the actual values of the tokens, its representations etc.
5. The Lexical Analyzer will generate a table with fields token, attribute value and description as output.
6. We developed a tool called CEM which will take any kind of HLL program as input and tokenize. CEM is not Language Specific.

Table 1:- Table Driven Lexical Analyzer.

Token	**Attribute**	**Description**
Id	The pointers in symbol table, mostly numerical value	Start with letter followed by letter or digit
Digit	The pointers in symbol table, mostly numerical value	Numerical values
Literal	“%d”	Any Text in quotes
Relop	<,>,<=,>=	Relational operators

Syntax and Semantic Analyzers:-
A mobile App is made on Android for the parsing techniques. The module is also introduced in CEM.

Algorithm 3b:- Construction of NonRecursive Top Down Parsing (TDP)
NonRecursive TDP [2] is called as Predictive Parsing. It won't support backtracking.

1. Make the CFG suitable for TDP
2. If CFG is Left Recursive resolve by eliminating it
   If $S \rightarrow Sa|b$ is there replace with
   $S \rightarrow bA$ and $A \rightarrow aS|\varepsilon$
3. If Left factoring is required do it by using finding common factors from different grammar productions. $S \rightarrow he|her$ is replaced with $S \rightarrow heP$ and $P \rightarrow \varepsilon|r$
4. Then Compute the First and Follow: First of RHS of a production if terminal, then terminal itself or if nonterminal the FIRST of the nonterminal. If ε is in First then the first of next nonterminals.
5. Follow of a Nonterminal is the terminals come after the Nonterminal. For Start Symbol Follow is $.
6. Construct a Table driven Parsing Table using the data structure – multidimensional arrays or structures
7. Parsing is implemented by either using Stack or Linked list. Check whether the given input is accepted or not by the constructed Parsing Table.

Algorithm 3c:- Construction of Bottom Up LR Parsers
LR parsers [2] can be implemented very efficiently. Finding a reachable Item sets is the first step in the process of LR parsing.

1. Begin setting the Augmented Grammar. Start doing the parsing for LR (0).
2. Find the set of reachable items in each of the itemset.
3. Compute the set of states that can be generated and the transition between them.
4. Construct the resultant Transition Table using two modules – Action and Goto.

Action module:-
If the transition is through terminal check for the actions. If action is to shift then shift input symbol and move to next state.
If action is reduce then identify the handle given and reduce the no of transitions.
If action is falling towards accept then the transition successfully admits the augmented grammar.

Else:-
Receive an error as the Action.
The parsing can't move further.
Terminate the syntax Analysis

5. If error, identify the type of conflict occurred and proceed further with the next LR parser- CLR or LALR.
6. The same process will continue with slight modifications. For CLR a second component is made and for LALR the transition items from CLR is combined iif the first component is same.

Intermediate Representers for Context Information:-
Intermediate representations are implemented using various data structures. Lexical Analyzer [2] is constructed by using Record or structure. Syntax and Semantic Analyzers [2] are designed with the help of stack, linked list and structure.

Target Optimized Context Code:-
Plug-in for JAVA programming language is introduced for improving the Semantic Analysis in JAVA programming language. In CEM a plug-in for JAVA Semantic Analyzer is made. Code Optimizer is generated for eliminating the statements not available for next use, for eliminating common sub expressions, eliminating dead code and for replacing expensive statements with cheaper statements.

Research Method:-
Research Design: Implemented algorithms and mathematical models for various types of High Level Programming strategies.

Experiments performed to evaluate strategy/ tool: For strategy the execution is done by hand and verified. A set of test cases generated using different high Level Languages, scripting languages for the tool. For all the test cases each module is tested. Activity based learning, Game based Learning [3], Project based learning [3], TPS activity[7][8].

Ubiquitous Lexical Analyzer Game Based Learning Strategy:-
We included the idea of tokenization using the games TIC-TAC-TOE, ODD MAN OUT and Match the following. The tokens or lexemes or patterns have to be identified from a given set of solutions using the above games. The generation of tokens and finding matching patterns are done by using a game called Omit_Snake.

Syntax and Semantic Analyzers Game Based Learning Strategy:-]
We introduced the gaming [3] concept for the better understanding of the parsing process. The Games like Snake and Ladder, Hill Climbing, Trees and Bees etc are done for making the concept very interesting.

The game Trees and Bees is illustrated in Fig 3.1.
This game illustrates the process of building the item sets for LR parsers. How to add the closure of the item is clearly explained through this game. The game Snake and Ladder is illustrated in Fig 3.2.

Through this game we are trying to focus more on the parsing process when an input is given. Initially the input value is at the beginning point of the game. When parsing proceeds the shift action will move to next and when a handle occurs the snake swallow the handle. After swallowing it will start from the next value and the parsing continues.

The game Hill Climbing is another strategy for the parsing process. Students were divided into groups and they did many such gaming strategies and understood the process well.

Fig 3.1:- Trees and Bees.

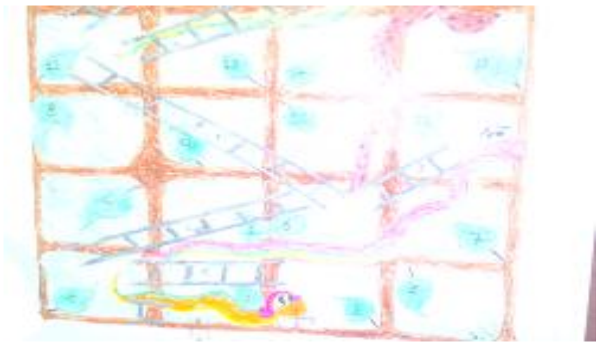


Fig 3.2:- Snake and Ladder.

The Module is very well executed using Think-Pair-Share (TPS)[7] technique implemented on Games. TPS helped the students to come up with many innovative game based strategies. Even the weaker students in the class enjoyed the activity and well executed.

Intermediate Representers for Context Information Game Based Learning Strategy:-

For Intermediate Representations, we designed a game plan of showing how a LAYS chips is manufactured in Fig 3.4. Through this game the initial process is to take potato peel it and give it to the cutting machine. It is cut properly into pieces and given to the next machine to fry it. After frying the other ingredients will get added. Each of these Lays chip manufacturing unit was designed by the Intermediate representations. Similarly the compiler phases also works. Initially Lexical analyzer will generate tokens. Then parser will check for the errors by combining various modules. Finally semantic analyzer will check whether any additional modifications required.

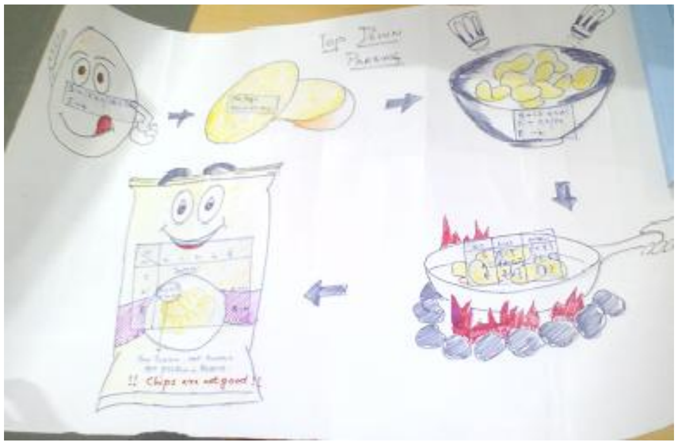


Fig 3.4:- LAYS Manufacturing Model

Results:-

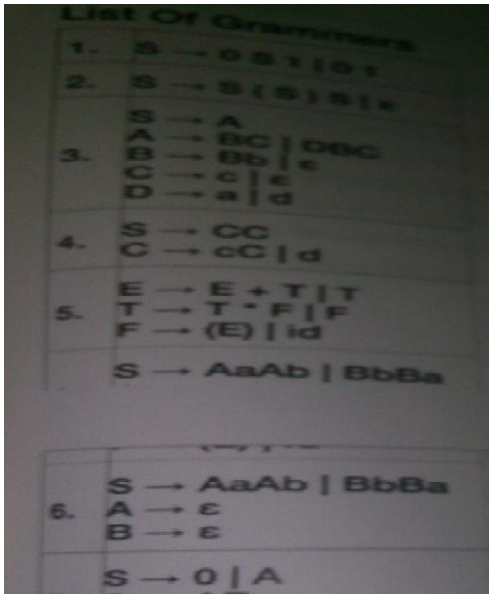
What are the main modules in its implementation?

Lexical Analyzer, Parser, Semantic Analyzer, Intermediate Representer, Code Optimizer, Code Generator, Math Compiler, Script Compiler, Eclipse Plug-in.

What all testing has been reported?

Testing based on High Level Programming Language, context free grammars, Esoteric languages, Intermediate outputs.

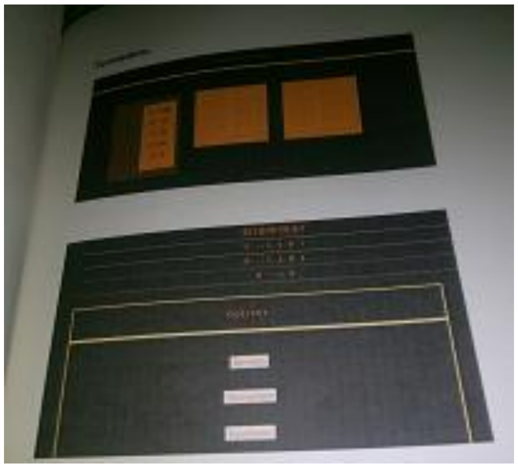
Evidence collected: Snapshots, Executable code for each modules, Activity charts, Mid-Term and Course End Survey Forms.

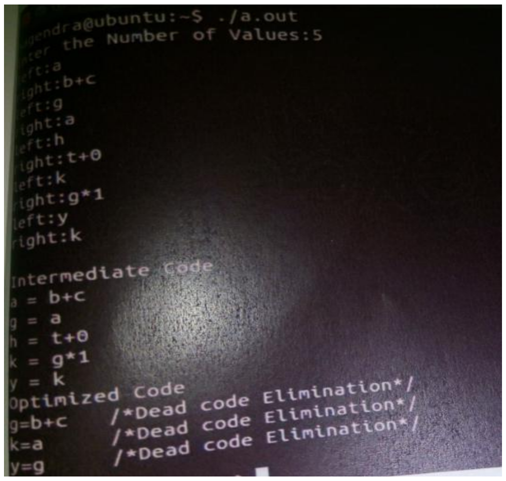


List Of Grammers

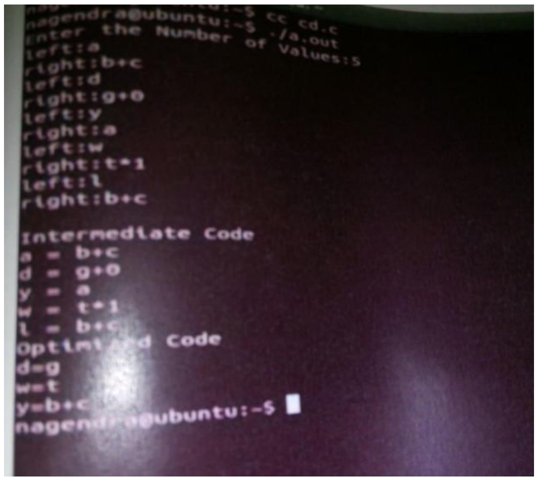
1.	S → 0 S 1 | 0 1
2.	S → S (S) S | ε
3.	S → A A → BC | DBC B → Bb | ε C → c | ε D → a | d
4.	S → CC C → cC | d
5.	E → E + T | T T → T * F | F F → (E) | id
	S → AaAb | BbBa

6.	S → AaAb | BbBa A → ε B → ε
	S → 0 | A

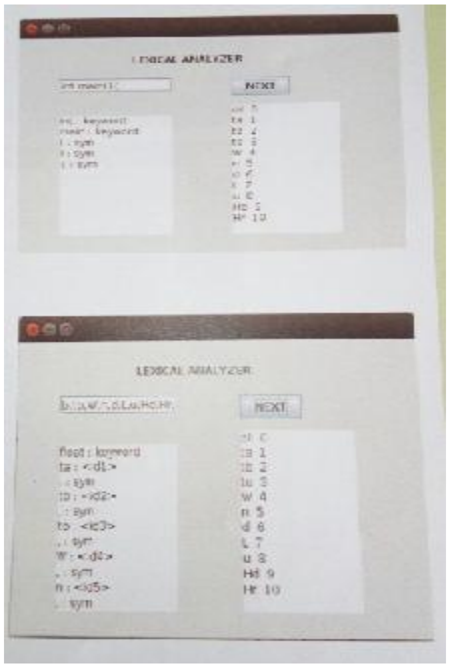




```
gendra@ubuntu:~$ ./a.out
nter the Number of Values:5
eft:a
ight:b+c
eft:g
ight:a
eft:h
ight:t+0
eft:k
ight:g*1
eft:y
ight:k

Intermediate Code
a = b+c
g = a
h = t+0
k = g*1
y = k
Optimized Code
g=b+c     /*Dead code Elimination*/
k=a       /*Dead code Elimination*/
y=g       /*Dead code Elimination*/
```

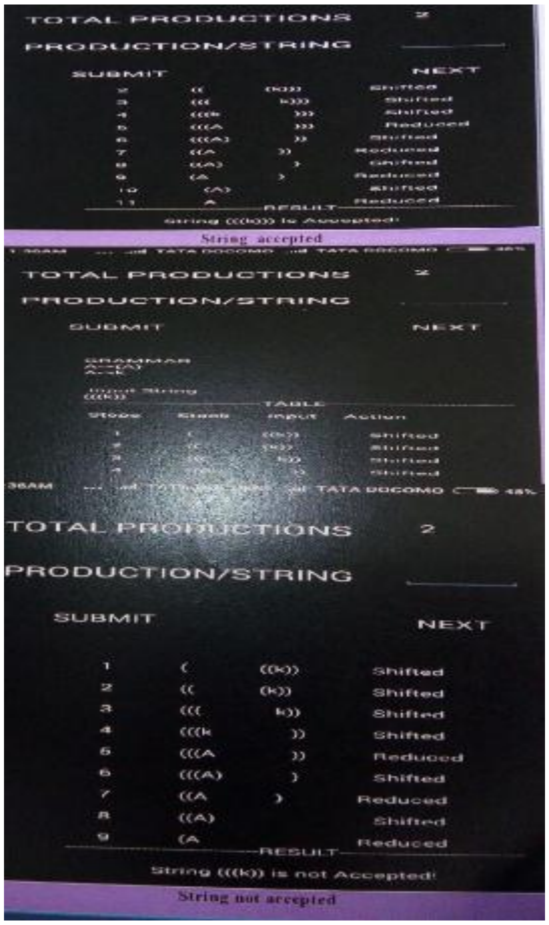


```
nagendra@ubuntu:~$ cc cd.c
nagendra@ubuntu:~$ ./a.out
Enter the Number of Values:5
left:a
right:b+c
left:d
right:g+0
left:y
right:a
left:w
right:t*1
left:l
right:b+c

Intermediate Code
a = b+c
d = g+0
y = a
w = t*1
l = b+c
Optimized Code
d=g
w=t
y=b+c
nagendra@ubuntu:~$
```



```
LEXICAL ANALYZER

int main(){                      NEXT

int : keyword                    [illegible]
main : keyword                   ta 1
( : sym                          tb 2
) : sym                          tc 3
{ : sym                          w 4
                                 n 5
                                 d 6
                                 L 7
                                 u 8
                                 Hd 9
                                 Hr 10
```

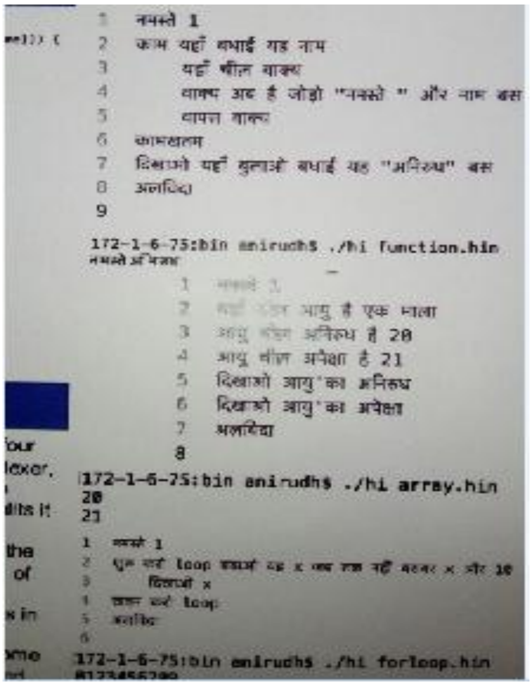
```
LEXICAL ANALYZER

[illegible]                      NEXT

                                 [illegible] 0
float : keyword                  ta 1
ta : <id1>                       tb 2
, : sym                          tc 3
tb : <id2>                       w 4
, : sym                          n 5
tc : <id3>                       d 6
, : sym                          L 7
w : <id4>                        u 8
, : sym                          Hd 9
n : <id5>                        Hr 10
, : sym
```



```
TOTAL PRODUCTIONS          2
PRODUCTION/STRING          ______
   SUBMIT                        NEXT
      2      ((        (k)))      Shifted
      3      (((       k)))       Shifted
      4      (((k       )))       Shifted
      5      (((A       )))       Reduced
      6      (((A)      ))        Shifted
      7      ((A      ))          Reduced
      8      ((A)       )         Shifted
      9      (A        )          Reduced
      10      (A)                 Shifted
      11      A                   Reduced
   ----------------------RESULT----------------------
           String (((k))) is Accepted!
                 String accepted
```

```
TOTAL PRODUCTIONS          2
PRODUCTION/STRING          ______
   SUBMIT                        NEXT

     GRAMMAR
     A-->(A)
     A-->k

     Input String
     (((k))
   ----------------------TABLE----------------------
     Steps     Stack     Input     Action
       1        (        ((k))      Shifted
       2        ((       (k))       Shifted
       3        (((      k))        Shifted
       4        (((k     ))         Shifted
```

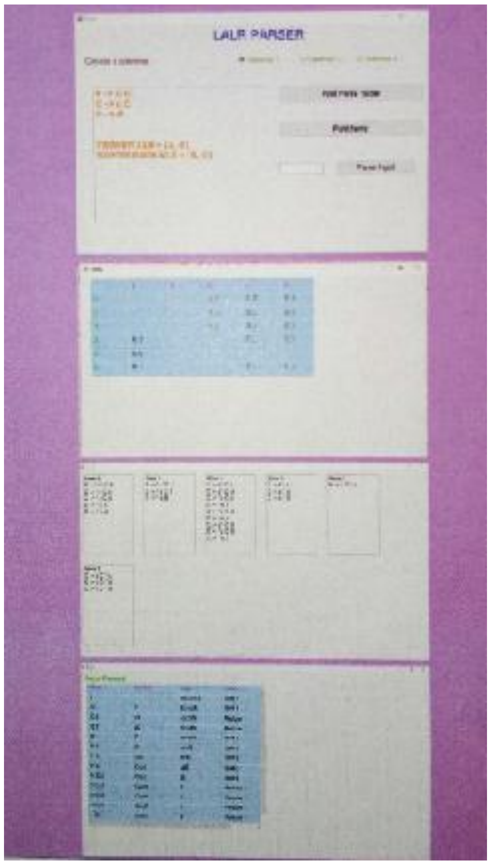
```
TOTAL PRODUCTIONS          2
PRODUCTION/STRING          ______
   SUBMIT                        NEXT

      1      (        ((k))       Shifted
      2      ((       (k))        Shifted
      3      (((       k))        Shifted
      4      (((k       ))        Shifted
      5      (((A       ))        Reduced
      6      (((A)       )        Shifted
      7      ((A      )           Reduced
      8      ((A)                 Shifted
      9      (A                   Reduced
   ----------------------RESULT----------------------
           String (((k)) is not Accepted!
                 String not accepted
```



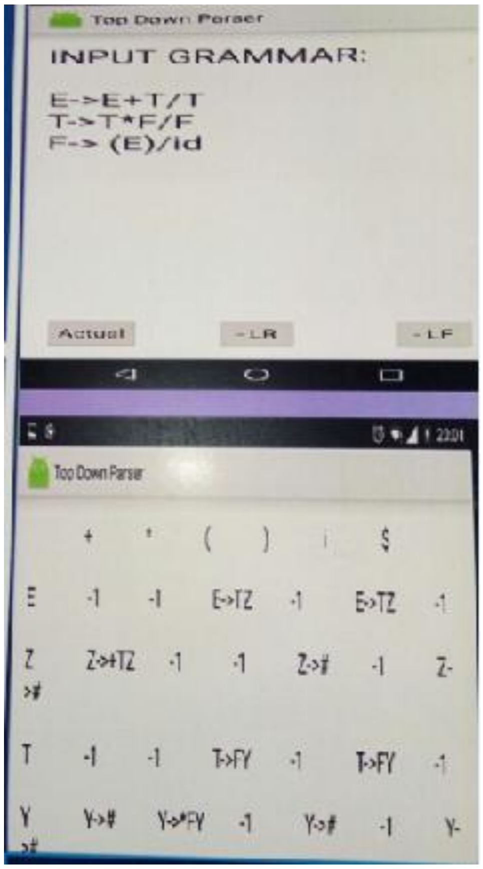
```
नमस्ते 1
काम यहाँ बधाई गड नाम
     यहाँ शील वाक्य
     वाक्य अब है जोड़ो "नमस्ते " और नाम बस
     वापस वाक्य
कामखतम
दिखाओ यहाँ बुलाओ बधाई यह "अनिरुध" बस
अलविदा

172-1-6-75:bin anirudh$ ./hi function.hin
नमस्ते अनिरुध
```

```
नमस्ते 1
यहाँ [illegible] आयु है एक माला
आयु [illegible] अनिरुध है 20
आयु शील अपेक्षा है 21
दिखाओ आयु'का अनिरुध
दिखाओ आयु'का अपेक्षा
अलविदा

172-1-6-75:bin anirudh$ ./hi array.hin
20
21
```

```
नमस्ते 1
[illegible] loop [illegible] x [illegible] x और 10
    दिखाओ x
[illegible] loop
अलविदा

172-1-6-75:bin anirudh$ ./hi forloop.hin
0123456789
```


LALR PARSER
Parse Input

SUBMIT
ENDTAG



```
INPUT GRAMMAR:

E->E+T/T
T->T*F/F
F-> (E)/id
```

```
      +       *       (       )       i       $

E     -1      -l      E->TZ   -1      E->TZ   -1

Z     Z->+TZ  -1      -1      Z->#    -1      Z-
>#

T     -l      -1      T->FY   -1      T->FY   -1

Y     Y->#    Y->*FY  -1      Y->#    -1      Y-
>#
```

Discussion:-

The algorithms described in implementation part and the mathematical models prescribed are able to build a new system capable of monitoring the compiled systems more accurately. The efficiency in the working of the systems is computed by giving various categories of inputs.

We executed the idea by teaching the students through game based learning [3], by building mobile app and by developing a toolkit. The Design of a compiler includes six phases. Each phase was taught by incorporating different gaming strategies, making game chart and by developing toolkit.
We executed by using both instructional strategies and by developing a set of tools for each phase of compiler.

Using Instructional Strategies:-
Execution of Idea Instructors job:-
Instructor divided the students into groups. Discussed the problem definition with the students. Given them some time to Think and Pair. After students share their ideas. Instructor given a feedback for the ideas and came up with a numerous set of interesting games. Divided the students into groups. Given clear Instructions on the overall view of how the phase will work and motivated the students to think on implementing the same through a game.

For developing tool the instructor helped in giving the ideas what should be the input and how the expected output should look like. Instructor helped in giving suggestions for improving the tool design and given ideas in the usage of various data structures.

Execution of Idea Students job:-
Students in each group discussed for a time given for Think Phase and Pair Phase. After that they shared their ideas. They came up with a set of ideas for implementing the task of the phases through a game. They discussed with the instructor, taking input from the instructor they build nice game models and charts. For developing CEM each group of students working in different modules of CEM sit together and discussed about the format and the input output procedures.

Improvement using the idea:-
We collected midterm and course end survey forms, Individual feedback from set of students. Out of 120 students 115 students given excellent feedback and they requested for learning some other courses also using this strategy.

We have implemented and also evaluated the performance of subsystems as a single unit by giving varying models of input high level programs. All the individual subsystems are working perfectly and accurately. We have to implement the connectivity as a whole for building the final compiler. The data structures used for the implementation of each individual subsystem have to be connected using a well behaved connector.

Limitations of CEM are that each module is built separately, we need to connect each module using interfaces or connectors. For XML and HTML code generation phase is not built. It is very vast to build the final phase for these languages.

Conclusion:-
This context aware [1] Compiler Evaluator Module (CEM) simulator is module based and is applied to generate a context aware compiler as a whole in a large programming environment. We defined a set of generators for evaluating the performance of each phases of the design of a compiler using Structured Data and Game Based Learning approach. We transformed the basic compiler design statics into an enhanced, easy and portable set of modules. Now we have to find proper interfaces to connect all the generators and to generate a compactable and well designed generator. Compared to all the other simulator tools available our proposed generator is very efficiently doing the task with a very clear and understandable manner.

The proposed Compiler Evaluator Module (CEM) checks some additional tasks also. We incorporated the additional Semantic Analysis tasks in to the generator. Finally output of the generator is a very optimized and efficient target code.

References:-

1. Context systems
2. Compiler phases
3. Game based learning
4. Parsing Emu
5. Grammatica: Reference manual for Grammatica version 1.6.3.
6. Jaccie
7. TPFOSSS: A Modified TPS Technique to Improve Student's Conceptual understanding of Compiler Construction Course, Sunita Dol, Dattatray Gandhmal, 6th IEEE International Conference on Technology for Education, December 18-21, 2014
8. Effect of Think-Pair-Share in a Large CS1 Class: 83% Sustained Engagement, Aditi Kothiyal, Rwitajit Majumdar, Sahana Murthy, Sridhar Iyer, ICER'13, August 12–14, 2013, San Diego, California, USA.